Badger Basketball March Madness Simulation

Table of contents


```
library(hoopR)
library(tidyverse)
library(BradleyTerry2)
oi_colors <-
  palette.colors(palette = "Okabe-Ito")
par(mar = c(3,3,2,1), mgp = c(1.8, 0.5, 0))
```

0.1 Motivation

The Wisconsin Badgers had a stellar 2024-2025 season. They made a run all the way to the BIG 10 championship game and ended up with a 3 seed in March Madness. However, the Badgers' tournament run came to an early end against BYU in just the second round. Every year, Badger fans seem disappointed regardless of how far the team makes it, unless they win it all. This provokes the question of how far are teams actually expected to make it in this brutal tournament. Using

a Bradley-Terry model to get team strengths, we will use these strengths to find how likely one team is to beat another. Using this, we will simulate 2025 March Madness 10,000 times to see if the Badgers really under performed, and find how many teams really had a shot at winning it all.

0.2 Data Preparation

Since we are simulating the 2025 March Madness we will need the data from the 2024-2025 NCAAMB season, we will be using hoop R to get this. Our data frame "games", will hold the winner, loser, and score of every D1 game played in the 2024-2025 season.

```
games <- load_mbb_schedule(season = 2025) %>%                    ①
  filter(!is.na(home_score) & !is.na(away_score)) %>%            ②
  transmute(
    Date = as.Date(game_date),
    Home = home_location,
    Opponent = away_location,
    Home_Score = home_score,                                     ③
    Away_Score = away_score,
    Home_Winner = as.numeric(home_winner),
    Opp_Winner = as.numeric(away_winner),
    Neutral = ifelse(neutral_site, 1, 0),
    Notes = notes_headline,
  )
head(games)
```

① Download all D1 games for the 2024-25 season
② Keep only games with recorded scores
③ Keep both home/away scores to later aggregate wins by pairing

```
-- ESPN MBB Schedule from hoopR data repository ----------------- hoopR 2.1.0 --

i Data updated: 2025-10-31 03:07:34 CDT

# A tibble: 6 x 9
  Date       Home  Opponent Home_Score Away_Score Home_Winner Opp_Winner Neutral
  <date>     <chr> <chr>         <int>      <int>       <dbl>      <dbl>   <dbl>
1 2025-04-07 Hous~ Florida          63         65           0          1       1
2 2025-04-05 Duke  Houston          67         70           0          1       1
3 2025-04-05 Aubu~ Florida          73         79           0          1       1
4 2025-04-05 Vill~ UCF              98        104           0          1       1
5 2025-04-05 Nebr~ Boise S~         79         69           1          0       1
6 2025-03-30 Aubu~ Michiga~         70         64           1          0       1
# i 1 more variable: Notes <chr>
```

Our next step is to reformat this data into a style that the BradleyTerry2 package will accept. Then to fit this data with our BradleyTerry2 package to get team strengths relative to one another.

```r
unik_teams <- sort(unique(c(games$Home, games$Opponent)))                    ①

results <- games %>%
  rename(home.team = Home, away.team = Opponent) %>%
  group_by(home.team, away.team) %>%                                         ②
  summarise(
    home.win = sum(Home_Winner),
    away.win = sum(Opp_Winner),
    .groups = "drop"
  ) %>%
  mutate(
    home.team = factor(home.team, levels = unik_teams),                      ③
    away.team = factor(away.team, levels = unik_teams)
  )

fit <- BradleyTerry2::BTm(                                                   ④
  outcome = cbind(home.win, away.win),
  player1 = home.team,
  player2 = away.team,
  refcat = "Montana",
  data = results
)
```

① Make sorted list of all team names
② Sum up wins across repeated matchups
③ Set teams as factors with the same levels
④ Build the BT model using total home/away wins

We can take a look at our results from fitting this model to find the strengths of a few teams relative to one another. As we can see, teams like Florida that performed very well over the regular season have a higher ability than a weaker team like Wofford.

```r
lambda_hat <- BradleyTerry2::BTabilities(fit)                                ①
lambda_hat[c("Wisconsin", "Florida", "Montana", "Wofford"),]
```

① Get each team's estimated strength

```
             ability      s.e.
Wisconsin  2.9976432 0.7436584
Florida    5.3226305 0.8641439
Montana    0.0000000 0.0000000
Wofford   -0.9991234 0.7440504
```

Our next step in preparing our data, is to get every single possible match up in college basketball. We will then use a function bt_prob, to get the probability of one team beating another for every since match up. This will make it very easy to look up when two teams are playing in march

madness, and finding the probability of one team winning. All of the probabilities in this data frame will be the probability of team1 beating team2.

```
bt_prob <- function(team1, team2) {                                    ①
  diff <- lambda_hat[team1, "ability"] - lambda_hat[team2, "ability"]
  1 / (1 + exp(-diff))                                                 ②
}

possible_matchups <- expand.grid(
  team1 = unik_teams,
  team2 = unik_teams,
  stringsAsFactors = FALSE
) |>
  filter(team1 != team2)

possible_matchups <- possible_matchups |>
  mutate(prob_team1_wins = mapply(bt_prob, team1, team2))              ③
```

① Function to convert the strength difference to a win chance
② Use the logistic formula to get a probability
③ Compute probabilities for all pairs at once

We can look through all of the possible matchups Wisconsin can have, and see the probability of them winning against every team in college basketball according to our model. Lets take a look at the Badger's chances against some teams they played in the regular season or march madness.

```
wisco_matchups <- possible_matchups |> filter(team1 == "Wisconsin") |>  ①
  mutate(prob_team1_wins = round(prob_team1_wins, 4))
wisco_matchups |> filter(team2 %in% c("Purdue", "Montana", "BYU", "Michigan"))  ②
```

① Show Wisconsin's chances vs everyone else
② A peek at a few familiar opponents

```
      team1    team2 prob_team1_wins
1 Wisconsin      BYU          0.5156
2 Wisconsin Michigan          0.4318
3 Wisconsin  Montana          0.9525
4 Wisconsin   Purdue          0.5133
```

0.2.1 Creating Covariates

Now that we each team's estimated ability from our Bradley-Terry model, we can add more information that might explain team performance. These include offensive rating, defensive rating, net ratings for home/away games, and strength of schedule. These covariates can help capture dimensions of team quality that wins and losses alone cannot measure.

0.2.1.1 Offensive and Defensive Rating

We calculate how many points each team scores and allows per game. The net rating is then computed as offensive rating minus defensive rating.

```
unik_teams <- sort(unique(c(games$Home, games$Opponent)))                    ①

offensive_ratings <- games %>%
  select(Home, Home_Score) %>%
  rename(team = Home, points_scored = Home_Score) %>%
  bind_rows(                                                                 ②
    games %>%
      select(Opponent, Away_Score) %>%
      rename(team = Opponent, points_scored = Away_Score)
  ) %>%
  group_by(team) %>%
  summarise(
    offensive_rating = mean(points_scored, na.rm = TRUE),                    ③
    games_played = n(),
    .groups = "drop"
  )

defensive_ratings <- games %>%
  select(Home, Away_Score) %>%
  rename(team = Home, points_allowed = Away_Score) %>%
  bind_rows(
    games %>%
      select(Opponent, Home_Score) %>%
      rename(team = Opponent, points_allowed = Home_Score)
  ) %>%
  group_by(team) %>%
  summarise(
    defensive_rating = mean(points_allowed, na.rm = TRUE),
    .groups = "drop"
  )

team_ratings <- offensive_ratings %>%
  left_join(defensive_ratings, by = "team") %>%
  mutate(
    net_rating = offensive_rating - defensive_rating                         ④
  )
```

① Make list of all unique teams
② Combine home and away scoring into one table
③ Calculate average points scored per game
④ Compute net rating

0.2.1.2 Strength of Schedule

We measure how strong a team's opponents were, on average, using each opponent's net rating. This captures how difficult a team's schedule was across both home and away games.

```
sos_home <- games %>%
  left_join(team_ratings %>% select(team, net_rating),
            by = c("Opponent" = "team")) %>%
  group_by(Home) %>%
  summarise(
    sos_home = mean(net_rating, na.rm = TRUE),
    .groups = "drop"
  ) %>%
  rename(team = Home)

sos_away <- games %>%
  left_join(team_ratings %>% select(team, net_rating),
            by = c("Home" = "team")) %>%
  group_by(Opponent) %>%
  summarise(
    sos_away = mean(net_rating, na.rm = TRUE),
    .groups = "drop"
  ) %>%
  rename(team = Opponent)

strength_of_schedule <- sos_home %>%
  full_join(sos_away, by = "team") %>%
  mutate(
    strength_of_schedule = (sos_home + sos_away) / 2                    ①
  )

team_ratings <- team_ratings %>%
  left_join(strength_of_schedule %>% select(team, strength_of_schedule),
            by = "team")
```

① Avg home and away opponent ratings to get one overall strength of schedule

0.2.1.3 Home vs. Away Performance

Next we analyze how teams perform differently at home versus on the road. We calculate average point differentials for home and away games, and measure home-court advantage.

```
home_performance <- games %>%
  group_by(Home) %>%
  summarise(
    home_offensive = mean(Home_Score, na.rm = TRUE),
    home_defensive = mean(Away_Score, na.rm = TRUE),
    home_net_rating = home_offensive - home_defensive,                  ①
    home_games = n(),
```

```
    .groups = "drop"
  ) %>%
  rename(team = Home)

away_performance <- games %>%
  group_by(Opponent) %>%
  summarise(
    away_offensive = mean(Away_Score, na.rm = TRUE),
    away_defensive = mean(Home_Score, na.rm = TRUE),
    away_net_rating = away_offensive - away_defensive,                    ②
    away_games = n(),
    .groups = "drop"
  ) %>%
  rename(team = Opponent)

home_away_splits <- home_performance %>%
  left_join(away_performance, by = "team") %>%
  mutate(
    home_advantage = home_net_rating - away_net_rating,                   ③
    travel_rating = away_net_rating
  )

team_ratings <- team_ratings %>%
  left_join(home_away_splits %>% select(team, home_net_rating, away_net_rating,
                                        home_advantage, travel_rating),
            by = "team")
```

① Compute home point differential (home net rating)
② Compute away point differential (away net rating)
③ Measure how much better teams perform at home vs. away

0.2.1.4 Weighting Covariates with Linear Regression

To understand how each covariate contributes to team ability, we fit a linear regression using baseline Bradley-Terry abilities as the dependent variable. Each coefficient represents how much a one-unit increase in that metric affects team ability.

```
lambda_baseline <- BTabilities(fit)                                       ①


team_abilities <- data.frame(
  team = rownames(lambda_baseline),
  baseline_ability = lambda_baseline[, "ability"]
) %>%
  left_join(team_ratings %>% select(team, net_rating, away_net_rating,
                                    strength_of_schedule),
            by = "team")
```

```r
head(team_abilities)

lm_fit <- lm(baseline_ability ~ net_rating + away_net_rating +
               strength_of_schedule,
             data = team_abilities)                                    ②
summary(lm_fit)
```

① get baseline team abilities from the BT model
② Regress ability on net rating, away rating, and schedule strength

```
               team baseline_ability net_rating away_net_rating
1 Abilene Christian       -1.3806983   1.322581       -5.400000
2         Air Force       -2.6427934 -12.000000      -17.416667
3             Akron        0.6202885   8.882353        0.812500
4           Alabama        4.2575952   9.405405        1.875000
5       Alabama A&M       -4.3221143  -5.612903      -18.214286
6     Alabama State       -1.7439520   1.235294       -5.277778
  strength_of_schedule
1           -2.6205335
2            2.6317126
3           -0.3233879
4            7.3063628
5           -4.8292700
6           -3.7439174

Call:
lm(formula = baseline_ability ~ net_rating + away_net_rating +
    strength_of_schedule, data = team_abilities)

Residuals:
     Min       1Q   Median       3Q      Max
-12.1953  -0.3772   0.0564   0.4932   2.0712

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)          -1.73818    0.14108 -12.320  < 2e-16 ***
net_rating            0.35943    0.01957  18.362  < 2e-16 ***
away_net_rating      -0.10598    0.01996  -5.309 1.92e-07 ***
strength_of_schedule  0.40985    0.02012  20.370  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.106 on 363 degrees of freedom
  (319 observations deleted due to missingness)
Multiple R-squared:  0.8195,    Adjusted R-squared:  0.818
F-statistic: 549.3 on 3 and 363 DF,  p-value: < 2.2e-16
```

Example: for every one point increase in net rating you add .359 to the teams ability.

- Net rating weight = .359
- Away rating Weight = -.106
- Strength of Schedule weight = .410

0.2.1.5 Adjusting Team Abilities with Covariates

We now combine these weights with our covariates to create an updated ability measure for each team. This enhanced model captures a more complete picture of team strength rather than wins/losses alone.

```
unik_teams <- sort(unique(c(games$Home, games$Opponent)))

results <- games %>%
  rename(home.team = Home, away.team = Opponent) %>%
  group_by(home.team, away.team) %>%
  summarise(home.win = sum(Home_Winner), away.win = sum(Opp_Winner),
            .groups = "drop") %>%
  mutate(home.team = factor(home.team, levels = unik_teams),
         away.team = factor(away.team, levels = unik_teams))


lambda_baseline <- BTabilities(fit)

weight_net <- 0.359                                                    ①
weight_away <- -0.106
weight_sos <- 0.410

team_abilities <- team_abilities %>%
  mutate(
    adjusted_ability = baseline_ability +
                       weight_net * net_rating +
                       weight_away * away_net_rating +
                       weight_sos * strength_of_schedule               ②
  )

team_abilities %>%
  mutate(ability_change = adjusted_ability - baseline_ability) %>%
  arrange(desc(abs(ability_change))) %>%
  head(10) %>%
  select(team, baseline_ability, adjusted_ability, ability_change)
```

① Assign weights from regression coefficients
② Compute adjusted ability as weighted sum of covariates

```
                    team baseline_ability adjusted_ability ability_change
1              Life Pacific       -18.475609       -31.509990     -13.034382
2  Mississippi Valley State        -7.011317       -14.287207      -7.275891
3                   Florida         5.322631        12.389578       7.066947
4                    Auburn         4.961668        11.890979       6.929310
5                      Duke         4.413373        11.332369       6.918996
6                  Maryland         2.984890         9.539972       6.555082
7              Florida Tech       -18.925032       -25.406059      -6.481027
8                   Gonzaga         2.413375         8.689324       6.275949
9                 Tennessee         4.414510        10.683470       6.268960
10                  Houston         4.642074        10.910936       6.268863
```

We then have to standardize team ability estimates to have a mean of 0 and a standard deviation of 1. This step keeps the ranking of teams the same but compresses the range of their ability values. Without this step, a few top teams would have unrealistically large scores, causing them to win nearly every simulated matchup. By rescaling, the stronger teams still have higher chances of winning, but upsets remain possible. This captures more of the unpredictability that defines March Madness.

```
team_abilities_clean <- team_abilities |>
  filter(team %in% unik_teams, !is.na(adjusted_ability)) |>
  select(team, baseline_ability, adjusted_ability)

ability_comparison <- team_abilities_clean %>%
  mutate(
    adjusted_before = adjusted_ability,
    adjusted_after = scale(adjusted_ability)[,1]
  ) %>%
  select(team, baseline_ability, adjusted_before, adjusted_after)

team_abilities_clean <- team_abilities_clean %>%
  mutate(adjusted_ability = scale(adjusted_ability)[,1])

summary(ability_comparison)
```

```
     team           baseline_ability   adjusted_before    adjusted_after
 Length:367         Min.   :-19.4922   Min.   :-31.5100   Min.   :-6.57237
 Class :character   1st Qu.: -1.9808   1st Qu.: -2.4778   1st Qu.:-0.54539
 Mode  :character   Median : -1.0194   Median : -0.2214   Median :-0.07697
                    Mean   : -0.7942   Mean   :  0.1494   Mean   : 0.00000
                    3rd Qu.:  0.5559   3rd Qu.:  2.8232   3rd Qu.: 0.55506
                    Max.   :  5.3226   Max.   : 12.3896   Max.   : 2.54102
```

```
summary(team_abilities_clean$adjusted_ability)
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-6.57237 -0.54539 -0.07697  0.00000  0.55506  2.54102
```

```r
team_abilities_clean %>%
  filter(team %in% c("Wisconsin", "Florida", "Duke", "Montana")) %>%
  select(team, baseline_ability, adjusted_ability)
```

```
       team baseline_ability adjusted_ability
1      Duke         4.413373        2.3215452
2   Florida         5.322631        2.5410179
3   Montana         0.000000        0.3255455
4 Wisconsin         2.997643        1.5479021
```

This gives a clear sense of which teams improved or declined once we account for the covariates we added, while also showing how standardization shortens the range of ability values to keep results more balanced and realistic.

0.3 Simulation

0.3.1 Final Data Preparation

Before we simulate the tournament we need to ensure that we have all of the correct teams from the tournament. We make a list of all the teams in the tournament in order of how they show up in the bracket. For example the first 4 teams are Auburn, Alabama State, Louisville and Creighton. This order is very important for our simulation. The winner of the first two teams plays the winner of the second two, and this structure follows all the way to the championship game. After we enter the teams we will filter our possible match ups to only include games with these teams. We can take a look at probabilities the badgers have of beating any other team in the tournament.

```r
march_teams <- c("Auburn", "Alabama State", "Louisville", "Creighton", "Michigan",
                 "UC San Diego", "Texas A&M", "Yale", "Ole Miss",
                 "North Carolina", "Iowa State", "Lipscomb", "Marquette",
                 "New Mexico", "Michigan State", "Bryant", "Florida",
                 "Norfolk State", "UConn", "Oklahoma", "Memphis",
                 "Colorado State", "Maryland", "Grand Canyon", "Missouri",
                 "Drake", "Texas Tech", "UNC Wilmington", "Kansas", "Arkansas",
                 "St. John's", "Omaha", "Duke", "Mount St. Mary's",
                 "Mississippi State", "Baylor", "Oregon", "Liberty", "Arizona",
                 "Akron", "BYU", "VCU", "Wisconsin", "Montana", "Saint Mary's",
                 "Vanderbilt", "Alabama", "Robert Morris", "Houston",
                 "SIU Edwardsville", "Gonzaga", "Georgia", "Clemson", "McNeese",
                 "Purdue", "High Point", "Illinois", "Xavier", "Kentucky",
                 "Troy", "UCLA", "Utah State", "Tennessee", "Wofford")          ①


lambda_adjusted <- as.matrix(team_abilities_clean %>% select(adjusted_ability))
rownames(lambda_adjusted) <- team_abilities_clean$team
colnames(lambda_adjusted) <- "ability"
```

```r
bt_prob_adjusted <- function(team1, team2) {
  diff <- lambda_adjusted[team1, "ability"] - lambda_adjusted[team2, "ability"]
  1 / (1 + exp(-diff))
}

possible_matchups <- expand.grid(
  team1 = march_teams,
  team2 = march_teams,
  stringsAsFactors = FALSE
) %>%
  filter(team1 != team2)                                                    ②

possible_matchups$prob_team1_wins <- mapply(
  bt_prob_adjusted,
  possible_matchups$team1,
  possible_matchups$team2
)

wisconsin_matchups <- possible_matchups %>%
  filter(team1 == "Wisconsin") %>%
  arrange(desc(prob_team1_wins))                                            ③

head(wisconsin_matchups, 10)
```

① A list of all tournament teams in order
② Keep only matchups between tournament teams
③ See Wisconsin's best probabilistic matchups

```
       team1            team2 prob_team1_wins
1  Wisconsin    Alabama State       0.8860904
2  Wisconsin Mount St. Mary's       0.8670540
3  Wisconsin SIU Edwardsville       0.8498938
4  Wisconsin    Norfolk State       0.8249739
5  Wisconsin           Bryant       0.8000228
6  Wisconsin            Omaha       0.7955138
7  Wisconsin          Wofford       0.7946412
8  Wisconsin             Troy       0.7730644
9  Wisconsin          Montana       0.7724780
10 Wisconsin    Robert Morris       0.7602166
```

The last step in our data preparation is creating a data frame that will hold the matchups for round1. We will go region by region for simplicity over our march_teams. As stated before, they are in an order such that every set of two teams is a match up. Our function make_round1 will take each set of 16 teams and return a data frame of the match ups. We will then add these regions together in the order they are added in march_teams, and then add a game column that will enumerate the games to aid in simulation.

```r
make_round1 <- function(teams) {                                                    ①
  seed_pairs <- data.frame(
    team1 = teams[c(1, 3, 5, 7, 9, 11, 13, 15)],
    team2 = teams[c(2, 4, 6, 8, 10, 12, 14, 16)]
  )
  return(seed_pairs)
}

south_round1 <- make_round1(march_teams[1:16])
west_round1 <- make_round1(march_teams[17:32])
east_round1 <- make_round1(march_teams[33:48])
midwest_round1 <- make_round1(march_teams[49:64])

round1_df <- rbind(south_round1, west_round1, east_round1, midwest_round1)         ②
round1_df$game = 1:32                                                               ③
round1_df |> sample_n(10)
```

① Function to pair seeds for a region
② Stack all the regions into one Round of 64 table
③ Add a game number for tracking

```
          team1          team2 game
1      Kentucky           Troy   30
2      Maryland   Grand Canyon   12
3     Wisconsin        Montana   22
4         UConn       Oklahoma   10
5           BYU            VCU   21
6       Gonzaga        Georgia   26
7    Texas Tech UNC Wilmington   14
8       Florida  Norfolk State    9
9       Memphis Colorado State   11
10 Saint Mary's     Vanderbilt   23
```

0.3.2 Simulations

Now that we have our tournament matchups, we can begin simulation. Our first function will be simulate_game. This function will take team1 and team2 playing each other, get the probability team 1 will win from our possible_matchups table and get 1 or 0 from rbinom with that probability. If a 1 is output from this team 1 wins and if 0 team 2 wins.

```r
simulate_game <- function(t1, t2, possible_matchups) {                              ①
  prob_team_1 <- possible_matchups$prob_team1_wins[
    possible_matchups$team1 == t1 & possible_matchups$team2 == t2
  ]
```

```
  outcome <- rbinom(n = 1, size = 1, prob = prob_team_1)                    ②

  if (outcome == 1) {
    return(t1)
  } else {
    return(t2)
  }
}
```

① Take two teams and find win chance
② Flip a weighted coin using that chance

The next step in our simulation will be to simulate a round. Since our matchups are in a numbered order, our function will iterate through each game in our round, simulate the game and store the winner. Since our winners list is also ordered, this means we can easily create a df for the next round granted the previous round was not the championship. To create this next_round we will go through winners in sets of two, and match each two teams up with each other. We will then return this list of winners and the next round data frame.

```
simulate_round <- function(round_df, possible_matchups) {                   ①
  winners = character(nrow(round_df))                                       ②
  for(i in 1:nrow(round_df)) {
    team1 = round_df$team1[i]
    team2 = round_df$team2[i]

    winners[i] = simulate_game(team1, team2, possible_matchups)
  }

  if (length(winners) > 1) {
    next_round <- data.frame(
      team1 = winners[seq(1, length(winners), 2)],
      team2 = winners[seq(2, length(winners), 2)],
      game = 1:(length(winners)/2)
    )
  } else {
    next_round <- NULL
  }


  return(list(
    winners = winners,
    next_round = next_round
  ))
}
```

① Run through all games in the round

② Make a vector to store winners

Finally we have our simulate_tournament function, where we are able to take advantage of the fact that our games take place in a countable order. Before we enter the loop we need to first simulate our round1. We store our winners of round 1 in a list that will hold the winners for each round, then we store the next round data frame. After we have finished the first round, we are able to simulate rounds 2 through the championship. In our loop we add our winners for each round and the next round matchups. This loop will run until we have a champion. We then add names of rounds to our list that stores the winners in each round and return it.

```
set.seed(1121)
simulate_tournament <- function(round1_df, possible_matchups) {                ①
  rounds <- list()

  round_results <- simulate_round(round1_df, possible_matchups)                ②
  rounds[[1]] <- round_results$winners
  next_round_df <- round_results$next_round

  round_num <- 2
  while(!is.null(next_round_df)) {
    round_results <- simulate_round(next_round_df, possible_matchups)          ③
    rounds[[round_num]] <- round_results$winners
    next_round_df <- round_results$next_round
    round_num <- round_num + 1
  }

  names(rounds) <- c("Round Of 32", "Sweet Sixteen", "Elite Eight", "Final Four",
                     "Championship Game", "Champion")
  return(rounds)
}

simulate_tournament(round1_df, possible_matchups)
```

① Run the whole bracket from Round of 64 to the title
② Simulate the first round, then continue
③ Repeat until champion is found

```
$`Round Of 32`
 [1] "Auburn"         "Creighton"      "UC San Diego"   "Texas A&M"
 [5] "Ole Miss"       "Lipscomb"       "New Mexico"     "Michigan State"
 [9] "Norfolk State"  "Oklahoma"       "Memphis"        "Maryland"
[13] "Missouri"       "UNC Wilmington" "Arkansas"       "St. John's"
[17] "Duke"           "Baylor"         "Oregon"         "Arizona"
[21] "BYU"            "Wisconsin"      "Vanderbilt"     "Alabama"
[25] "Houston"        "Georgia"        "Clemson"        "Purdue"
[29] "Illinois"       "Troy"           "Utah State"     "Tennessee"
```

```
$`Sweet Sixteen`
 [1] "Auburn"          "Texas A&M"       "Ole Miss"        "Michigan State"
 [5] "Oklahoma"        "Memphis"         "Missouri"        "Arkansas"
 [9] "Duke"            "Arizona"         "BYU"             "Alabama"
[13] "Houston"         "Purdue"          "Illinois"        "Tennessee"

$`Elite Eight`
[1] "Auburn"          "Michigan State" "Oklahoma"        "Missouri"
[5] "Arizona"         "Alabama"         "Houston"         "Tennessee"

$`Final Four`
[1] "Auburn"   "Oklahoma" "Alabama"  "Houston"

$`Championship Game`
[1] "Auburn"  "Houston"

$Champion
[1] "Auburn"
```

0.4 Results

Now we can simulate this tournament 10,000 times. We will store the results in a data frame that has rows that represent teams and columns representing the rounds they made it to. We will also divide these counts by our number of simulations to get a data frame of probabilities.

```
set.seed(1121)                                                          ①
counts <- as.data.frame(matrix(0, nrow = length(march_teams), ncol = 6))  ②
rownames(counts) <- march_teams
colnames(counts) <- c("Round Of 32", "Sweet Sixteen", "Elite Eight",
                      "Final Four", "Championship Game", "Champion")

n_sims = 10000                                                          ③

for(i in 1:n_sims) {
  results <- simulate_tournament(round1_df, possible_matchups)

  for (round in names(results)) {                                       ④
    counts[results[[round]], round] <- counts[results[[round]], round] + 1
  }
}

probs <- counts / n_sims                                                ⑤
```

① Set seed for repeatable consistent results
② Make a table to count how far each team gets.
③ Number of simulations
④ After each run, add to each team's round count

⑤ Turn counts into probabilities

0.4.1 Probability Tables

```
probs <- probs |> rownames_to_column("Team")
counts <- counts |> rownames_to_column("Team")
counts <- counts |> arrange(desc(Champion))
probs <- probs |> arrange(desc(Champion))
```

These tables show how often each team reached each round (raw counts and probabilities). We can view the first few rows:

```
head(probs)
```

```
       Team Round Of 32 Sweet Sixteen Elite Eight Final Four Championship Game
1   Florida      0.9270        0.7150      0.5133     0.3682             0.2403
2    Auburn      0.9515        0.7215      0.5278     0.3749             0.2330
3      Duke      0.9350        0.6784      0.4836     0.3106             0.1831
4   Houston      0.9199        0.5923      0.4177     0.2449             0.1466
5 Tennessee      0.8760        0.5925      0.3671     0.2167             0.1241
6   Alabama      0.8507        0.5918      0.3940     0.2249             0.1253
  Champion
1   0.1575
2   0.1411
3   0.1001
4   0.0793
5   0.0698
6   0.0660
```

```
head(counts)
```

```
       Team Round Of 32 Sweet Sixteen Elite Eight Final Four Championship Game
1   Florida        9270          7150        5133       3682               2403
2    Auburn        9515          7215        5278       3749               2330
3      Duke        9350          6784        4836       3106               1831
4   Houston        9199          5923        4177       2449               1466
5 Tennessee        8760          5925        3671       2167               1241
6   Alabama        8507          5918        3940       2249               1253
  Champion
1     1575
2     1411
3     1001
4      793
5      698
6      660
```

0.4.2 Wisconsin Performance Snapshot

Here we isolate Wisconsin to see how often they advanced throught the tournament.

```
counts |> filter(Team == "Wisconsin")
```

```
       Team Round Of 32 Sweet Sixteen Elite Eight Final Four Championship Game
1 Wisconsin        7701          4054        1881        835                333
  Champion
1      130
```

```
probs |> filter(Team == "Wisconsin")
```

```
       Team Round Of 32 Sweet Sixteen Elite Eight Final Four Championship Game
1 Wisconsin      0.7701        0.4054      0.1881     0.0835             0.0333
  Champion
1    0.013
```

These show that Wisconsin advanced to the Sweet Sixteen around 40% of the time and only made it to the championship about 3.3% of the time. This suggests that while they often win an early round game, deep tournament runs are extremely rare.

0.5 Visualization

0.5.1 Advancement Probabilities (Top Teams)

We next visualize the probability that each of the top 8 teams advanced to each round

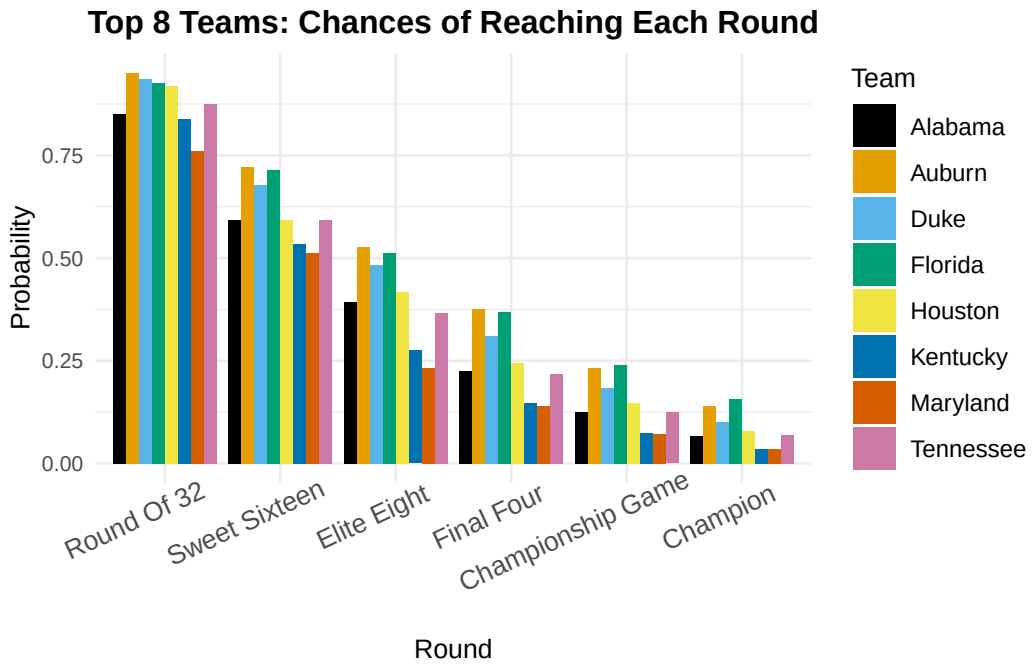
```
round_order <- c(
  "Round Of 32",
  "Sweet Sixteen",
  "Elite Eight",
  "Final Four",
  "Championship Game",
  "Champion"
)

probs_long <- probs |>
  tidyr::pivot_longer(cols = -Team, names_to = "Round",
                      values_to = "Probability") |>
  dplyr::mutate(
    Round = factor(Round, levels = round_order)
  )

top_teams <- probs |>
```

```
  dplyr::arrange(desc(Champion)) |>
  dplyr::slice_head(n = 8) |>
  dplyr::pull(Team)

ggplot(
  probs_long |> dplyr::filter(Team %in% top_teams),
  aes(x = Round, y = Probability, fill = Team)
) +
  geom_col(position = position_dodge(width = 0.90)) +
  scale_fill_manual(values = oi_colors) +
  labs(
    title = "Top 8 Teams: Chances of Reaching Each Round",
    x = "Round",
    y = "Probability",
    fill = "Team"
  ) +
  theme_minimal(base_size = 10) +
  theme(
    plot.title = element_text(face = "bold", hjust = 0.5),
    axis.text.x = element_text(angle = 25, hjust = 0.75, size = 10),
    axis.title.x = element_text(vjust = -0.5),
    legend.position = "right",
    legend.title = element_text(size = 10),
    legend.text = element_text(size = 9),
  )
```

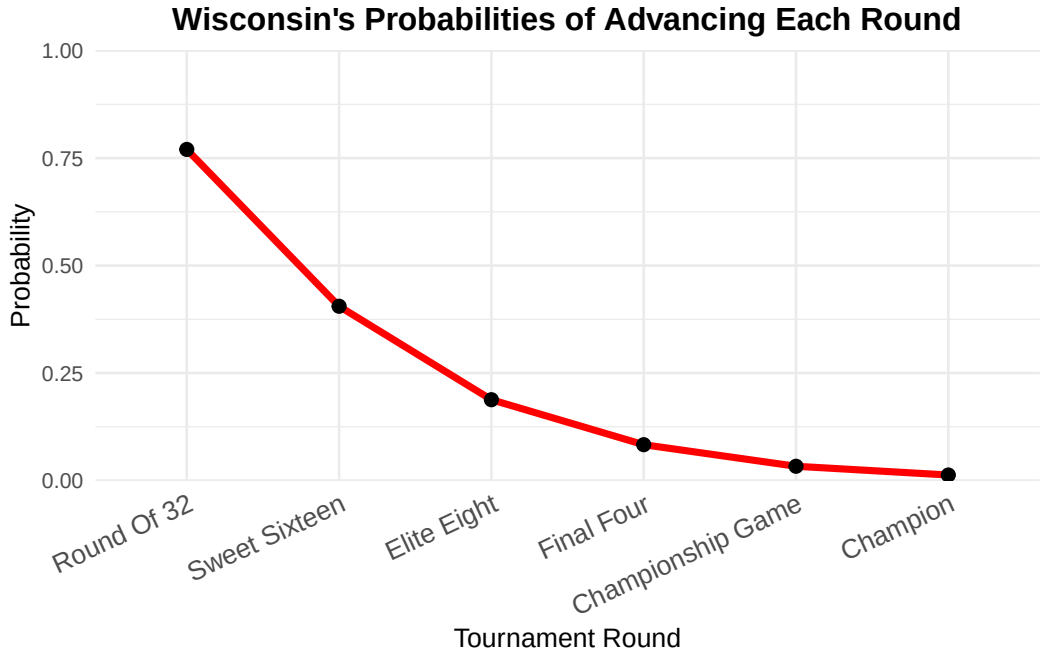


This bar chart highlights how dominant teams like Florida, Auburn, and Duke consistently advance deeper in the tournament.

0.5.2 Wisconsin's Round-by-Round Probabilities

```
wisco_probs <- probs |>
  dplyr::filter(Team == "Wisconsin") |>
  tidyr::pivot_longer(
    cols = -Team,
    names_to = "Round",
    values_to = "Probability"
  ) |>
  dplyr::mutate(
    Round = factor(Round, levels = round_order)
  )

ggplot(wisco_probs, aes(x = Round, y = Probability, group = 1)) +
  geom_line(color = "red", linewidth = 1.25) +
  geom_point(color = "black", size = 2) +
  labs(
    title = "Wisconsin's Probabilities of Advancing Each Round",
    x = "Tournament Round",
    y = "Probability"
  ) +
  scale_y_continuous(limits = c(0, 1), expand = c(0, 0)) +
  theme_minimal(base_size = 10) +
  theme(
    plot.title = element_text(face = "bold", hjust = 0.5),
    axis.text.x = element_text(angle = 25, hjust = 1, size = 10),
  )
```

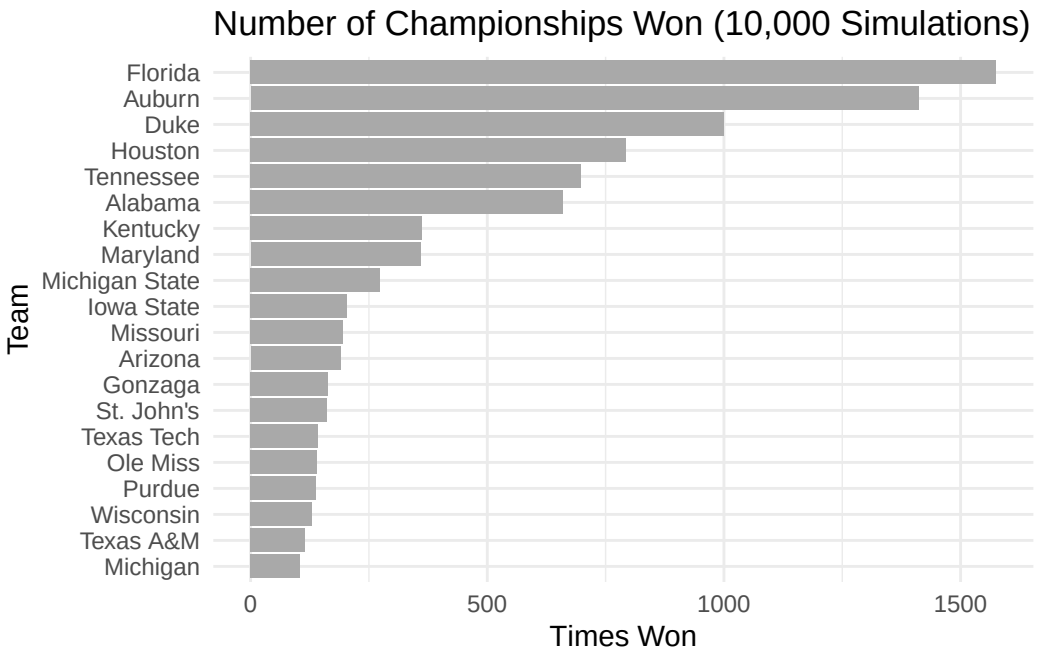


We can see the line starts high and drops sharply after each round, really displaying the difficulty of progressing. By the Elite Eight, probabilities are below 20%, and just above 1% to win the championship.

0.5.3 Championship Wins (Top 20 Teams)

```
counts_long <- counts |>
select(Team, Champion) |>
arrange(desc(Champion)) |>
slice_head(n = 20)

ggplot(counts_long, aes(x = reorder(Team, Champion), y = Champion)) +
geom_col(fill = "darkgrey") +
coord_flip() +
labs(title = "Number of Championships Won (10,000 Simulations)",
x = "Team", y = "Times Won") +
theme_minimal()
```



We can see which teams dominated across the 10,000 simulations. Florida and Auburn clearly stand out as the most frequent champions.

0.6 Next Steps

Future developments could focus on improving model accuracy by including additional advanced metrics and postseason specific factors. Some key statistics, such as KenPom's adjusted offensive and defensive efficiencies, are behind a paywall, limiting access to higher quality measures of team strength and tempo. Including these could significantly improve the precision of our model.

Another limitation is that our model does not account for free throw performance, which often plays a much larger role in postseason games, where possessions slow down and margins tighten. Expanding the model to include free throw rate, turnover percentage, or rebounding efficiency could help capture more of what drives tournament success.

Overall, with more key data and adjustments, this simulation could become an even more powerful tool for evaluating team performance and predicting March Madness outcomes.