Win your office pool with mRchmadness

Eli Shayer and Scott Powers

It's Selection Sunday! The full field of 68 teams for the NCAA men's basketball tournament has been announced, and now that colleague of yours who cares way too much about sports is inviting you to submit a bracket to the office pool. You have a few days to choose one of the 9,233,372,000,000,000,000 possible brackets to call your own. But which one?

Scraping data

Is this the first day you've paid any attention to college basketball all year? We've got you covered, with scraping functions that will tell you everything that went down this season. You can scrape all of the game scores from the 2016-2017 season using the **scrape.game.results** function. In fact, you can use this same function to scrape game results for any other season, past or future, as long as the data are available on ESPN.com. Later, we'll use the data from the current year to predict matchup results.

```
games.2017 = scrape.game.results(2017)
```

It also helps to know who everybody else is picking to advance in each round of the tournament, so that you know what you need to beat. Another function, **scrape.population.distribution**, grabs these data for you, based on the population distribution of picks according to ESPN. Note that these data are currently available online only for the 2016 and 2017 tournaments.

```
pred.pop.2017 = scrape.population.distribution(2017)
```

We've made life even easier for you by pre-scraping **games.2017**, **pred.pop.2017**, **games.2016** and **pred.pop.2017**. These are all available as datasets in the package. You're welcome.

```
head(games.2017)
#>     game.id home.id away.id home.score away.score neutral ot
#> 1 400915164     213     399         81         87       0
#> 2 400916480    2132     399         74         51       0
#> 3 400918604     399      NA         97         56       0
#> 4 400917472    2253     399         82         77       1
#> 5 400918605     225     399         76         80       1
#> 6 400917502     399    2561         81         72       0
head(pred.pop.2017)
#>          name round1 round2 round3 round4 round5 round6
#> 1     Arizona  0.951  0.826  0.658  0.378  0.158  0.068
#> 2   Arkansas  0.496  0.037  0.017  0.006  0.003  0.001
#> 3      Baylor  0.881  0.532  0.144  0.051  0.021  0.008
#> 4    Bucknell  0.118  0.032  0.008  0.003  0.001  0.001
#> 5      Butler  0.889  0.618  0.104  0.036  0.016  0.007
#> 6 Cincinnati  0.746  0.144  0.043  0.014  0.006  0.002
```

Predicting matchup results

Now that you know everything that happened this year, you know how good all of the teams are, right? The Bradley-Terry model can help you make sense of these scores. According to this model, in game i between home team H_i and away team A_i, the number of points y_i (which may be negative) by which the home team wins has the following distribution:

$$y_i \sim \mathcal{N}(\beta_{H_i} - \beta_{A_i}, \sigma^2)$$

where the β's represent the unknown quality of the teams. We provide the **bradley.terry** function to estimate the β's, σ and the corresponding probabilities for each team beating each other team. This function returns a matrix of probabilities, with one row for each team and one column for each team. Each entry of the matrix gives the estimated probability of the team in that row beating the team in that column. Using this, we can estimate for the probability that North Carolina beats Duke, for example.

```
set.seed(1)
probability.matrix = bradley.terry(games = games.2017)
probability.matrix["153", "150"]
#> [1] 0.6091454
```

Not satisfied with this model? Perhaps you'd trust the predictions of the pundits at FiveThirtyEight.com more. We've provided those in a data.frame as well.

```
head(pred.538.2017)
#>          name     round1     round2      round3       round4       round5
#> 8     Arizona 0.95250163 0.56827508 0.392453362 0.1610122161 0.0796408317
#> 30   Arkansas 0.50573824 0.10119193 0.053221267 0.0173209052 0.0059177930
#> 17     Baylor 0.90349888 0.46131029 0.188677847 0.0643593790 0.0288998537
#> 50   Bucknell 0.09817048 0.02844209 0.003276057 0.0007033058 0.0001151156
#> 19     Butler 0.89020591 0.62012561 0.231390822 0.0862911847 0.0329823117
#> 22 Cincinnati 0.61386176 0.25317186 0.100053658 0.0522705796 0.0236312907
#>          round6
#> 8  4.393347e-02
#> 30 1.774174e-03
#> 17 1.399311e-02
#> 50 2.500357e-05
#> 19 1.102163e-02
#> 22 9.309790e-03
```

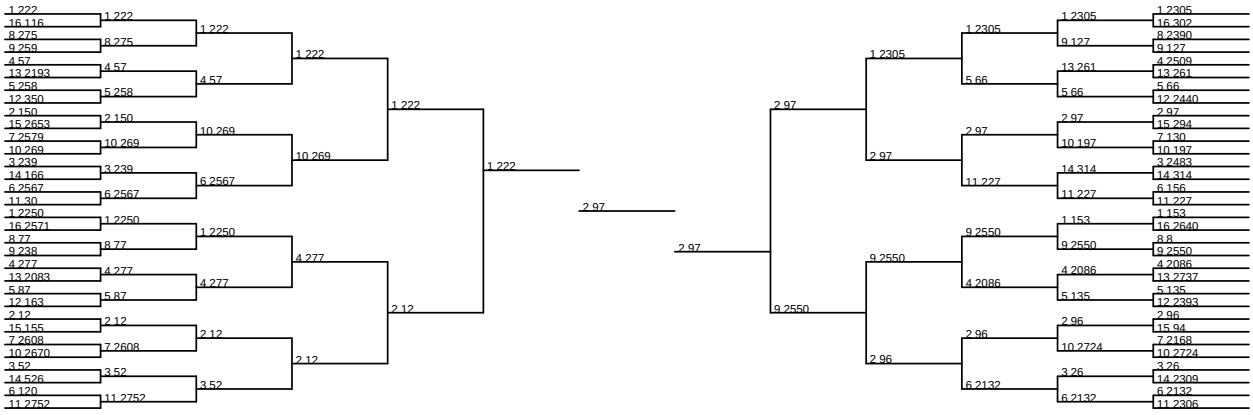
Simulating the tournament

We've already taken care of those pesky first four games for you and stored the bracket in a vector called **bracket.2017**. This lists the tournament teams in order of overall seed.

```
head(bracket.2017)
#> [1] "222"  "2305" "153"  "2250" "12"   "96"
```

If you want to play out your fantasy, though, you can specify any bracket you'd like, as long as it's a character vector of length 64. Once you've done so, you can use the **sim.bracket** function to play out your own personal tournament and the **draw.bracket** function to display the outcome.

```
set.seed(2017)
outcome = sim.bracket(bracket.empty = bracket.2017, probability.matrix = probability.matrix)
draw.bracket(bracket.empty = bracket.2017, bracket.filled = outcome)
```



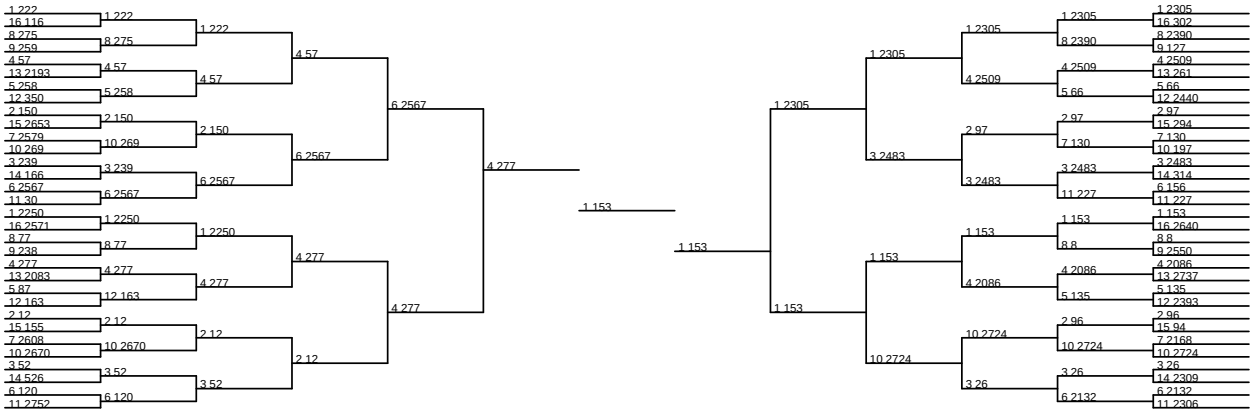
Congratulations are in order to Louisville, for winning the set.seed(2017) NCAA men's basketball tournament!

Finding a good bracket

That's all well and good for Louisville, but wouldn't you prefer to win something yourself? Now you can, using the **find.bracket** function. This function produces a number (*num.candidates*) of candidate brackets and then evaluates each of them across a number (*num.sims*) of simulations. The larger you make *num.candidates* and *num.sims*, the better the bracket will be, but at the cost of increased computation time. You can choose whether you want the bracket which maximizes your expected score, expected percentile within your pool, or even the probability of winning your pool!

You can also customize these results based on the scoring rules and size (excluding you) of your pool. Below we search for a good bracket to use if we want to maximize our chances of winning a pool with 30 other people in it, using the default scoring rules for CBS Sports.

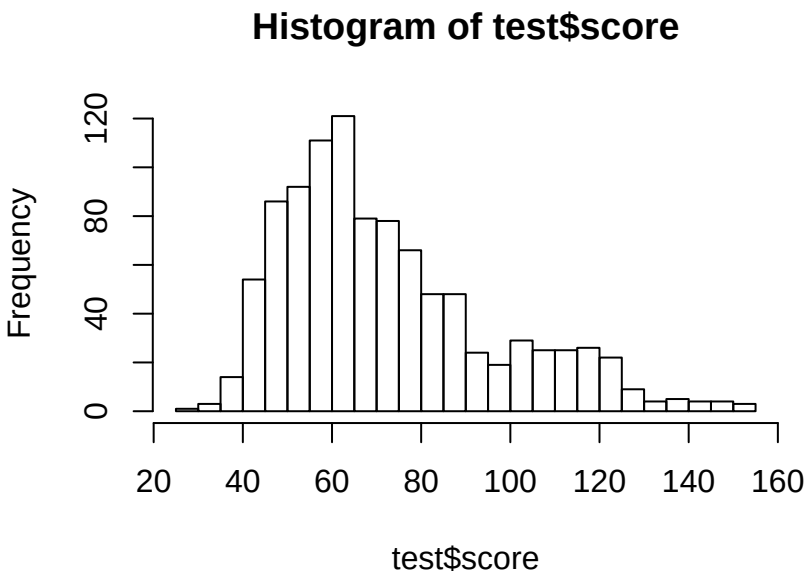
```
set.seed(42)
my.bracket = find.bracket(bracket.empty = bracket.2017, probability.matrix = probability.matrix,
                          num.candidates = 100, num.sims = 1000, criterion = "win",
                          pool.size = 30, bonus.round = c(1, 2, 4, 8, 16, 32),
                          bonus.seed = rep(0, 16), bonus.combine = "add")
draw.bracket(bracket.empty = bracket.2017, bracket.filled = my.bracket)
```

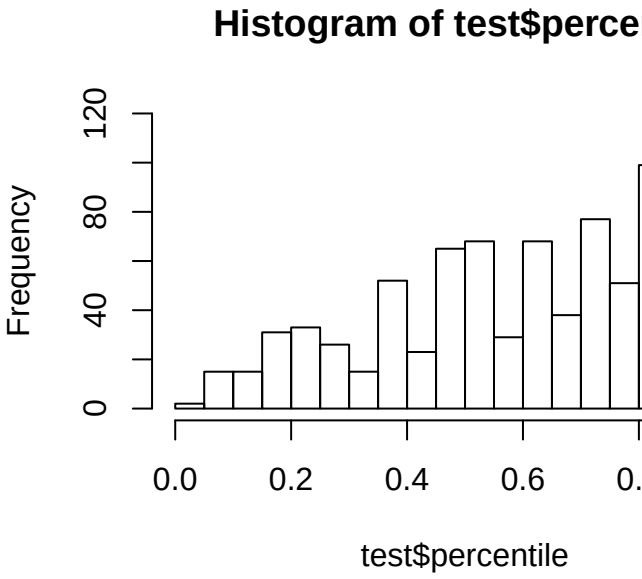


Testing your bracket

So you're picking North Carolina? Good choice. Let's see how well we can expect this bracket to do, using the **test.bracket** function. This function simulates your pool to determine your expected score, expected percentile and probability of winning.

```r
set.seed(8675309)
test = test.bracket(bracket.empty = bracket.2017, probability.matrix = probability.matrix,
                    bracket.picks = my.bracket, pool.size = 30, num.sims = 1000,
                    bonus.round = c(1, 2, 4, 8, 16, 32), bonus.seed = rep(0, 16),
                    bonus.combine = "add")
hist(test$score, breaks = 20)
hist(test$percentile, breaks = 20)
mean(test$win)
#> [1] 0.054
```





There you have it. You can expect this bracket to win 5.4% of the Groundhog Day replays of your bracket pool. Here's to the "real" outcome being one of those 5.4%!