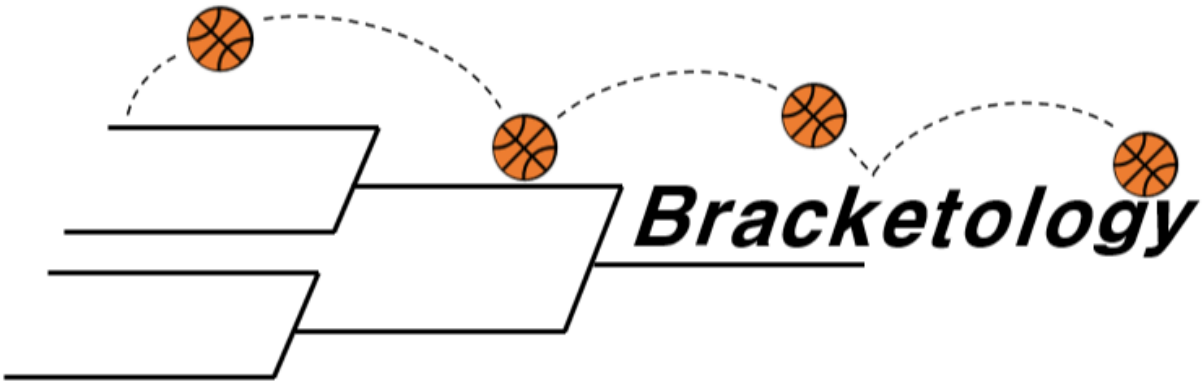
bracketology

Release 0.0.99

Mar 09, 2020

Contents:



The goal of bracketology is to speed up the analysis of NCAA march madness data and help develop algorithms for filling out brackets.

Documentation https://bracketology.readthedocs.io/en/latest/

GitHub Repo https://github.com/stahl085/bracketology

Issue Tracker https://github.com/stahl085/bracketology/issues

Backlog https://github.com/stahl085/bracketology/projects/1?fullscreen=true

PyPI https://pypi.org/project/bracketology/

CHAPTER 1

Before You Start

Here are the main things you need to know:

- The main parts of this package are the `Bracket` objects and simulator functions in the `simulators` module
- A Bracket is composed of `Team` and `Game` objects
- Game objects have two Team objects as attributes, and the round number
- Teams have a name, seed, and dictionary for statistics
- Simulator functions have 1 argument of type Game, and return the winning Team of that Game

CHAPTER 2

Installation

Install from pip

```
pip install bracketology
```

Or download directly from PyPi

CHAPTER 3

Getting Started

Import bracketology and create a bracket from last year.

```
from bracketology import Bracket, Game, Team

# Create a bracket object from 2019
year = 2019
b19 = Bracket(year)
```

CHAPTER 4

Tutorial

4.1 Inspecting the Bracket Object

Here are three different ways you can inspect the Bracket.

- Inspect teams in each region (dictionary of actual results)
- Inspect actual results by round (dictionary)
- Inspect simulated results by round (list of Team attributes)

4.1.1 Get Teams in each Region

Print out all the teams in each region. The *regions* attribute is a dictionary with the information of all the teams in each region.

```
>>> print(b19.regions)
{
    'East': [{'Team': 'Duke', 'Seed': 1},
             {'Team': 'Michigan St', 'Seed': 2},
             {'Team': 'LSU', 'Seed': 3},
             ...],
    'West': [{'Team': 'Gonzaga', 'Seed': 1},
             {'Team': 'Michigan', 'Seed': 2},
             {'Team': 'Texas Tech', 'Seed': 3},
             ...],
    'Midwest': [{'Team': 'North Carolina', 'Seed': 1},
                {'Team': 'Kentucky', 'Seed': 2},
                {'Team': 'Houston', 'Seed': 3},
                ...],
    'South': [{'Team': 'Virginia', 'Seed': 1},
              {'Team': 'Tennessee', 'Seed': 2},
              {'Team': 'Purdue', 'Seed': 3},
```

(continues on next page)

(continued from previous page)

```
                ...]
}
```

4.1.2 Actual Results by Round

The *result* attribute will return a dictionary (similar to *regions* above) but will be broken out by which teams actually made it to each round. You can use it to inspect the real tournament results.

```
>>> print(b19.result.keys())
dict_keys(['first', 'second', 'sweet16', 'elite8', 'final4', 'championship', 'winner
↪'])

>>> print(b19.result['final4'])
[{'Team': 'Michigan St', 'Seed': 2}, {'Team': 'Virginia', 'Seed': 1},
 {'Team': 'Texas Tech', 'Seed': 3}, {'Team': 'Auburn', 'Seed': 5}]

>>> print(b19.result.get('winner'))
{'Team': 'Virginia', 'Seed': 1}
```

4.1.3 Simulation Results by Round

Print out all the teams that are simulated to make it to each round. The first round is filled out by default. This is a list of *Team* objects that are simulated to make it to each round. Right now *round2* is an empty list because we have not simulated the bracket yet.

```
>>> print(b19.round1)
[<1 Duke>, <2 Michigan St>, <3 LSU>, ... , <1 Gonzaga>, <2 Michigan>, <3 Texas Tech>,
 ... , <1 North Carolina>, <2 Kentucky>, <3 Houston>, ... , <1 Virginia>, <2␣
↪Tennessee>, <3 Purdue>]

>>> print(b19.round2)
[]
```

4.2 Creating a Simulator Algorithm

A simulator function needs to take in a *Game* and Return a *Team*.

First we create some faux teams and games to test our simulator function on.

```
# Create teams
team1 = Team(name='Blue Mountain State',seed=1)
team2 = Team(name='School of Hard Knocks',seed=2)

# Create a game between the teams
game1 = Game(team1, team2, round_number=1)
```

Then we define the simulator function.

```
import random
def pick_a_random_team(the_game):

```

(continues on next page)

(continued from previous page)

```
    # Extract Teams from Game
    team1 = the_game.top_team
    team2 = the_game.bottom_team

    # Randomly select a winner
    if random.random() < 0.5:
        winner = team1
    else:
        winner = team2

    # Return the lucky team
    return winner
```

Test the function out on a game.

```
>>> pick_a_random_team(game1)
<2 School of Hard Knocks>
```

Let's run some simulations with our function!

```
# Initialize Simulation Parameters
BMS_wins = 0
HardKnocks_wins = 0
n_games = 1000

# Loop through a bunch of games
for i in range(n_games):

    # Simulate the winner
    winner = pick_a_random_team(game1)

    # Increment win totals
    if winner.seed == 1:
        BMS_wins += 1
    elif winner.seed == 2:
        HardKnocks_wins += 1
    else:
        raise Exception("We have a tie??")

# Calculate total win percentage
BMS_win_pct = round(BMS_wins/n_games, 4) * 100
HardKnocks_win_pct = round(HardKnocks_wins/n_games, 4) * 100

# Print out results
print(f"Blue Mountain State Win Percentage:   %{BMS_win_pct}")
print(f"School of Hard Knocks Win Percentage: %{HardKnocks_win_pct}")
```

Output:

```
Blue Mountain State Win Percentage:   %50.9
School of Hard Knocks Win Percentage: %49.1
```

4.3 Evaluting Simulator Results

Let's evaluate our simulator function on some actual brackets.

```
# Initialize simulation parameters
n_sims = 1000 # number of times to simulate through all years
total_sims = (n_sims * len(brackets))
scores = []
correct_games = []

# Loop through a plethora of brackets
for i in range(n_sims):
    for bracket in brackets:

        # Run the algorithm on the bracket
        bracket.score(sim_func=pick_a_random_team, verbose=False)

        # Save the scoring results in a list
        scores.append(bracket.total_score)
        correct_games.append(bracket.n_games_correct)

# Calculate the average across all simulations
avg_score = round(sum(scores) / total_sims)
avg_correct = round(sum(correct_games) / total_sims)

# Print result
print(f"Average number total score {avg_score}/192")
print(f"Average number of games guessed correctly {avg_correct}/64")
```

Output:

```
Average number total score 31/192
Average number of games guessed correctly 21/64
```

Easy, right!

CHAPTER 5

Reference

5.1 Bracket Objects

class bracketology.**Bracket**(*year*)

A NCAA tournament for a specific year

year

Calendar year of the tournament (1985-2019)

Type int

result

The actual tournament results for that year

Type (dict)

regions

The teams that year broken down by region

Type (dict)

East

SubBracket for East

Type (*SubBracket16*)

West

SubBracket for West

Type (*SubBracket16*)

Midwest

SubBracket for Midwest

Type (*SubBracket16*)

South

SubBracket for South

Type (*SubBracket16*)

Finals
Final Four and Championship games

Type (*FinalFour*)

round1, round2, ... , round6
Which teams are simulated to make it to each round

Type (list of Teams)

winner
Simulated tournament winner

Type *Team*

n_games_correct int
Number of games the simulation got correct

total_score int
Total points earned by the simulator function (32 points per round)

__init__(*year*)

Parameters **year** (*int*) – Year of the NCAA tournament

score(*sim_func=None*, *verbose=True*)
Calculates the number of games correct from the simulation and that total score (32 points per round). Will run a new simulation as well if passed *sim_func* argument.

Parameters

- **sim_func** (*function, optional*) – A function that take in *Game* and returns a *Team* of that Game. Can be null if the bracket has already been simulated
- **verbose** (*bool, optional*) – Whether or not to print the score. If False, will not print score, only sets the *n_games_correct* and *total_score* parameters. The default is True.

sim(*sim_func*)
Simulate the entire bracket with *sim_func*, from first round to deciding the winner

Parameters **sim_func** (*function*) – A function that take in *Game* and returns a *Team* of that Game

class bracketology.**SubBracket16**(*region*)
A region, or sub-bracket of 16 teams for a NCAA tournament

region
Name of the region for the bracket

Type str

team01, ..., team16
Each team in the subracket named for its seed

Type *Team*

Game1, ..., Game15
The games that make up the bracket. Round 1 is 1-8, Round 2 is 9-12, Sweet 16 is 13 and 14, Final Four is 15

Type *Game*

round1, ..., round6
List of games in each round

Type list

winner
The team from this region that is simulated to make the final four

Type *Team*

__init__(*region*)

Parameters **region** (*str*) – Name of the region for this sub bracket.

initialize_first_round(*teams*)

run_bracket(*sim_func*)

class bracketology.**FinalFour**(*year*)
A bracket of four teams to simulate the final four of a NCAA tournamnet

Game1
Final four game for regions 1 and 3

Type *Game*

Game2
Final four game for regions 2 and 4

Type *Game*

Championship
Final game of tournament to determine the simulated winner

Type *Game*

winner
The team from this region that is simulated to make the final four

Type *Team*

__init__(*year*)

Parameters **year** (*int*) – Year of the NCAA tournament

set_matches(*teams*)

class bracketology.**Game**(*top_team*, *bottom_team*, *round_number*)
A game between two teams in the bracket

top_team
The "top team" in the game refers to bracket position, not seed

Type *Team*

bottom_team
The "bottom team" in the game refers to bracket position, not seed

Type *Team*

round_number
Which round of the tournament is it (1-6)

Type int

__init__(*top_team*, *bottom_team*, *round_number*)

Parameters

- **top_team** (Team) – The top team of the game, not by seed but by position on the bracket. Closest to the top left

- **bottom_team** (Team) – Team closest to the bottom right of the bracket
- **round_number** (*int*) – Round number, first round is 1, championship game is 6

class bracketology.**Team**(*name*, *seed*)
A team that is in a Game or a Bracket

name
Name of the school (or abbreviation)

Type str

seed
Seed of the team in the tournament (1-16)

Type int

stats
A dictionary with other information about the team, like season stats

Type (dict)

__init__(*name*, *seed*)

Parameters

- **name** (*str*) – The name of the school for the team.
- **seed** (*int*) – The seed of the team in the bracket.

5.2 Simulators

bracketology.**simulators**
alias of *bracketology.simulators*

class bracketology.simulators.**upset_prob**
Given a probability between 0-1 will return a function that can be as an algorithm to fill out an NCAA bracket with *p* as the probability of an upset

Parameters **p** (*float*) – The probability of an upset

Returns **scoring_func** – function to pick an upset of a Game with probability *p*

Return type function

Index

Symbols

B

C

E

F

G

I

M

N

R

S

T

U

W

Y