Bayesian Hierarchical wOBA Projection

Max Wassarman

2025-08-05

Overview

Implementation of Bayesian hierarchical model to project wOBA. Combines Marcel projections with wOBA components and uses player, season and team random effects.

Dependencies

```
library(tidyverse)
library(Metrics)
library(brms)
library(splines)
library(corrplot)
library(bayesplot)
library(rstan)
library(posterior)
library(cmdstanr) # Note: install_cmdstan() required for first-time setup
library(gt)
library(gtExtras)
```

Data Loading

```
# Load FanGraphs data and filter for players with sufficient plate appearances
data <- read_csv("fangraphs_04-24.csv") |>
  filter(PA > 50)
```

Marcel Projection

Weighted average of recent performance with regression towards league mean. More recent seasons get higher weights (5/4/3), and players with fewer plate appearances are regressed more heavily toward league average.

Data Preparation

```r
# Split data into historical and current
historical_data <- data |>
  filter(Season < 2020) |>
  select(IDfg, Season, Name, Team, Age, AB, PA, wOBA)

current_data <- data |>
  filter(Season < 2024) |>
  select(IDfg, Season, Name, Team, Age, AB, PA, wOBA)
```

League Average Calculation

```r
league_avg <- historical_data |>
  group_by(Season) |>
  summarize(lg_wOBA = mean(wOBA, na.rm = TRUE))
```

Marcel Projection Implementation

```r
model_data <- historical_data |>
  arrange(IDfg, Season) |>
  group_by(IDfg) |>
  mutate(
    # Create lagged variables for previous seasons
    wOBA_prev1 = lag(wOBA, 1),
    PA_prev1 = lag(PA, 1),
    wOBA_prev2 = lag(wOBA, 2),
    PA_prev2 = lag(PA, 2),
    wOBA_prev3 = lag(wOBA, 3),
    PA_prev3 = lag(PA, 3),

    # Count years of avalible data
    years_of_data = (!is.na(wOBA_prev1)) +
                    (!is.na(wOBA_prev2)) +
                    (!is.na(wOBA_prev3))
  ) |>
  ungroup() |>
  left_join(league_avg, by = "Season")

# Calculate weighted averages based on avalible data
model_data <- model_data |>
  mutate(
    # 3 year
    wOBA_marcel_3yr = case_when(
      years_of_data >= 3 ~ (5*wOBA_prev1*PA_prev1 + 4*wOBA_prev2*PA_prev2 + 3*wOBA_prev3*PA_prev3) /
                      (5*PA_prev1 + 4*PA_prev2 + 3*PA_prev3),
      TRUE ~ NA_real_
    ),

    # 2 year
    wOBA_marcel_2yr = case_when(
```

```r
      years_of_data >= 2 ~ (5*wOBA_prev1*PA_prev1 + 4*wOBA_prev2*PA_prev2) /
                      (5*PA_prev1 + 4*PA_prev2),
      TRUE ~ NA_real_
    ),

    # 1 year
    wOBA_marcel_1yr = case_when(
      years_of_data >= 1 ~ wOBA_prev1,
      TRUE ~ NA_real_
    ),

    # Get correct calculation based on available data
    wOBA_marcel_temp = case_when(
      years_of_data == 3 ~ wOBA_marcel_3yr,
      years_of_data == 2 ~ wOBA_marcel_2yr,
      years_of_data == 1 ~ wOBA_marcel_1yr,
      TRUE ~ NA_real_
    ),

    # Get pa for regression to mean
    pa_regression = case_when(
      years_of_data == 3 ~ PA_prev1 + PA_prev2 + PA_prev3,
      years_of_data == 2 ~ PA_prev1 + PA_prev2,
      years_of_data == 1 ~ PA_prev1,
      TRUE ~ 0
    ),

    # Players with fewer PA regressed more toward league average
    reg_weight = pa_regression / (pa_regression + 1500),

    # Final Marcel calculation
    wOBA_marcel = reg_weight * wOBA_marcel_temp + (1 - reg_weight) * lg_wOBA
  ) |>
  filter(!is.na(wOBA_marcel))
```

Component Analysis

A simple effort to help capture player skills that may not be fully reflected in wOBA alone.

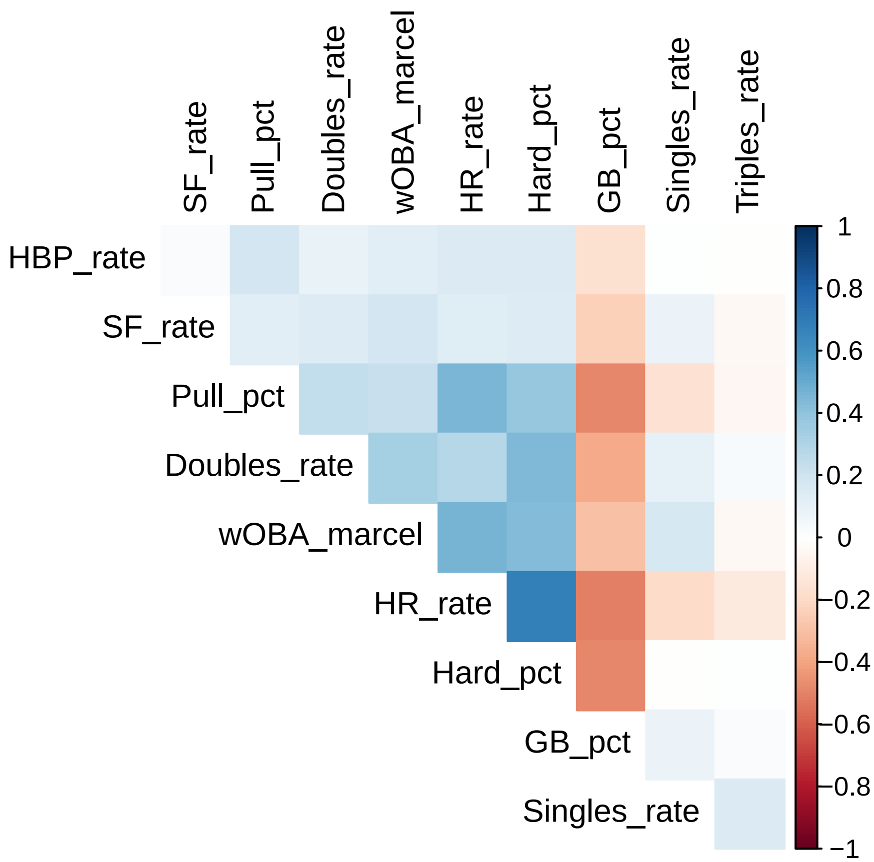
Component Data Preparation

```r
# Join component statistics to Marcel projections
component_data <- model_data |>
  left_join(data |>
      select(IDfg, Season,
             AB, PA, BB, SO, `1B`, `2B`, `3B`, HR, HBP, SF,
             `K%`, `BB%`, `Hard%`, `Pull%`, `GB%`),
    by = c("IDfg", "Season")
  ) |>
  mutate(
```

```r
    # Calculate rate statistics per at-bat
    Singles_rate = `1B` / AB.x,
    Doubles_rate = `2B` / AB.x,
    Triples_rate = `3B` / AB.x,
    HR_rate = HR / AB.x,
    HBP_rate = HBP / AB.x,
    SF_rate = SF / AB.x,
    Hard_pct = `Hard%`,
    Pull_pct = `Pull%`,
    GB_pct = `GB%`
  )
```

Correlation Analysis

```r
# Examine correlations between Marcel projections and component statistics
cor_matrix <- component_data |>
  select(
    wOBA_marcel,
    Singles_rate, Doubles_rate, Triples_rate, HR_rate, HBP_rate, SF_rate,
    Hard_pct, Pull_pct, GB_pct
  ) |>
  cor(use = "pairwise.complete.obs")

# Visualize correlation structure
corrplot(cor_matrix, method = "color", type = "upper",
         order = "hclust", diag = FALSE, tl.col = "black")
```



Component Residuals

```r
# Create linear models to predict each component from Marcel projection
# Residuals capture component-specific skills independent of overall performance
options(scipen = 999)

# Base relationship: wOBA vs Marcel projection
print("Base wOBA ~ Marcel relationship:")
```

```
## [1] "Base wOBA ~ Marcel relationship:"
```

```r
print(summary(lm(wOBA ~ wOBA_marcel, data = model_data)))
```

```
##
## Call:
## lm(formula = wOBA ~ wOBA_marcel, data = model_data)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.249512 -0.024355  0.005157  0.031966  0.170467
##
## Coefficients:
##             Estimate Std. Error t value            Pr(>|t|)
```

```
## (Intercept) -0.35628    0.01139  -31.27 <0.0000000000000002 ***
## wOBA_marcel  2.13709    0.03684   58.01 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.05018 on 6762 degrees of freedom
## Multiple R-squared:  0.3323, Adjusted R-squared:  0.3322
## F-statistic:  3365 on 1 and 6762 DF,  p-value: < 0.00000000000000022
```

```
# Component models: Each component predicted by Marcel projection
singles_model <- lm(Singles_rate ~ wOBA_marcel, data = component_data)
doubles_model <- lm(Doubles_rate ~ wOBA_marcel, data = component_data)
triples_model <- lm(Triples_rate ~ wOBA_marcel, data = component_data)
hr_model <- lm(HR_rate ~ wOBA_marcel, data = component_data)
hbp_model <- lm(HBP_rate ~ wOBA_marcel, data = component_data)
sf_model <- lm(SF_rate ~ wOBA_marcel, data = component_data)
hard_model <- lm(Hard_pct ~ wOBA_marcel, data = component_data)
pull_model <- lm(Pull_pct ~ wOBA_marcel, data = component_data)
gb_model <- lm(GB_pct ~ wOBA_marcel, data = component_data)

# Calculate residuals: difference between actual and expected component values
component_data <- component_data |>
  mutate(
    Singles_rate_resid = residuals(singles_model),
    Doubles_rate_resid = residuals(doubles_model),
    Triples_rate_resid = residuals(triples_model),
    HR_rate_resid = residuals(hr_model),
    HBP_rate_resid = residuals(hbp_model),
    SF_rate_resid = residuals(sf_model),
    Hard_pct_resid = residuals(hard_model),
    Pull_pct_resid = residuals(pull_model),
    GB_pct_resid = residuals(gb_model)
  )
```

Bayesian Hierarchical Model

Priors

Priors based on domain knowledge and previous model fits

```
priors <- c(
  # Intercept: Below league average wOBA
  prior(normal(-0.86, 0.01), class = "Intercept"),

  # Marcel coefficient: Strong positive relationship with some uncertainty
  prior(normal(1.7, 0.37), class = "b", coef = "wOBA_marcel"),

  # Component residuals: Rough Priors. Have not dug into these. Seemed quite
  # prone to overfitting so tightened down
  prior(normal(0, 0.005), class = "b", coef = "Singles_rate_resid"),
  prior(normal(0, 0.005), class = "b", coef = "Doubles_rate_resid"),
  prior(normal(0, 0.005), class = "b", coef = "Triples_rate_resid"),
```

```r
  prior(normal(0, 0.005), class = "b", coef = "HR_rate_resid"),
  prior(normal(0, 0.005), class = "b", coef = "HBP_rate_resid"),
  prior(normal(0, 0.005), class = "b", coef = "SF_rate_resid"),

  # Age spline coefficients: Trying to allow for more realistic age curves
  prior(normal(0, 0.03), class = "b", coef = "nsAgedfEQ41"),
  prior(normal(0, 0.03), class = "b", coef = "nsAgedfEQ42"),
  prior(normal(0, 0.03), class = "b", coef = "nsAgedfEQ43"),
  prior(normal(0, 0.03), class = "b", coef = "nsAgedfEQ44"),

  # Player effects: Some players are outliers (heavy tails observed in QQ plot)
  prior(student_t(3, 0, 0.1), class = "sd", group = "IDfg"),

  # Season effects: Normal (some minor deviation at both extremes)
  prior(normal(0, 0.05), class = "sd", group = "Season"),

  # Team effects: Normal (QQ plot showed good normality)
  prior(normal(0, 0.05), class = "sd", group = "Team"),

  # Residual error: Overall uncertainty
  prior(normal(0, 0.05), class = "sigma")
)
```

Model Fitting

Formula: wOBA predicted by Marcel + component residuals + age + random effects

```r
# Depending on computer will have to change with this
# Also just the current version that I ran (included everything).
# Testing some things out.
model <- brm(
  wOBA ~ wOBA_marcel +
         Singles_rate_resid + Doubles_rate_resid + Triples_rate_resid +
         HR_rate_resid + HBP_rate_resid + SF_rate_resid +
         Hard_pct_resid + Pull_pct_resid + GB_pct_resid +
         ns(Age, df = 4) +
         (1 | IDfg) + (1 | Season) + (1 | Team),
  data = component_data,
  family = gaussian(),
  prior = priors,
  warmup = 1000,
  chains = 4, iter = 5000, seed = 0804,
  backend = "cmdstanr",
  cores = 6
)
```

```
## Start sampling
```

```
## Running MCMC with 4 chains, at most 6 in parallel...
##
## Chain 1 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 2 Iteration:    1 / 5000 [  0%]  (Warmup)
```

```
## Chain 3 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 4 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 4 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 3 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 1 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 2 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 4 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 3 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 1 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 2 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 4 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 3 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 1 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 2 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 4 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 3 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 1 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 2 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 4 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 3 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 1 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 4 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 2 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 3 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 4 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 1 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 2 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 3 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 4 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 1 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 2 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 3 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 4 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 1 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 3 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 4 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 4 Iteration: 1001 / 5000 [ 20%]  (Sampling)
## Chain 2 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 1 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 4 Iteration: 1100 / 5000 [ 22%]  (Sampling)
## Chain 3 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 3 Iteration: 1001 / 5000 [ 20%]  (Sampling)
## Chain 2 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 1 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 1 Iteration: 1001 / 5000 [ 20%]  (Sampling)
## Chain 4 Iteration: 1200 / 5000 [ 24%]  (Sampling)
## Chain 2 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 2 Iteration: 1001 / 5000 [ 20%]  (Sampling)
## Chain 1 Iteration: 1100 / 5000 [ 22%]  (Sampling)
## Chain 4 Iteration: 1300 / 5000 [ 26%]  (Sampling)
## Chain 3 Iteration: 1100 / 5000 [ 22%]  (Sampling)
## Chain 2 Iteration: 1100 / 5000 [ 22%]  (Sampling)
## Chain 1 Iteration: 1200 / 5000 [ 24%]  (Sampling)
## Chain 4 Iteration: 1400 / 5000 [ 28%]  (Sampling)
```

```
## Chain 2 Iteration: 1200 / 5000 [ 24%]  (Sampling)
## Chain 1 Iteration: 1300 / 5000 [ 26%]  (Sampling)
## Chain 4 Iteration: 1500 / 5000 [ 30%]  (Sampling)
## Chain 3 Iteration: 1200 / 5000 [ 24%]  (Sampling)
## Chain 2 Iteration: 1300 / 5000 [ 26%]  (Sampling)
## Chain 1 Iteration: 1400 / 5000 [ 28%]  (Sampling)
## Chain 4 Iteration: 1600 / 5000 [ 32%]  (Sampling)
## Chain 2 Iteration: 1400 / 5000 [ 28%]  (Sampling)
## Chain 1 Iteration: 1500 / 5000 [ 30%]  (Sampling)
## Chain 4 Iteration: 1700 / 5000 [ 34%]  (Sampling)
## Chain 3 Iteration: 1300 / 5000 [ 26%]  (Sampling)
## Chain 2 Iteration: 1500 / 5000 [ 30%]  (Sampling)
## Chain 1 Iteration: 1600 / 5000 [ 32%]  (Sampling)
## Chain 4 Iteration: 1800 / 5000 [ 36%]  (Sampling)
## Chain 2 Iteration: 1600 / 5000 [ 32%]  (Sampling)
## Chain 1 Iteration: 1700 / 5000 [ 34%]  (Sampling)
## Chain 4 Iteration: 1900 / 5000 [ 38%]  (Sampling)
## Chain 3 Iteration: 1400 / 5000 [ 28%]  (Sampling)
## Chain 2 Iteration: 1700 / 5000 [ 34%]  (Sampling)
## Chain 1 Iteration: 1800 / 5000 [ 36%]  (Sampling)
## Chain 4 Iteration: 2000 / 5000 [ 40%]  (Sampling)
## Chain 2 Iteration: 1800 / 5000 [ 36%]  (Sampling)
## Chain 1 Iteration: 1900 / 5000 [ 38%]  (Sampling)
## Chain 4 Iteration: 2100 / 5000 [ 42%]  (Sampling)
## Chain 3 Iteration: 1500 / 5000 [ 30%]  (Sampling)
## Chain 2 Iteration: 1900 / 5000 [ 38%]  (Sampling)
## Chain 1 Iteration: 2000 / 5000 [ 40%]  (Sampling)
## Chain 4 Iteration: 2200 / 5000 [ 44%]  (Sampling)
## Chain 2 Iteration: 2000 / 5000 [ 40%]  (Sampling)
## Chain 1 Iteration: 2100 / 5000 [ 42%]  (Sampling)
## Chain 4 Iteration: 2300 / 5000 [ 46%]  (Sampling)
## Chain 3 Iteration: 1600 / 5000 [ 32%]  (Sampling)
## Chain 2 Iteration: 2100 / 5000 [ 42%]  (Sampling)
## Chain 1 Iteration: 2200 / 5000 [ 44%]  (Sampling)
## Chain 4 Iteration: 2400 / 5000 [ 48%]  (Sampling)
## Chain 2 Iteration: 2200 / 5000 [ 44%]  (Sampling)
## Chain 1 Iteration: 2300 / 5000 [ 46%]  (Sampling)
## Chain 4 Iteration: 2500 / 5000 [ 50%]  (Sampling)
## Chain 3 Iteration: 1700 / 5000 [ 34%]  (Sampling)
## Chain 2 Iteration: 2300 / 5000 [ 46%]  (Sampling)
## Chain 1 Iteration: 2400 / 5000 [ 48%]  (Sampling)
## Chain 4 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 2 Iteration: 2400 / 5000 [ 48%]  (Sampling)
## Chain 1 Iteration: 2500 / 5000 [ 50%]  (Sampling)
## Chain 4 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 3 Iteration: 1800 / 5000 [ 36%]  (Sampling)
## Chain 2 Iteration: 2500 / 5000 [ 50%]  (Sampling)
## Chain 1 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 4 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 2 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 1 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 4 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 3 Iteration: 1900 / 5000 [ 38%]  (Sampling)
## Chain 2 Iteration: 2700 / 5000 [ 54%]  (Sampling)
```

```
## Chain 1 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 4 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 2 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 1 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 4 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 3 Iteration: 2000 / 5000 [ 40%]  (Sampling)
## Chain 2 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 1 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 4 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 2 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 4 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 1 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 3 Iteration: 2100 / 5000 [ 42%]  (Sampling)
## Chain 2 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 4 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 1 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 2 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 4 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 1 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 3 Iteration: 2200 / 5000 [ 44%]  (Sampling)
## Chain 2 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 4 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 1 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 2 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 4 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 1 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 3 Iteration: 2300 / 5000 [ 46%]  (Sampling)
## Chain 2 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 4 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 1 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 2 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 4 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 1 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 3 Iteration: 2400 / 5000 [ 48%]  (Sampling)
## Chain 2 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 4 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 1 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 2 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 4 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 1 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 3 Iteration: 2500 / 5000 [ 50%]  (Sampling)
## Chain 2 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 4 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 1 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 2 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 4 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 1 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 3 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 2 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 4 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 1 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 2 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 4 Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 1 Iteration: 4300 / 5000 [ 86%]  (Sampling)
```

```
## Chain 3 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 2 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 4 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 1 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 2 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 4 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 1 Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 3 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 2 Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 4 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 1 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 2 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 4 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 1 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 3 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 2 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 4 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 4 finished in 738.7 seconds.
## Chain 1 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 2 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 1 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 3 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 2 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 1 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 1 finished in 773.4 seconds.
## Chain 2 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 2 finished in 778.3 seconds.
## Chain 3 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 3 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 3 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 3 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 3 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 3 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 3 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 3 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 3 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 3 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 3 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 3 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 3 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 3 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 3 Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 3 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 3 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 3 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 3 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 3 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 3 finished in 1243.4 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 883.5 seconds.
## Total execution time: 1243.6 seconds.
```

```
## Warning: 4000 of 16000 (25.0%) transitions hit the maximum treedepth limit of 10.
```

```
## See https://mc-stan.org/misc/warnings for details.
```

Save the model

```r
# Save the fitted model for future use
saveRDS(model, file = "baseball_woba_model_2024.rds")
```

Model Diagnostics

```r
# Model summary and diagnostics
print(summary(model))
```

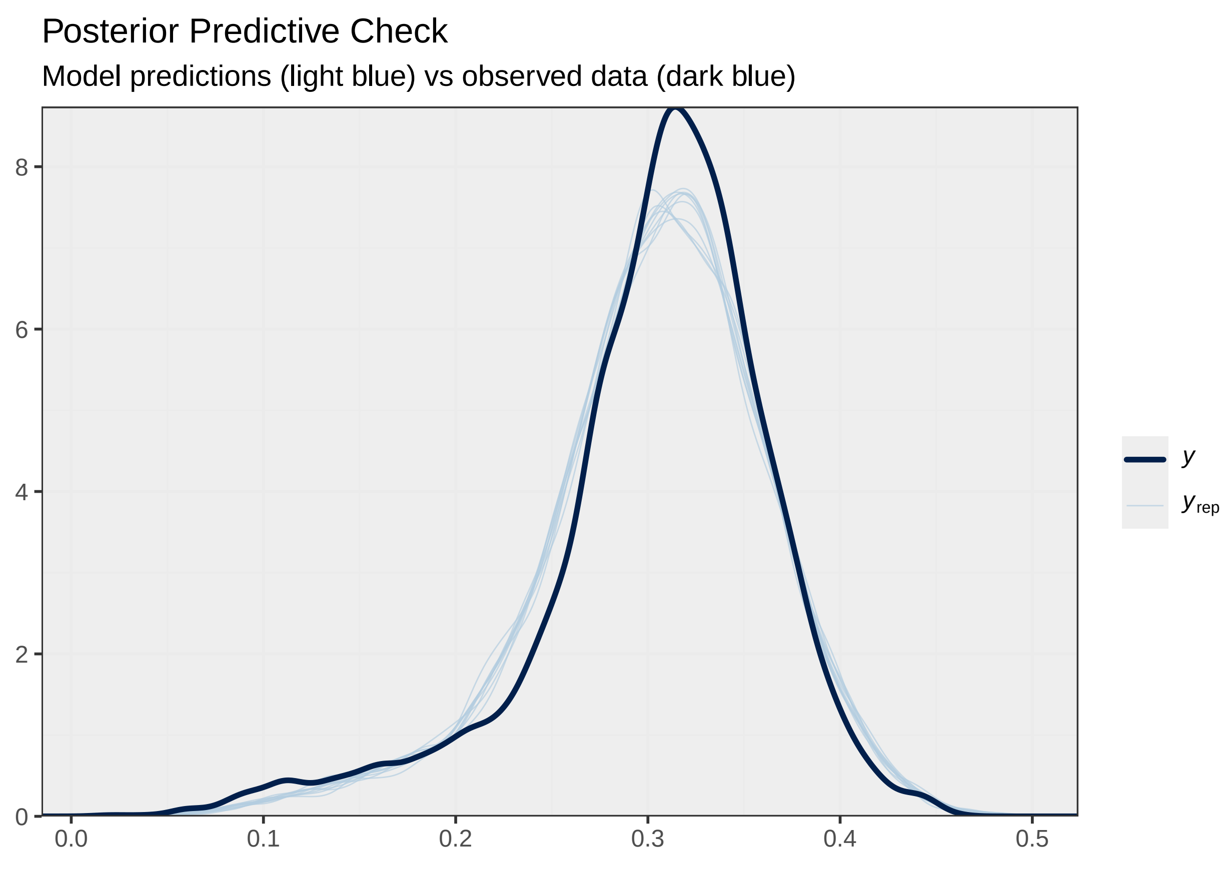
```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: wOBA ~ wOBA_marcel + Singles_rate_resid + Doubles_rate_resid + Triples_rate_resid + HR_rate_
##    Data: component_data (Number of observations: 6764)
##   Draws: 4 chains, each with iter = 5000; warmup = 1000; thin = 1;
##          total post-warmup draws = 16000
##
## Multilevel Hyperparameters:
## ~IDfg (Number of levels: 1536)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     0.02      0.00     0.02     0.02 1.00     3053     7804
##
## ~Season (Number of levels: 15)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     0.45      0.02     0.41     0.50 1.00     1175     2207
##
## ~Team (Number of levels: 33)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     0.00      0.00     0.00     0.01 1.00     4950     8620
##
## Regression Coefficients:
##                    Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept             -1.37      0.02    -1.40    -1.33 1.00     6691     9986
## wOBA_marcel            1.69      0.05     1.60     1.78 1.00     4226     8530
## Singles_rate_resid     0.09      0.00     0.08     0.10 1.00    26764    12238
## Doubles_rate_resid     0.03      0.00     0.02     0.04 1.00    32295    11838
## Triples_rate_resid     0.00      0.00    -0.00     0.01 1.00    36650    11327
## HR_rate_resid          0.02      0.01     0.02     0.03 1.00    31588    11625
## HBP_rate_resid         0.01      0.01    -0.00     0.02 1.00    35059    11308
## SF_rate_resid         -0.00      0.01    -0.01     0.01 1.00    33662    11622
## Hard_pct_resid         0.41      0.01     0.39     0.42 1.00    12523    12652
## Pull_pct_resid         0.03      0.01     0.02     0.05 1.00    12167    11507
## GB_pct_resid          -0.11      0.01    -0.12    -0.09 1.00    11851    12795
## nsAgedfEQ41           -0.01      0.00    -0.01     0.00 1.00     9606    11747
## nsAgedfEQ42           -0.02      0.00    -0.03    -0.01 1.00    11066    11044
## nsAgedfEQ43           -0.03      0.01    -0.05    -0.01 1.00    10204    11087
## nsAgedfEQ44           -0.03      0.01    -0.05    -0.01 1.00    10816    12101
##
```

```
## Further Distributional Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     0.03      0.00     0.03     0.03 1.00     7814    10758
##
## Draws were sampled using sample(hmc). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

```
# Posterior predictive check: Compare model predictions to actual data
pp_check_plot <- pp_check(model) +
  theme_bw() +
  theme(panel.background = element_rect(fill = '#eeeeee')) +
  labs(title = "Posterior Predictive Check",
       subtitle = "Model predictions (light blue) vs observed data (dark blue)")
```

```
## Using 10 posterior draws for ppc type 'dens_overlay' by default.
```

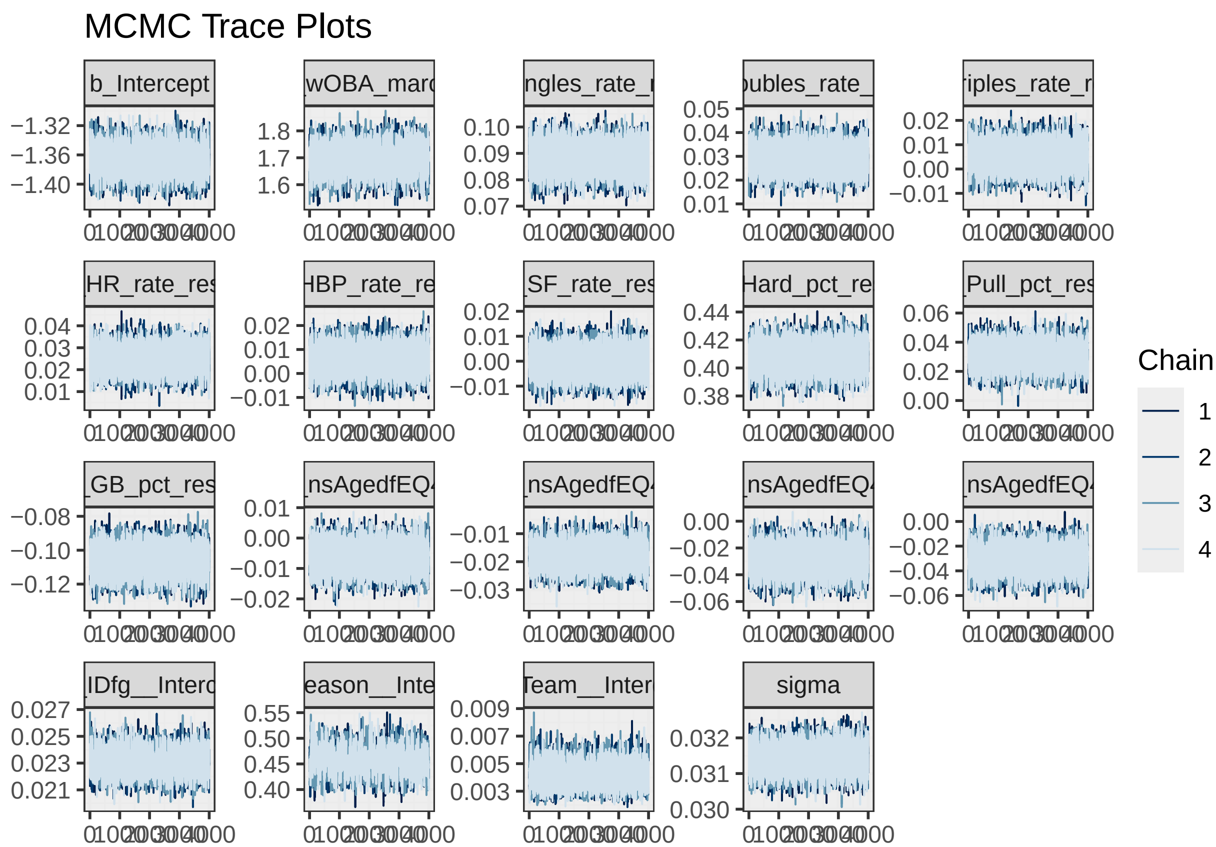
```
print(pp_check_plot)
```




```
# MCMC trace plots: Check for convergence
chains_plot <- mcmc_plot(model, type = "trace") +
  theme_bw() +
  theme(panel.background = element_rect(fill = '#eeeeee')) +
  labs(title = "MCMC Trace Plots")
```

```
## No divergences to plot.
```

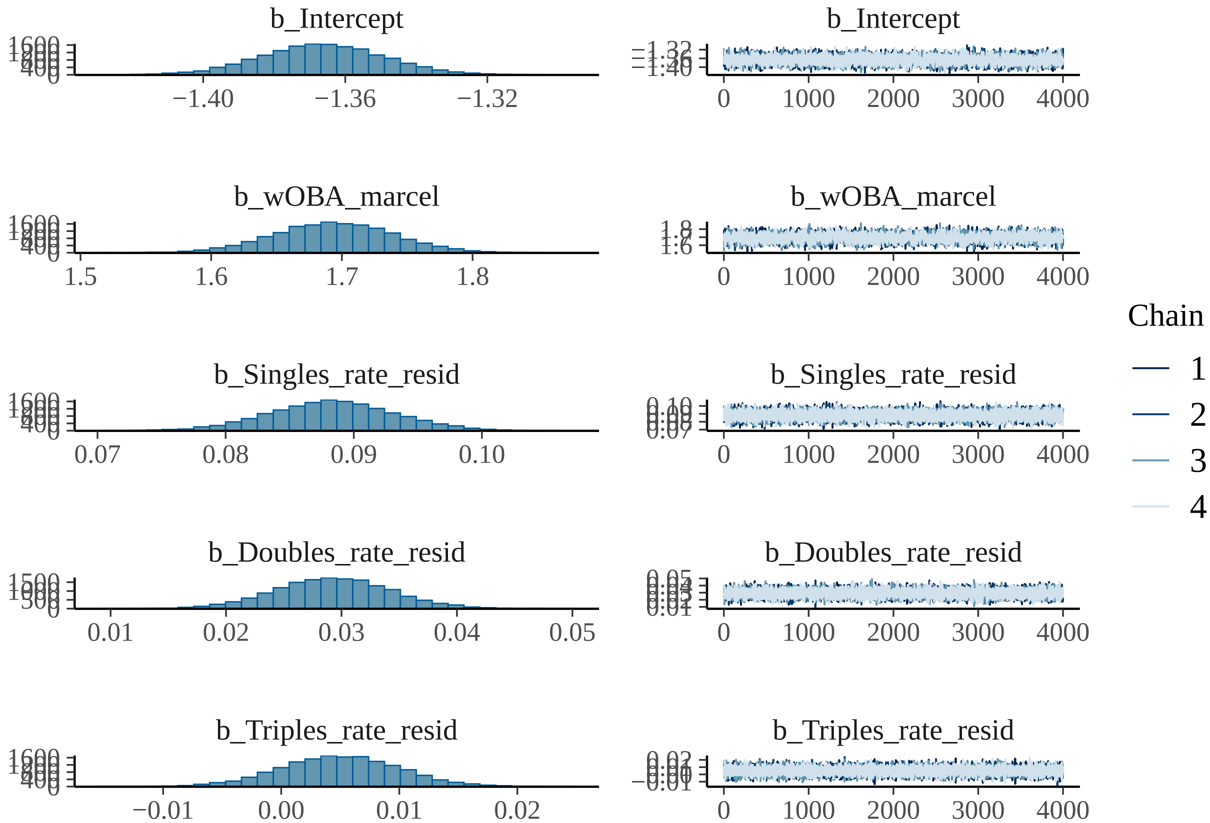
```r
print(chains_plot)
```

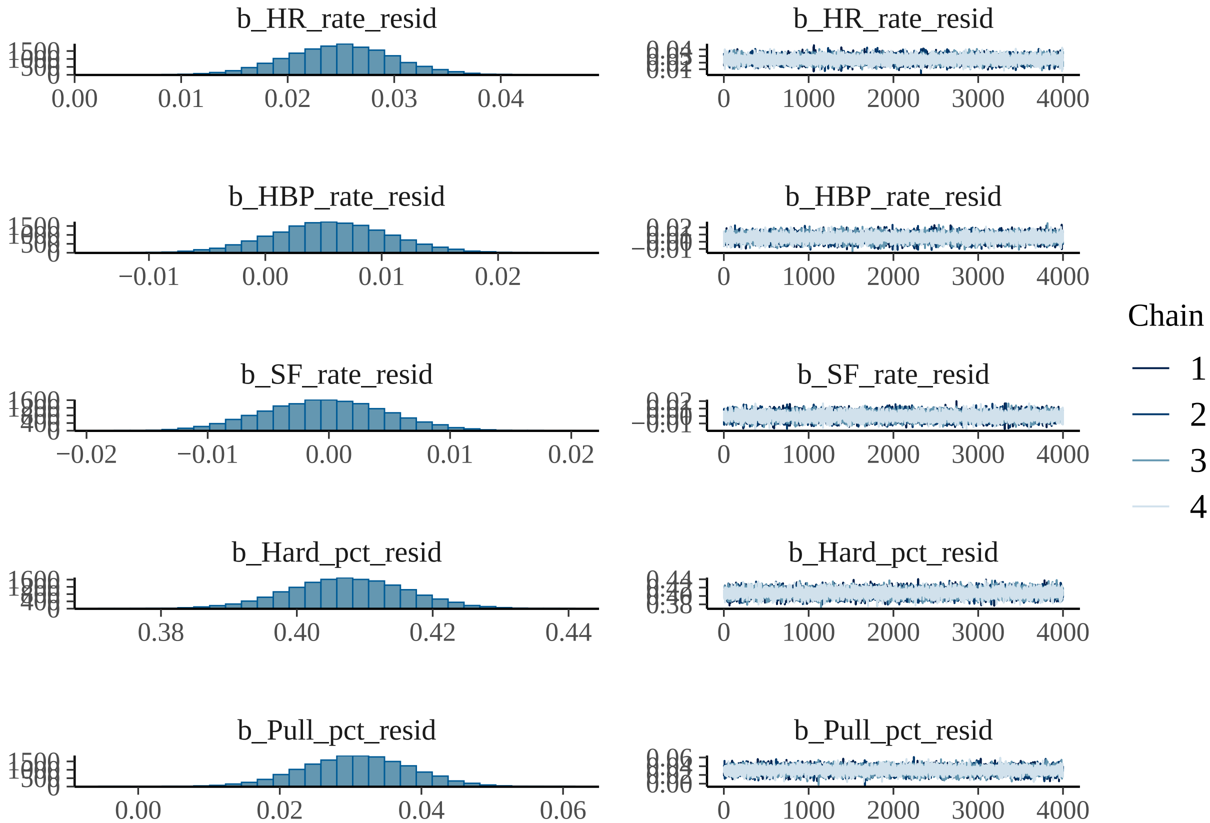


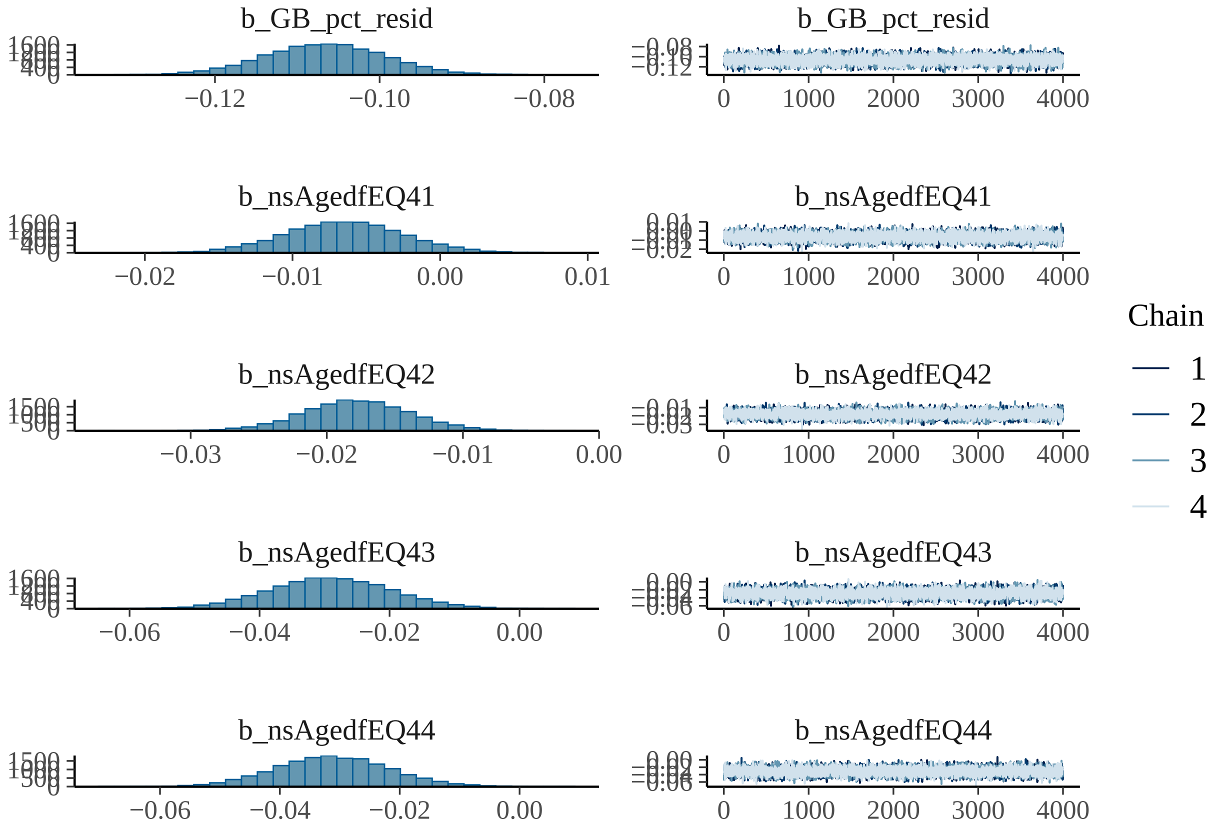

```r
# R-hat convergence diagnostic
rhat_values <- rhat(model)
print(summary(rhat_values))
```

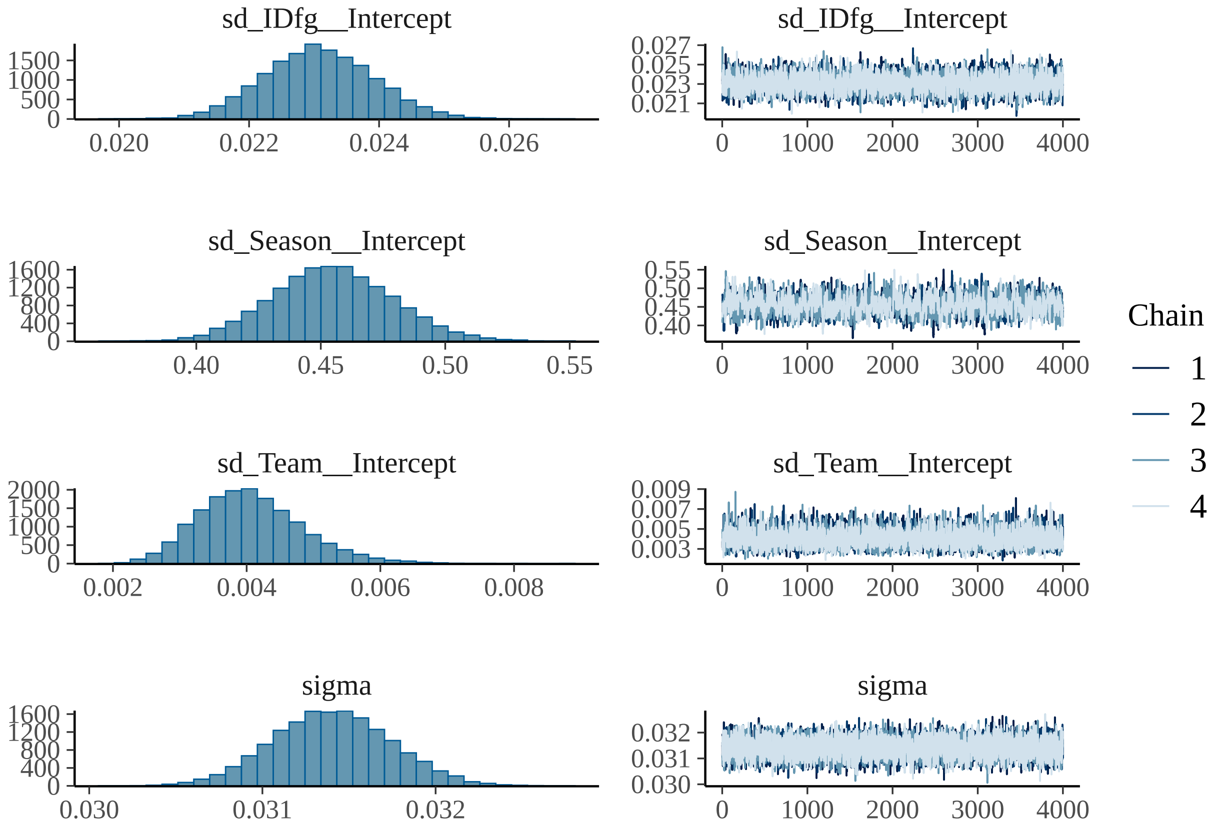
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.9998  1.0001  1.0002  1.0003  1.0004  1.0026
```

```r
# Alternative model summary visualization
plot(model, type = "mermaid")
```


b_Intercept
−1.40
−1.36
−1.32
b_Intercept
0
1000
2000
3000
4000
b_wOBA_marcel
1.5
1.6
1.7
1.8
b_wOBA_marcel
b_Singles_rate_resid
0.07
0.08
0.09
0.10
b_Singles_rate_resid
b_Doubles_rate_resid
0.01
0.02
0.03
0.04
0.05
b_Doubles_rate_resid
b_Triples_rate_resid
−0.01
0.00
0.01
0.02
b_Triples_rate_resid
Chain
1
2
3
4

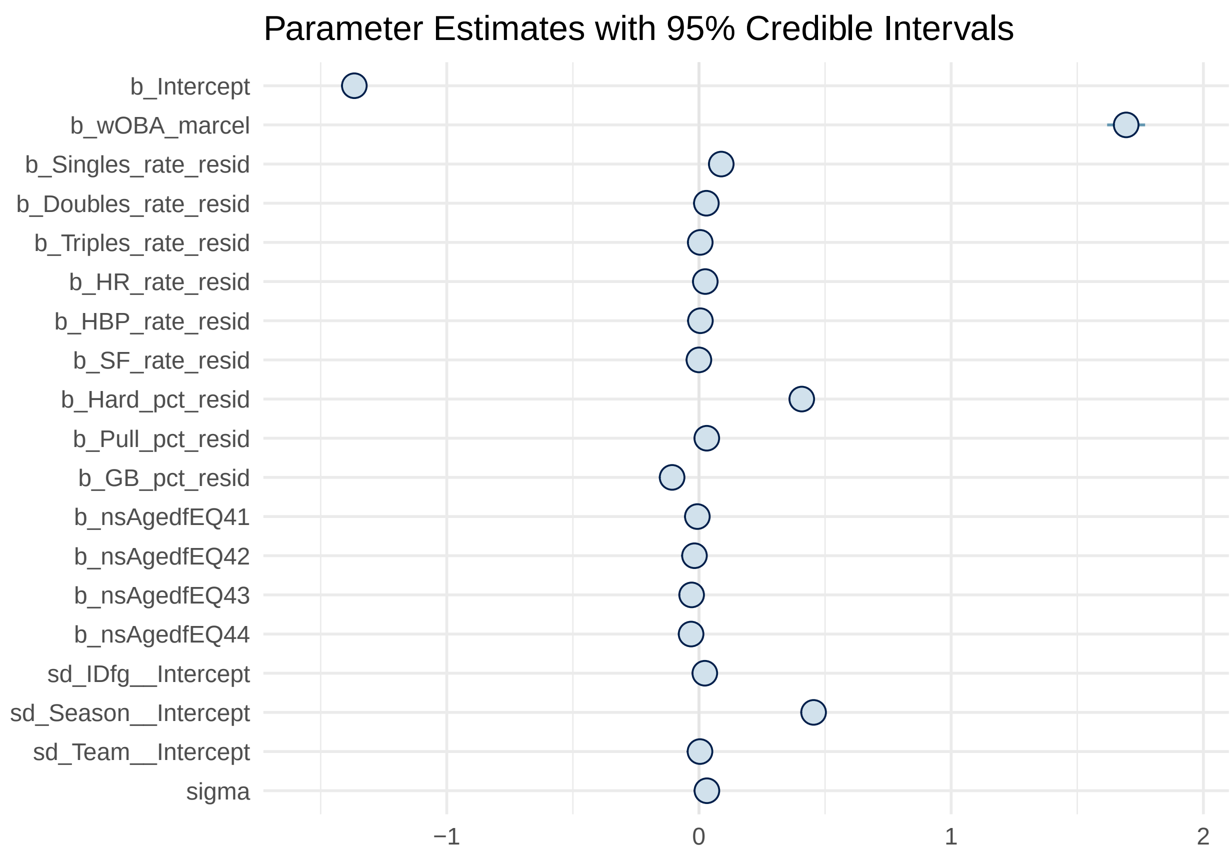

b_HR_rate_resid
b_HBP_rate_resid
b_SF_rate_resid
b_Hard_pct_resid
b_Pull_pct_resid
Chain
1
2
3
4


b_GB_pct_resid
−0.12
−0.10
−0.08
b_GB_pct_resid
0
1000
2000
3000
4000
b_nsAgedfEQ41
−0.02
−0.01
0.00
0.01
b_nsAgedfEQ41
b_nsAgedfEQ42
−0.03
−0.02
−0.01
0.00
b_nsAgedfEQ42
b_nsAgedfEQ43
−0.06
−0.04
−0.02
0.00
b_nsAgedfEQ43
b_nsAgedfEQ44
−0.06
−0.04
−0.02
0.00
b_nsAgedfEQ44
Chain
1
2
3
4



Parameter Estimates

```r
# Parameter estimates with credible intervals
mcmc_plot(model, type = "intervals") +
  theme_minimal() +
  labs(title = "Parameter Estimates with 95% Credible Intervals")
```



Model Projections for 2024

Generate projections for 2024 season using the fitted model.

Projections Data Preparation

```r
# Calculate 2023 league average for regression to mean
lg_wOBA_2023 <- current_data |>
  filter(Season == 2023) |>
  group_by(Season) |>
  summarize(lg_wOBA = mean(wOBA, na.rm = TRUE)) |>
  pull(lg_wOBA)

# Create Marcel projections for 2024 using data through 2023
projection_data <- current_data |>
  filter(Season < 2024) |>
  arrange(IDfg, Season) |>
  group_by(IDfg) |>
  filter(any(Season == 2023)) |>  # Only players active in 2023
  summarize(
    # Player information for 2024
    Name = last(Name),
```

```
    Team = last(Team),
    Age = last(Age) + 1,  # Age up by one year
    Season = 2024,

    # Historical performance data
    wOBA_prev1 = last(wOBA),
    PA_prev1 = last(PA),

    wOBA_prev2 = if(n() >= 2) nth(wOBA, n()-1) else NA_real_,
    PA_prev2 = if(n() >= 2) nth(PA, n()-1) else NA_real_,

    wOBA_prev3 = if(n() >= 3) nth(wOBA, n()-2) else NA_real_,
    PA_prev3 = if(n() >= 3) nth(PA, n()-2) else NA_real_,

    # Count available years
    years_of_data = sum(!is.na(c(last(wOBA),
                                 if(n() >= 2) nth(wOBA, n()-1) else NA,
                                 if(n() >= 3) nth(wOBA, n()-2) else NA)))
  ) |>
  mutate(
    # Apply Marcel projection
    wOBA_marcel_3yr = case_when(
      years_of_data >= 3 ~ (5*wOBA_prev1*PA_prev1 + 4*wOBA_prev2*PA_prev2 + 3*wOBA_prev3*PA_prev3) /
                        (5*PA_prev1 + 4*PA_prev2 + 3*PA_prev3),
      TRUE ~ NA_real_
    ),

    wOBA_marcel_2yr = case_when(
      years_of_data >= 2 ~ (5*wOBA_prev1*PA_prev1 + 4*wOBA_prev2*PA_prev2) /
                        (5*PA_prev1 + 4*PA_prev2),
      TRUE ~ NA_real_
    ),

    wOBA_marcel_1yr = wOBA_prev1,

    wOBA_marcel_temp = case_when(
      years_of_data == 3 ~ wOBA_marcel_3yr,
      years_of_data == 2 ~ wOBA_marcel_2yr,
      years_of_data == 1 ~ wOBA_marcel_1yr,
      TRUE ~ NA_real_
    ),

    pa_regression = case_when(
      years_of_data == 3 ~ PA_prev1 + PA_prev2 + PA_prev3,
      years_of_data == 2 ~ PA_prev1 + PA_prev2,
      years_of_data == 1 ~ PA_prev1,
      TRUE ~ 0
    ),

    reg_weight = pa_regression / (pa_regression + 1500),

    # Final Marcel projection for 2024
    wOBA_marcel = reg_weight * wOBA_marcel_temp +
```

```
      (1 - reg_weight) * lg_wOBA_2023
  ) |>
  filter(!is.na(wOBA_marcel))
```

Component Statistics for Projections

```
# Add 2023 component statistics for projection
projection_component_data <- projection_data |>
  left_join(
    data |>
      filter(Season == 2023) |>
      select(IDfg, AB, PA, `1B`, `2B`, `3B`, HR, HBP, SF,
             `Hard%`, `Pull%`, `GB%`),
    by = "IDfg"
  ) |>
  mutate(
    # Calculate 2023 component rates
    Singles_rate = `1B` / AB,
    Doubles_rate = `2B` / AB,
    Triples_rate = `3B` / AB,
    HR_rate = HR / AB,
    HBP_rate = HBP / PA,
    SF_rate = SF / PA,
    Hard_pct = `Hard%`,
    Pull_pct = `Pull%`,
    GB_pct = `GB%`
  )

# Calculate component residuals using historical models
projection_component_data <- projection_component_data |>
  mutate(
    Singles_rate_resid = Singles_rate - predict(
      singles_model, newdata = projection_component_data
      ),
    Doubles_rate_resid = Doubles_rate - predict(
      doubles_model, newdata = projection_component_data
      ),
    Triples_rate_resid = Triples_rate - predict(
      triples_model, newdata = projection_component_data
      ),
    HR_rate_resid = HR_rate - predict(
      hr_model, newdata = projection_component_data
      ),
    HBP_rate_resid = HBP_rate - predict(
      hbp_model, newdata = projection_component_data
      ),
    SF_rate_resid = SF_rate - predict(
      sf_model, newdata = projection_component_data
      ),
    Hard_pct_resid = Hard_pct - predict(
      hard_model, newdata = projection_component_data
```

```
    ),
    Pull_pct_resid = Pull_pct - predict(
      pull_model, newdata = projection_component_data
      ),
    GB_pct_resid = GB_pct - predict(
      gb_model, newdata = projection_component_data
      )
  )
```

Generate Posterior Predictions

```
# Generate posterior predictions
set.seed(0804)
posterior_projection <- posterior_predict(model,
                                          newdata = projection_component_data,
                                          allow_new_levels = TRUE)

# Summarize posterior predictions
projection_component_data <- projection_component_data |>
  mutate(
    # Point estimate: posterior mean
    wOBA_proj = colMeans(posterior_projection),

    # Uncertainty intervals: 80% credible interval for now
    wOBA_proj_lower = apply(posterior_projection, 2, quantile, probs = 0.10),
    wOBA_proj_upper = apply(posterior_projection, 2, quantile, probs = 0.90)
  ) |>
  select(IDfg, Season, Name, Team, Age, wOBA_proj, wOBA_proj_lower,
         wOBA_proj_upper, wOBA_marcel)
```

Model Validation

Compare the 2024 projections to actual performance to evaluate models accuracy.

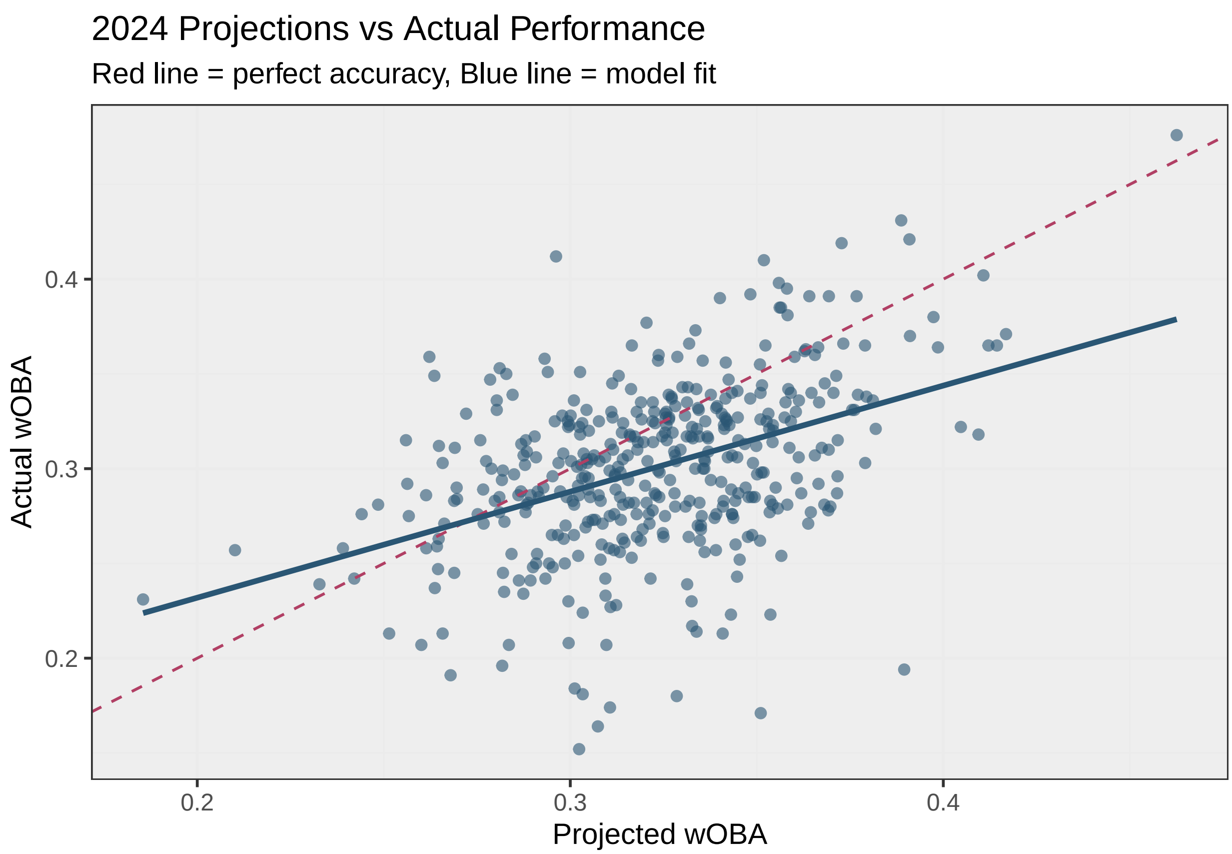
Actual vs Projected Performance

```
# Load actual 2024 performance
wOBA_2024 <- data |>
  filter(Season == 2024) |>
  select(IDfg, wOBA)

# Join projections with actual performance
validation_data <- left_join(
  projection_component_data, wOBA_2024, by = 'IDfg') |>
  rename(wOBA_actual_2024 = wOBA) |>
  drop_na()  # Keep only players with both projections and actual data
```

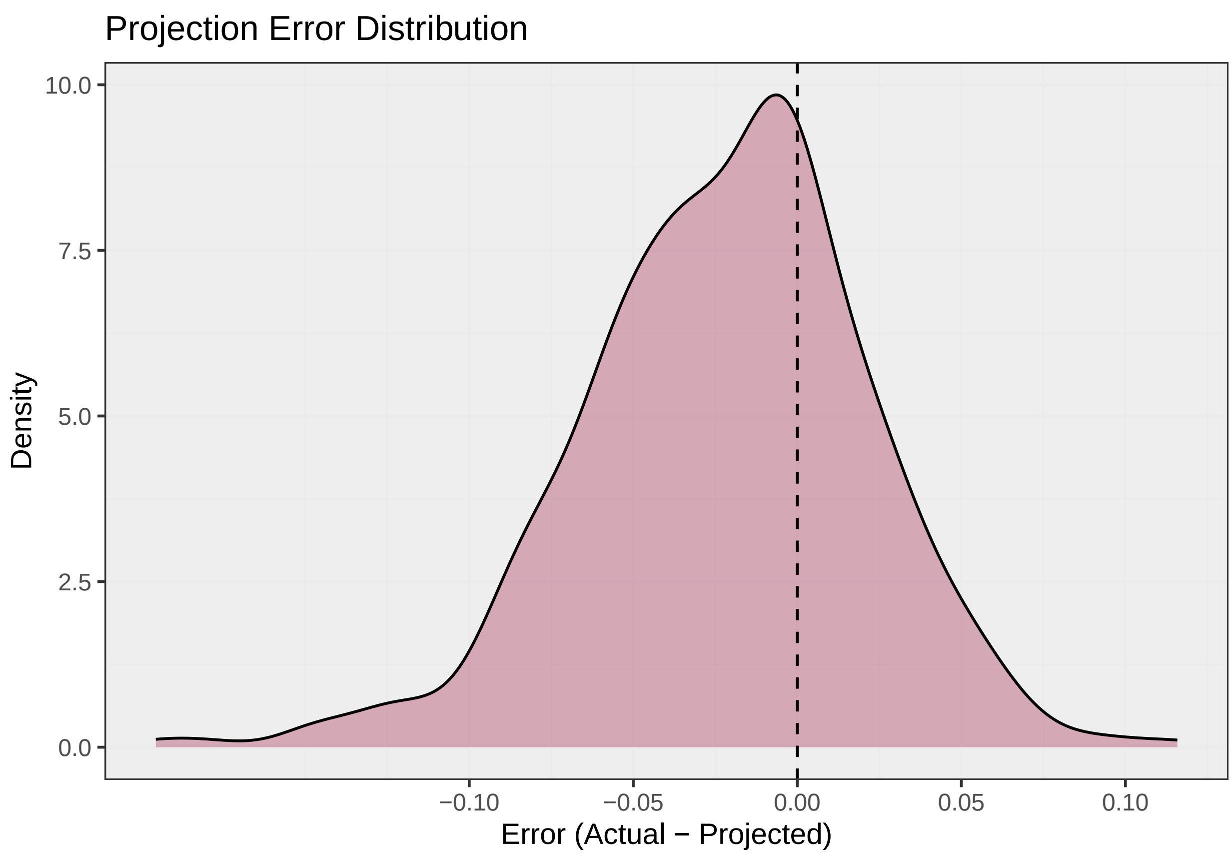
Validation Plots

```r
# Scatter plot: Projected vs Actual wOBA
proj_vs_actual <- ggplot(
  validation_data, aes(x = wOBA_proj, y = wOBA_actual_2024)) +
  geom_point(alpha = 0.6, color = "#2a5674") +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed", color= "#b13f64") +
  geom_smooth(method = "lm", se = FALSE, color = "#2a5674") +
  theme_bw() +
  theme(panel.background = element_rect(fill = '#eeeeee')) +
  labs(title = "2024 Projections vs Actual Performance",
       x = "Projected wOBA",
       y = "Actual wOBA",
       subtitle = "Red line = perfect accuracy, Blue line = model fit")

# Error distribution
error_dist <- validation_data |>
  mutate(error = wOBA_actual_2024 - wOBA_proj) |>
  ggplot(aes(x = error)) +
  geom_density(fill = "#b13f64", alpha = 0.4) +
  geom_vline(xintercept = 0, linetype = "dashed", color = "black") +
  scale_x_continuous(breaks = seq(-0.10, 0.10, 0.05)) +
  theme_bw() +
  theme(panel.background = element_rect(fill = '#eeeeee')) +
  labs(title = "Projection Error Distribution",
       x = "Error (Actual - Projected)",
       y = "Density")

print(proj_vs_actual)
```

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




```r
print(error_dist)
```



Model Performance Metrics

```r
# Calculate model performance
validation_summary <- validation_data |>
  summarise(
    n_players = n(),
    mean_abs_error = round(
      mean(
        abs(
          wOBA_actual_2024 - wOBA_proj), na.rm = TRUE), 4),
    root_mean_sq_error = round(
      sqrt(
        mean(
          (wOBA_actual_2024 - wOBA_proj)^2, na.rm = TRUE)), 4),
    correlation = round(
      cor(
        wOBA_proj, wOBA_actual_2024, use = "complete.obs"), 3),
    r_squared = round(
      cor(
        wOBA_proj, wOBA_actual_2024, use = "complete.obs")^2, 3)
  )

print("Model Performance Summary:")
```

```
## [1] "Model Performance Summary:"
```

```r
print(validation_summary)
```

```
## # A tibble: 1 x 5
##   n_players mean_abs_error root_mean_sq_error correlation r_squared
##       <int>          <dbl>              <dbl>       <dbl>     <dbl>
## 1       419         0.0372             0.0487       0.429     0.184
```

```r
# Just curious what this says
print("Bayesian R-squared:")
```

```
## [1] "Bayesian R-squared:"
```

```r
print(bayes_R2(model))
```

```
##     Estimate   Est.Error      Q2.5     Q97.5
## R2 0.7312343 0.004479943 0.7222173 0.7398089
```

Team Level Analysis

Team Projections Accuracy

```r
# Update team affiliations to 2024 (players may have changed teams)
team_data_2024 <- data |>
  filter(Season == 2024) |>
  select(IDfg, Team) |>
  distinct()

final_projections <- validation_data |>
  select(-Team) |>
  left_join(team_data_2024, by = "IDfg") |>
  mutate(error = abs(wOBA_actual_2024 - wOBA_proj))

# Team projection accuracy ranking
team_accuracy <- final_projections |>
  na.omit() |>
  group_by(Team) |>
  summarise(
    n_players = n(),
    mean_error = round(mean(error), 3),
    .groups = "drop"
  ) |>
  arrange(mean_error)

print("Teams with Most Accurate Projections:")
```

```
## [1] "Teams with Most Accurate Projections:"
```

```r
print(head(team_accuracy, 10))
```

```
## # A tibble: 10 x 3
##    Team  n_players mean_error
##    <chr>     <int>      <dbl>
##  1 PHI          12      0.015
##  2 BAL          11      0.018
##  3 SDP          12      0.019
##  4 LAD          12      0.024
##  5 TBR          10      0.024
##  6 ARI          15      0.026
##  7 CLE          12      0.028
##  8 MIN          15      0.031
##  9 SFG          12      0.033
## 10 MIL          11      0.034
```

```r
print("Teams with Least Accurate Projections:")
```

```
## [1] "Teams with Least Accurate Projections:"
```

```r
print(tail(team_accuracy, 10))
```

```
## # A tibble: 10 x 3
##    Team  n_players mean_error
##    <chr>     <int>      <dbl>
##  1 CIN          15      0.044
##  2 WSN          10      0.044
##  3 CHC          12      0.045
##  4 CHW          11      0.046
##  5 OAK          12      0.046
##  6 ATL          11      0.049
##  7 COL          14      0.05
##  8 MIA          11      0.05
##  9 TOR           8      0.05
## 10 TEX          13      0.057
```

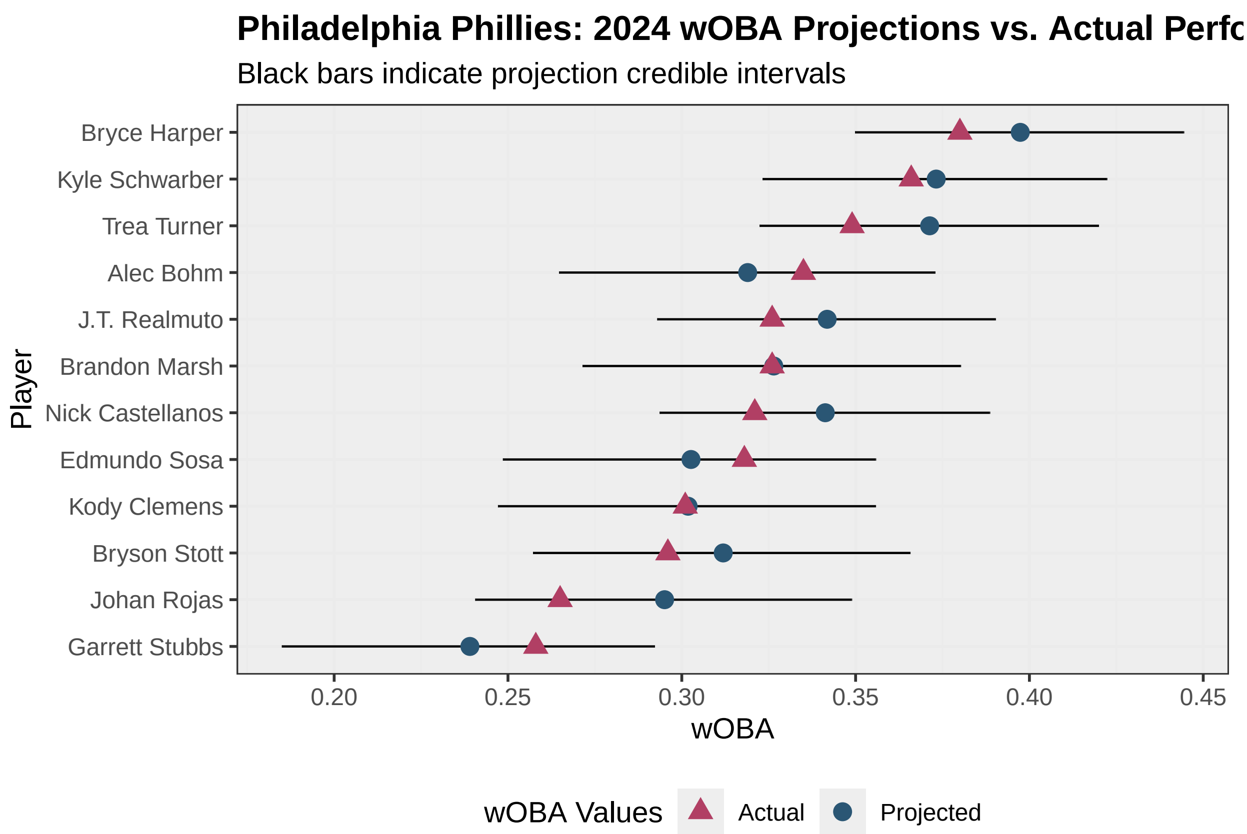
Individual Team Visualizations

```r
# Example: Philadelphia Phillies projection accuracy
phillies_plot <- final_projections |>
  filter(Team == "PHI") |>
  ggplot(aes(x = reorder(Name, wOBA_actual_2024), y = wOBA_proj)) +
  # Prediction intervals
  geom_errorbar(aes(ymin = wOBA_proj_lower, ymax = wOBA_proj_upper),
                width = 0, color = "black", size = 0.4) +
  # Projected values
  geom_point(aes(y = wOBA_proj, color = "Projected"),
             size = 3, shape = 16) +
  # Actual values
```

```r
  geom_point(aes(y = wOBA_actual_2024, color = "Actual"),
             size = 3, shape = 17) +
  scale_color_manual(name = "wOBA Values",
                     values = c("Projected" = "#2a5674",
                                "Actual" = "#b13f64")) +
  guides(color = guide_legend(override.aes = list(
    shape = c(17, 16),
    size = c(3, 3)
  ))) +
  coord_flip() +
  theme_bw() +
  theme(
    panel.background = element_rect(fill = '#eeeeee'),
    legend.position = "bottom",
    plot.title = element_text(face = "bold")
  ) +
  labs(
    title =
      "Philadelphia Phillies: 2024 wOBA Projections vs. Actual Performance",
    subtitle = "Black bars indicate projection credible intervals",
    x = "Player",
    y = "wOBA"
  )
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```r
print(phillies_plot)
```



Summary Tables

Best and Worst Projections

```r
# Most accurate individual projections
best_projections <- final_projections |>
  na.omit() |>
  arrange(error) |>
  head(10) |>
  select(Name, Team, wOBA_proj, wOBA_actual_2024, error) |>
  gt() |>
  tab_header(
    title = "Most Accurate Player Projections",
    subtitle =
      "Players with smallest absolute error between projected and actual 2024 wOBA"
  ) |>
  fmt_number(
    columns = c(wOBA_proj, wOBA_actual_2024, error),
    decimals = 3
  ) |>
  cols_label(
    Name = "Player",
    Team = "Team",
```

```r
    wOBA_proj = "Projected wOBA",
    wOBA_actual_2024 = "Actual wOBA",
    error = "Absolute Error"
  ) |>
  tab_style(
    style = cell_fill(color = "#e6f7e9"),
    locations = cells_body()
  )

# Least accurate individual projections
worst_projections <- final_projections |>
  na.omit() |>
  arrange(desc(error)) |>
  head(10) |>
  select(Name, Team, wOBA_proj, wOBA_actual_2024, error) |>
  gt() |>
  tab_header(
    title = "Least Accurate Player Projections",
    subtitle =
      "Players with largest absolute error between projected and actual 2024 wOBA"
  ) |>
  fmt_number(
    columns = c(wOBA_proj, wOBA_actual_2024, error),
    decimals = 3
  ) |>
  cols_label(
    Name = "Player",
    Team = "Team",
    wOBA_proj = "Projected wOBA",
    wOBA_actual_2024 = "Actual wOBA",
    error = "Absolute Error"
  ) |>
  tab_style(
    style = cell_fill(color = "#fbebeb"),
    locations = cells_body()
  )

#gt_two_column_layout(list(best_projections, worst_projections))

combined_data <- bind_rows(
  final_projections |>
    na.omit() |>
    arrange(error) |>
    head(5) |>
    mutate(category = "Most Accurate"),
  final_projections |>
    na.omit() |>
    arrange(desc(error)) |>
    head(5) |>
    mutate(category = "Least Accurate")
)

combined_projections <- combined_data |>
```

```r
  select(category, Name, Team, wOBA_proj, wOBA_actual_2024, error) |>
  gt(groupname_col = "category") |>
  tab_header(
    title = "Projection Accuracy Comparison",
    subtitle = "Most and least accurate 2024 wOBA projections"
  ) |>
  fmt_number(
    columns = c(wOBA_proj, wOBA_actual_2024, error),
    decimals = 3
  ) |>
  cols_label(
    Name = "Player",
    Team = "Team",
    wOBA_proj = "Projected wOBA",
    wOBA_actual_2024 = "Actual wOBA",
    error = "Absolute Error"
  ) |>
  tab_style(
    style = cell_fill(color = "#e6f7e9"),
    locations = cells_body(rows = category == "Most Accurate")
  ) |>
  tab_style(
    style = cell_fill(color = "#fbebeb"),
    locations = cells_body(rows = category == "Least Accurate")
  )

# print(combined_projections)
```