college-basketball-elo

Inspired by the folks at FiveThirtyEight who leverage Elo rating systems to make their NBA and NFL predictions, I created my own version of the Elo rating system for NCAA Men's College Basketball. FiveThirtyEight has this as well, but they don't expose the inner-workings of their model like they do for the NBA and NFL. Additionally, they use Elo as only one component in their ensemble methodology for NCAA predictions - and their predictions are only available during March Madness.

This version of the Elo rating system is primarily based on FiveThirtyEight's NBA methodology with some tuning adjustments. Game log data is from Sports Referenceand goes back to the start of the 2010 season. You can use these scripts to see team ratings, predict individual games, predict tournaments, or even simulate entire tournaments thousands of times as of any date in the last 10 years. This documentation walks through usage, methodology, tuning, and performance.

Usage

There are several ways to use this program. They are organized into two scripts, each with several optional inputs to control functionality.

elo.py

This script allows you to view the top elo-rated teams at any given time since the start of the 2010 season. In addition to specifying the historical date of interest, you also have the ability to control multiple aspects of the output with optional arguments.

```
python3 elo.py
```

By default, the script will:

1. Refresh data through games completed yesterday
2. Calculate the latest team Elo ratings
3. Open a Plotly table displaying the top 25 teams, their Elo ratings, and their change in rating from 7 days earlier. Also a part of the table is *Point Spread vs. Next Rank*. This indicates by how many points a team would be favored over the next team in the rankings. In the example below, Gonzaga would be a 4.0 favorite over Illinois, Illinois a 1.1 point favorite over Baylor, Baylor a 2.7 point favorite over Iowa, and so on.

Ratings through 20210314

Rank	Team	Elo Rating	Point Spread vs. Next Rank	7 Day Change
1	Gonzaga	2308	-4.0	+9
2	Illinois	2207	-1.1	+38
3	Baylor	2179	-2.7	-36
4	Iowa	2109	-0.1	-12
5	Alabama	2107	-0.0	+32

The script can be run with the following options:

- `-h` or `--help` : Displays the help messages for this script

- `-t` or `--topteams` : Specify with an integer how many of the top teams to output. The default is to display the top 25
- `-d` or `--date` : Use to see the top teams as of a date in the past. Enter date as a YYYYMMDD integer (e.g. 20190320). The default is to calculate through the last game in the most recent data
- `-p` or `--period` : The default is to display the Elo ratings change over the last 7 days in the table. Specify an integer number of days to use for that column instead of 7

predictions.py

This script allows you to make a number of different types of predictions using the Elo model. You can predict all scheduled games on a future day, ad-hoc individual matchups, populate a traditional bracket with picks, or simulate a tournament to get each team's chances of making it to each round. Similar to elo.py, you have the option to specify a historical date of interest along with a number of other options depending on the desired functionality.

Example 1

```
python3 predictions.py -F '20210327'
```

This example demonstrates the default functionality of the script. This will return predictions for all games scheduled according to Sports Reference on the date specified date with the `-F` flag. If no date is specified (`python3 predictions.py`) predictions will be made based on today's schedule on Sports Reference. The output is a Plotly table showing each matchup, each team's win probability, and predicted spread. There is also an indicator for whether or not the game is played at a neutral site. The output tables are saved in the `Outputs` folder as a csv for further examination. Running this script before 3/27/2021, we can use the latest version of the model to predict the outcomes of games scheduled for that day:

20210327 Game Predictions Based on Ratings through 20210322

Neutral	Home	Home Win Prob.	Home Pred. Spread	Away	Away Win Prob.	Away Pred. Spread
1	Villanova	29%	6	Baylor	71%	-6
1	Syracuse	36%	3.9	Houston	64%	-3.9
1	Oral Roberts	14%	12.1	Arkansas	86%	-12.1
1	Oregon State	34%	4.4	Loyola (IL)	66%	-4.4

You can also quickly see what the model would have picked for all games on a day in the past by setting `-F` to the date of the games and `-d` to the date on which the predictions would have been made. This is how you can quickly see what the model would have picked for every game on 3/22/2021 based on what we knew through 3/21/2021:

```
python3 predictions.py -F '20210322' -d '20210321'
```

Example 2

```
python3 predictions.py -G 'UConn' 'Maryland' -n
```

In this example we want to predict the result of the first round matchup in the 2021 NCAA Tournament between UConn and Maryland. No date is specified, so the model will update the data through games completed yesterday and simulate as if this game is played tomorrow (which it is at the time of writing). The `-n` flag indicates the game is played at a neutral site, so no home-court advantage is applied to UConn. This returns that UConn is favored by 2.5 points (Vegas says 3):

```
Ratings through 20210319
UConn vs. Maryland @ neutral site
UConn 59% -2.5 Maryland 41% 2.5
```

The advantage of this use over the default `predictions.py` functionality is you can predict the results of a hypothetical matchup or one that is not scheduled on Sports Reference.

Example 3

```
python3 predictions.py -P 'Data/tournament_results_2021.csv' -m 1
```

In this example, we want the model to predict the outcomes of a tournament with the matchups specified in `tournament_results_2021.csv` by selecting the team it thinks is most likely to win each of the 63 games in a traditional March Madness tournament. `tournament_results_2021.csv` must have a power of 2 (e.g. 64, 32, 128) team names in its first column in matchup order. Row 1 plays row 2. The winner of that matchup plays the winner of row 3 vs. row 4, and so on until there is only 1 team remaining. It does not matter what is actually in `tournament_results_2021.csv` outside of that first column. The script will predict a bracket from that starting point. The `-m` flag specifies how the model should determine the winner of each matchup. In this case, `1` tells the model to always select the team that is favored by Elo rating. `0` would tell the model to choose probabilisticaly (default) and `2` would choose randomly.

As an output, you will see your original first column, then half the number of teams in the second column, a quarter in the third, and so on until a column has just 1 name. A team that advances from one column to the next is deemed to have won their matchup. The last team remaining (in the rightmost column) is the champion. Next to each team name will be the percent probability that this team won their previous matchup. In this example, you see Gonzaga is predicted as the 2021 champion and had a 64% chance of winning their matchup with Illinois in the finals. If `-m` was set to `0`, Gonzaga would have a 64% chance of being picked to win this matchup. Instead, since `-m` was set to `1`, Gonzaga was picked automatically because their Elo rating is greater than Illinois'. The output tables are saved in the `Outputs` folder as a csv for further examination.

Tournament Predictions Based on Ratings through 20210314

first	second	sixteen	eight	four	final	champion
Gonzaga	Gonzaga,98%	Gonzaga,90%	Gonzaga,81%	Gonzaga,76%	Gonzaga,76%	Gonzaga,64%
Appalachian State	Oklahoma,57%	Virginia,60%	Iowa,52%	Alabama,50%	Illinois,54%	
Oklahoma	Creighton,76%	Kansas,69%	Michigan,60%	Baylor,66%		
Missouri	Virginia,81%	Iowa,70%	Alabama,56%	Illinois,65%		
Creighton	USC,51%	Michigan,65%	Baylor,71%			

Example 4

```
python3 predictions.py -S 'Data/tournament_results_2019.csv' 10000 -d '20190320'
```

In this example, we are simulating 10,000 possible outcomes for the 2019 March Madness tournament based on the Elo ratings from 3/20/2019 (the day before the tournament started). The `-d` flag, which is optional, was used here to roll the ratings back to the specific date where we would have made the predictions prior to the 2019 tournament. If it was not specified, the simulations would be for the 2019 tournament matchups, but with today's Elo ratings (which are not as favorable to Duke, for example). The `-d` flag can also be used with the functionality demonstrated in examples 1, 2, and 3.

As an output you will see one column for each round of the tournament a team could possibly make it to - including being named champion. Each row tells us in what share of simulations that team made it to the corresponding round. For example, in ~63% of simulations, Virginia made it to the Elite 8. In ~15%, they won the whole thing (which they did end up doing). The output is sorted by this final column. The winners of each matchup in each simulation are chosen probabilistically. The output tables are saved in the `Outputs` folder as a csv for further examination.

Tournament Predictions Based on Ratings through 20190320 and 10000 Simulations

team	second	sixteen	eight	four	final	champion
Virginia	0.9595	0.8294	0.6331	0.4201	0.2495	0.148
Duke	0.9613	0.8245	0.6275	0.3956	0.2537	0.1447
UNC	0.958	0.801	0.5564	0.3512	0.2148	0.1252
Michigan State	0.9325	0.7497	0.5622	0.3225	0.1948	0.1061
Kentucky	0.9373	0.7023	0.4723	0.2631	0.1462	0.0804

For completeness, `predictions.py` can be run with the following options:

- `-h` or `--help` : Displays the help messages for this script
- `-F` or `--ForecastDate` : Use if trying to predict games for a particular date in the future, set `-F` equal to the day you want to predict. If games are scheduled on Sports Reference, the output will be predictions for all games. `-F` can also be used on dates in the past, just make sure to also specify `-d` - the day through which the model is allowed to use information in making its predictions. Enter date as YYYYMMDD (e.g. 20190315).
- `-G` or `--GamePredictor` : Use this to predict a single game. Supply a home team and an away team - both as strings. Use `-n` flag to indicate a neutral site. This can be used in conjunction with the `-d` flag to make predictions as they would have been made in the past
- `-n` or `--neutral` : If provided and using `--GamePredictor` , this will indicate the matchup should be simulated as if the teams are at a neutral location. No advantage will be given to the home team (the first team listed)
- `-S` or `SimMode` : Use this to run monte carlo simulations for a tournament and see in what share of simulations a team makes it to each round. Enter the filename storing the tournament participants as a string and an integer number of simulations to run. Don't forget to use `-d` if predicting this tournament as of a date in the past
- `-d` or `--dateSim` : Specify a date in the past on which to make predictions. `-d` can be thought of as the "stop short" date. The model will freeze Elo ratings on the date specified to simulate what it would be like to make predictions as of that date. Enter date as a YYYYMMDD integer (e.g. 20190320). If not specified, the default is to calculate through the last game in the most recent data. `-d` can be specified at any time (making future/past game predictions for a day, single game predictions, single bracket predictions, or simulating bracket outcomes)
- `-P` or `--PredictBracket` : Use to predict results of a tournament (i.e. generate a single bracket). Enter the filename storing the tournament participants in the first column. Use the `-m` flag to specify how each matchup should be decided. Don't forget to use `-d` if predicting this tournament as of a date in the past
- `-m` or `--mode` : By default, the winner for each matchup in a tournament prediction is selected probabilistically (mode 0). Use `1` to have the model always pick the 'better' team according to

Elo ratings. Use `2` to decide each matchup with a coinflip (random selection)

Methodology

The methodology for Elo rating systems is discussed in much more detail in several sources on the web, including the Elo rating systems Wikipedia page. This particular implementation was most closely guided by FiveThirtyEight methodologies for the NBAand NFL.

Elo ratings are useful for addressing differences in strength of schedule. When you get to the end of the season and see a 22-6 team playing a 14-1 team, your instinct might be to think the 14-1 team is favored. They lost only one game all season. However, that 14-1 Colgate team only played 5 different opponents all season - and none of them are particularly challenging opponents. Meanwhile, that 22-6 Arkansas team lost several games, but also played one of the tougher schedules in the country - including 4 games against top-25 teams. Fortunately, Elo accounts for these differences. Colgate's wins don't earn them that many Elo points because they mostly played inferior opponents. Arkansas' wins earned them a lot of points because they were against some teams where they were not expected to win. Thus, Arkansas is heavily favored.

In a particular matchup, the difference in ratings can be used to predict both a win probability and a point spread. Arkansas' Elo rating is 2032 while Colgate's is 1849. The difference in ratings is +183 for Arkansas. We can calculate win probability as:

```
1 / (1 + 10**(-elo_spread/400))
```

Arkansas' win probability is 74% when we plug in +183 for `elo_spread`. Using -183 for `elo_spread`, we find that Colgate's win probability is 26% `(100% - 76%)`. To convert this difference in Elo ratings to a point spread we can divide +183 by **-25.5**. This tells us Arkansas is a -7.2 point favorite (Vegas says -8.5).

An important part of any sporting event is home-court advantage. Teams receive an automatic boost of **+83** Elo points when they are at home. This implies a 3.25-point advantage `(83/25.5)` at home. No advantage is applied when games are played at neutral locations.

One major advantage of an Elo system is its simplicity. Elo ratings are updated simply by the final results of games. Teams with higher Elo ratings are perceived as better than teams with lower ratings. A team's rating changes after every game they play. The rating goes up after a win and down after a loss. There are two factors that determine by how much it goes up or down:

1. Elo Difference - A win over an opponent with an Elo rating that is much higher than yours is more meaningful than a win over a team you are rated much higher than and expected to beat. Essentially, bigger upset wins are worth more. On the flipside, more surprising losses to much worse teams will cost more rating points.
2. Margin of Victory (MoV) - A win by 20+ points is more meaningful than a 1-point win. The more you win by, the more points you are rewarded for the win. On the flipside, the more you lose by, the more points you can expect to lose. The **Margin-of-Victory multiplier** is calculated as:

```
((MoV + 2.5)**.7) / (6 + .006 * elo_margin)
```

The best way to summarize how Elo ratings get adjusted, is to assume that they are constantly trying to converge on accuracy. They want to change as little as possible. If a result is surprising according to the ratings, then the ratings were clearly very wrong, so they should make dramatic changes. However, if the result is about what is expected, there won't be much to adjust. The ratings are close to accurate.

The full formula for updating a team's rating is best understood with an example. Let's consider the SEC championship between Alabama and LSU on 3/14/2021. Alabama's Elo rating was 2101 at the time, and LSU's was 2003. We can calculate Alabama's win probability at this neutral-site game is 64%. The final score was Alabama 80, LSU 79. To adjust Alabama, we find that the Elo difference was +98 `(2101 - 2003)` and the MoV was +1. Using the formula above, we find the Margin-of-Victory multiplier to be 0.36. We also need a constant called a K-factor that influences how much each game should adjust the ratings. Our optimal K-factor for every game is **47**. Putting it all together:

```
winner_change_in_elo = k_factor * MoV_multiplier * (1 - winner_win_probability)
+6 = 47 * 0.36 * (1 - 0.64)
```

We adjust Alabama's rating up 6 points to 2107 and LSU's down by 6 to 1997. Alabama still gets points for winning the game. However, they were expected to win by even more than they did (the model had them at 3.8-point favorites), so they don't get that many points. In contrast, we can look at Georgetown's 73-48 win over Creighton in the Big East Championship on 3/13/2021. Georgetown was a major underdog: 1903 Elo rating vs. 2053 with a win probability of 30%. Not only did they pull of the upset (according to the model), but they won by *a lot*. Thus, the model adjusted heavily:

```
winner_change_in_elo = k_factor * MoV_multiplier * (1 - winner_win_probability)
+66 = 47 * 2.0 * (1 - 0.30)
```

Elo ratings are meant to be a closed system. Upon initialization, each team starts at an Elo rating of **1500**. The total number of points in the system should not change (1500 * the total number of teams at the start). Therefore, the number of points gained by the winner of a game equals the number of points lost by the loser of the game. In practice, the total number of points in this system does not remain constant because new teams are constantly joining as new teams make the jump to D1 and D1 schools expand their schedules to non-D1 teams. These late-joiners we should assume are not very good. Thus, they start with an Elo rating of **1050** (i.e. they are automatically 17.5-point underdogs against an average opponent).

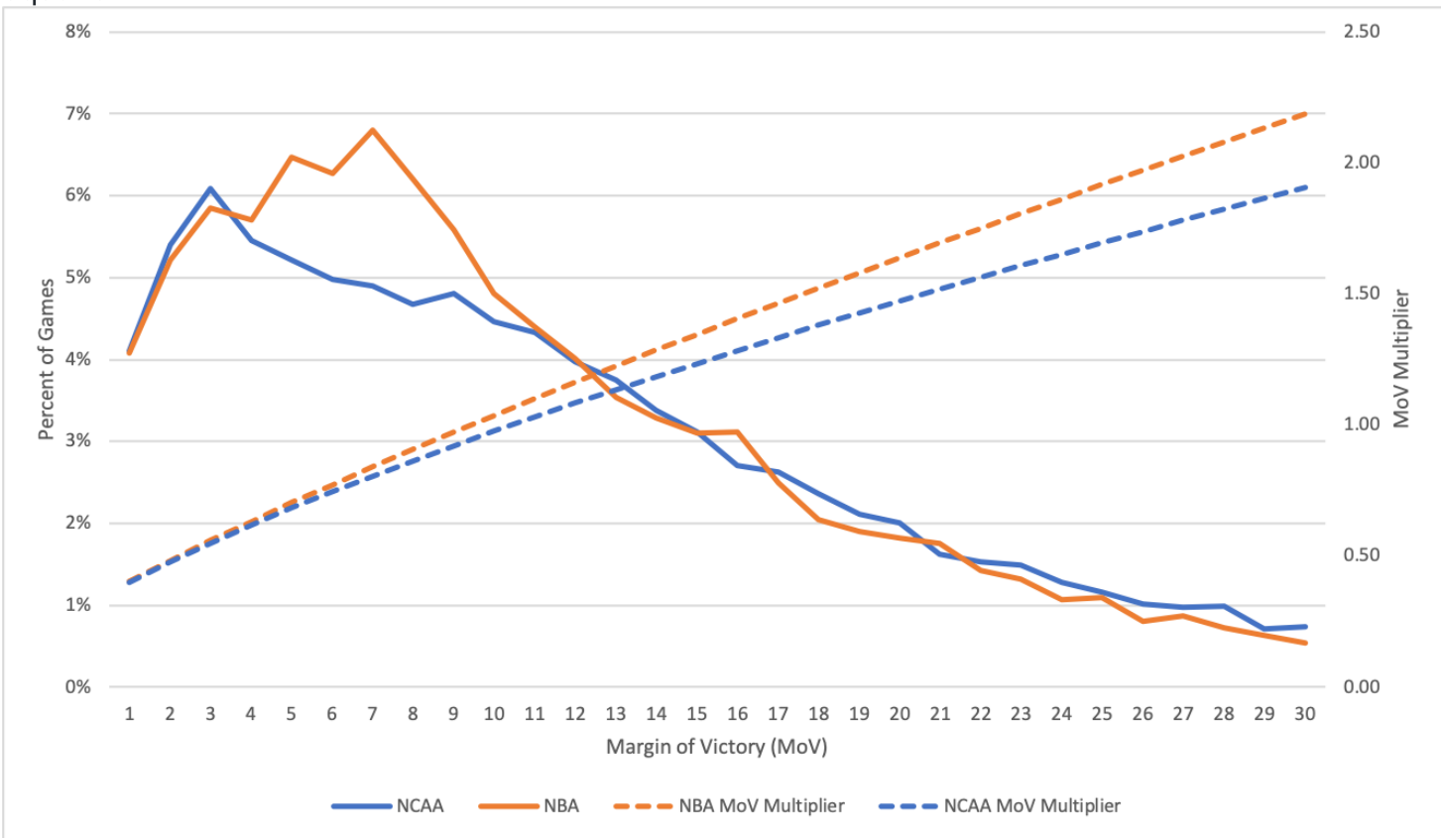
In most Elo systems, there is some adjustment made at the end of every season to revert teams back to average. It makes sense that as some players graduate and new players come in, we should be less confident in our current rating for a team. Really good teams should be adjusted down a litte. Really bad teams should be adjusted up a little. FiveThirtyEight, for example, starts every new season by taking `0.75 * end_of_season_rating + 0.25 * league_average_rating`. Their new season carryover is 0.75. However, my Elo model does not make end-of-season adjustments like this, so the new season carryover is **1.0**. The reasons are discussed in the tuning section.

Tuning and Performance

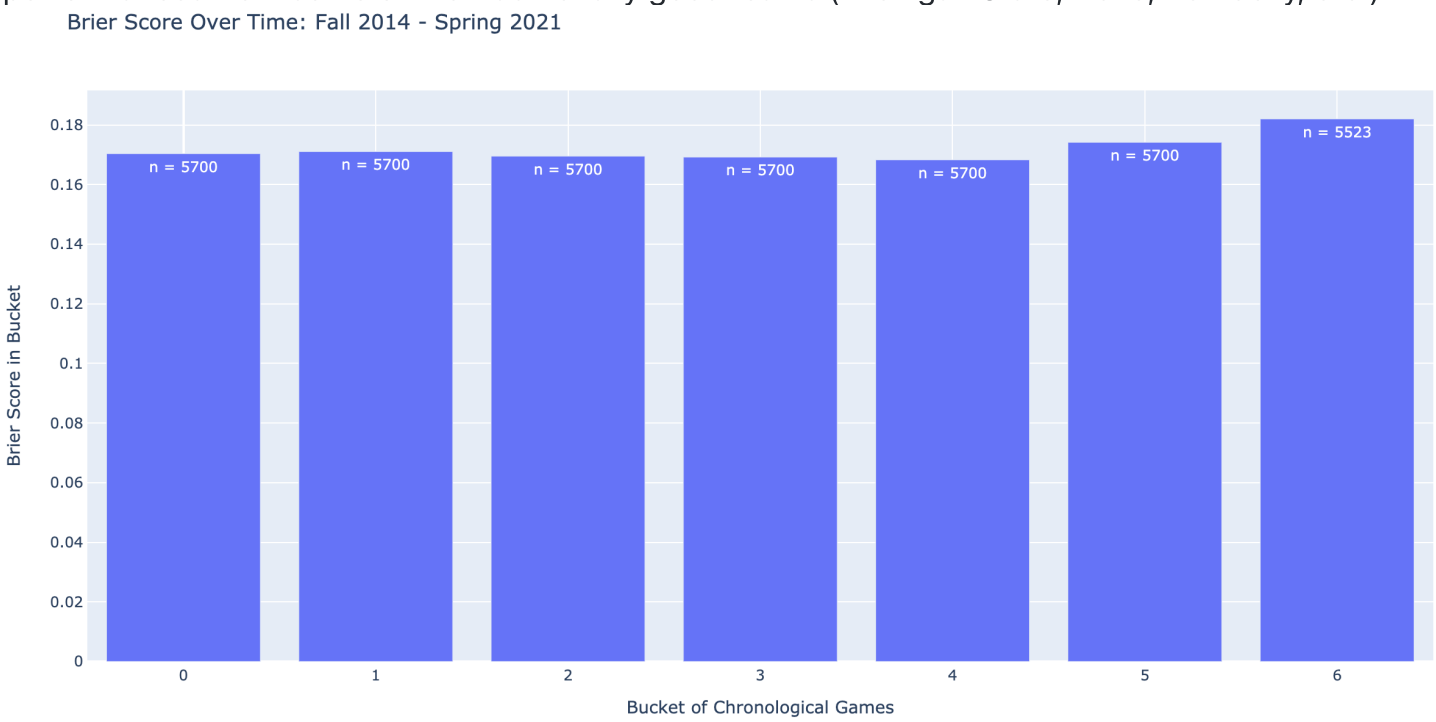
The primary variables subject to tuning were the K-factor, Home-Court Advantage, and New Season Carryover. Secondary variables subject to tuning were the MoV multiplier equation, and the starting Elo rating for new teams joining the system. The loss function used for tuning was the Brier score. Essentially, this is the sum of the squared difference between predicted win probability and result for every game in the dataset. The result is 1 for a win and 0 for a loss. So if the model gives a team a 70% win probability and they win, the Brier score is `(1 - 0.70)^2 = 0.09`. However, if they lose, the score is 0.49. The goal in the tuning process was to minimize the total Brier score. In every simulation, I don't start tracking Brier until after the 4th season to give the model some time to instantiate.

Throughout tuning, I narrowed in on optimal values of **47**, **+83**, and **1.0** for the primary variables: K-factor, Home-Court Advantage, and New Season Carryover. All three of these were surprising in their own way. A K-factor of 47 is much larger than the NBA K-factor of 20. This implies that momentum plays a bigger role in college basketball than in the NBA. A larger K-factor encourages larger swings in Elo rating after every game. A home-court advantage of +83 implies only a 3.25-point advantage at home - which is less than what the data indicates. In the last 10 years, the average NCAA home advantage was 7.1 points, while the NBA was 2.8. Perhaps this number is inflated due to a long right tail of dominant home victories by D1 teams that tend to host more blowout games instead of traveling to blow someone out. For example, Harvard (mid-to-bottom-tier D1) always hosts MIT (mid-tier D3) as part of a friendly rivalry where Harvard always wins by a million. Games like these are outliers that skew the 7.1-point advantage up while the Elo model discovers that for the *majority* of games +83 is appropriate. Lastly, a season carryover of 1.0 is very surprising. This implies that every team should pick right back up where they left off at the end of the last season. Obviously, in reality this is not true. However, in tuning the model, 1.0 worked best. One possible explanation is that teams tend to be able to recruit the same caliber players that they currently have year-over-year. Yes, Virginia loses some stars to the NBA or graduation, but they have equal caliber players coming in next fall to take their spot. 2021 might not be a good example because blue-bloods like Michigan State, Duke, and Kentucky really struggled, but generally, good teams stay good.

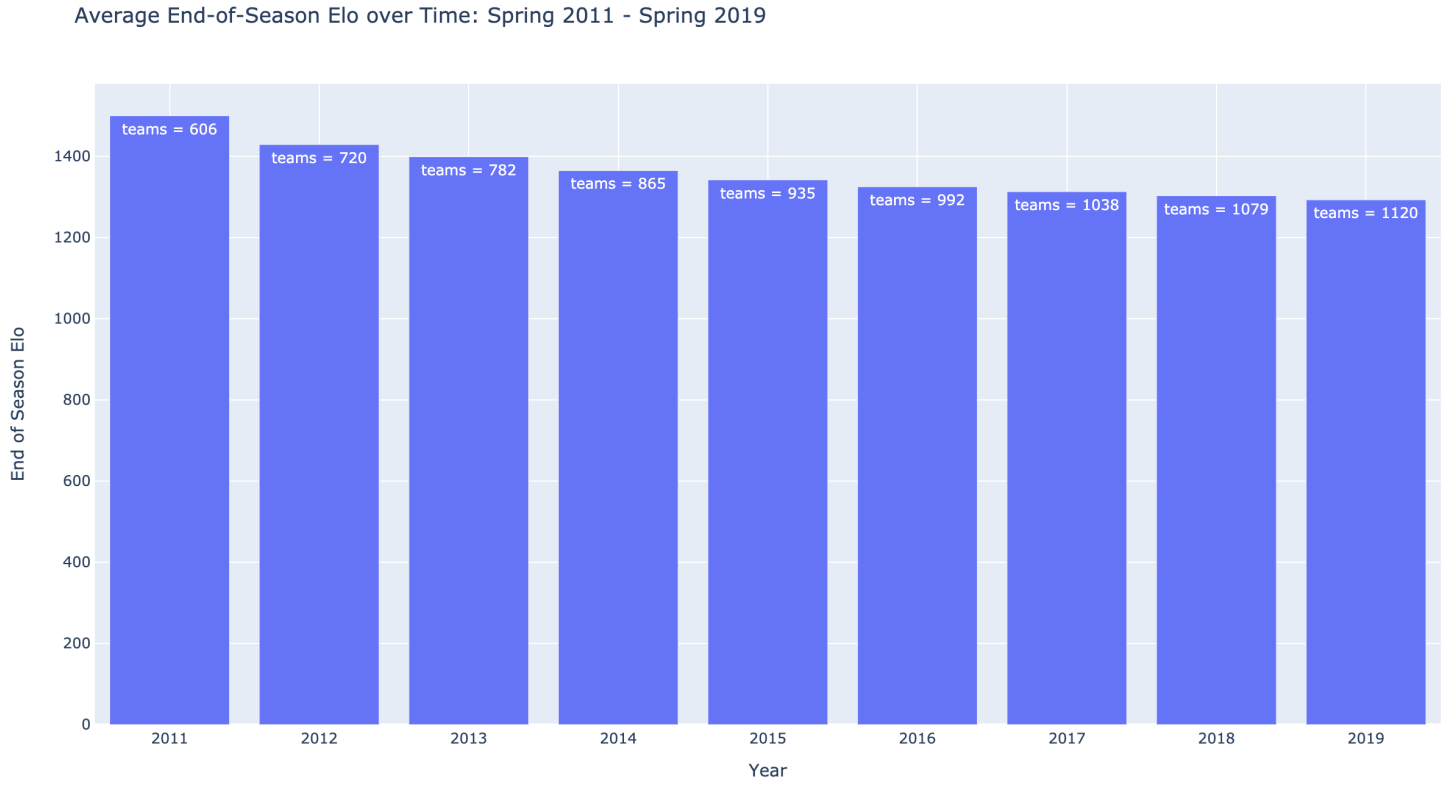
Secondary tuning of the starting elo for new teams narrowed in on **1050**. Lastly, I made minor adjustments to FiveThirtyEight's MoV multiplier equation to reflect the fact that MoV in college games is usually more than in NBA games. After some tinkering, I landed on the MoV multiplier equation explained above that essentially dampens the impact of MoV a little more than the NBA equation:



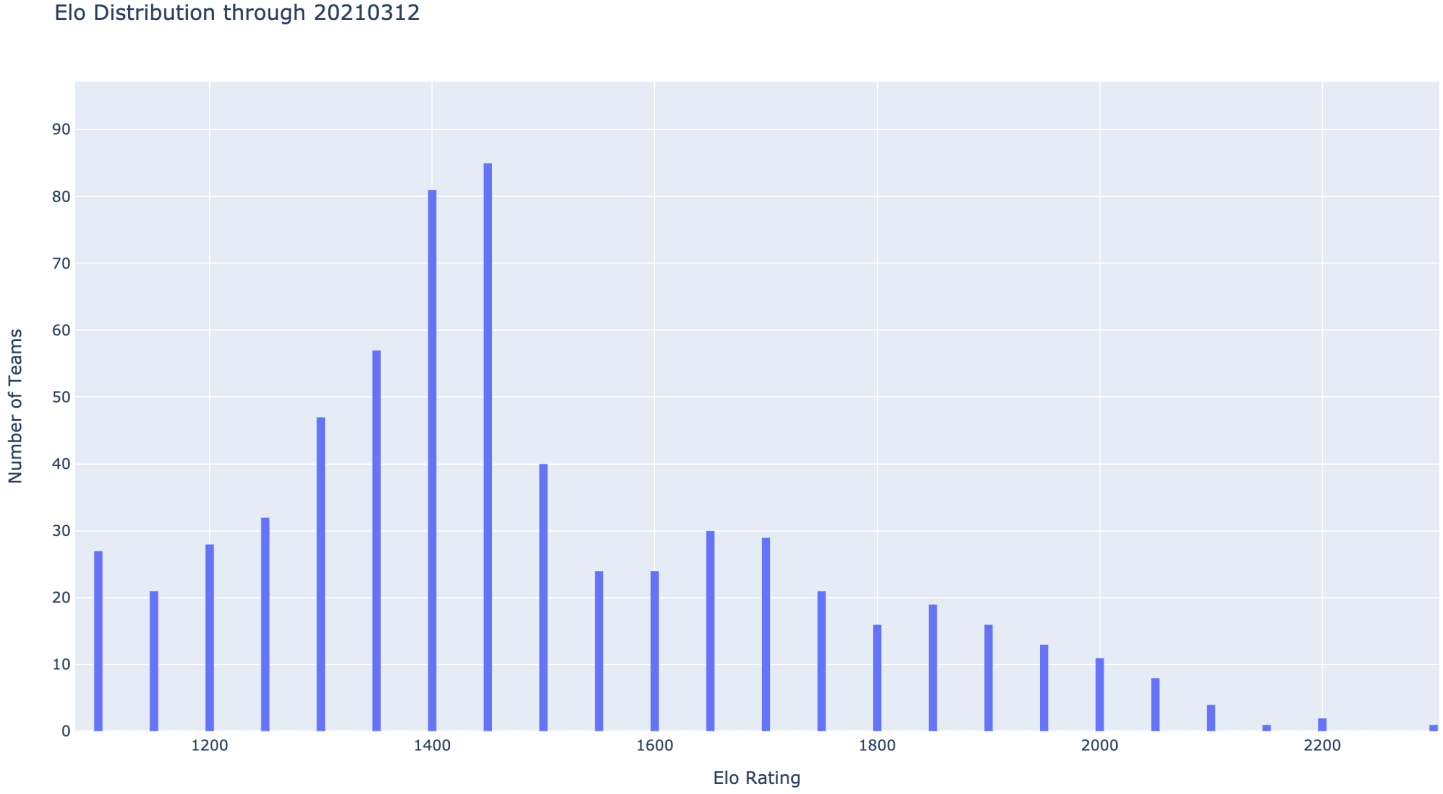
The plot below shows the average Brier score, the error function to minimize in tuning, per season over the last 7 years. 2021 is not quite complete yet, but the majority of games are complete. It is a good sign that the error remains relatively stable over time. This slight uptick in error in this latest season is something to keep an eye on. Hopefully this does not become a trend of increasing error over time. The 2020-2021 season has been a bit abnormal, so let's hope the increased error is due to Covid-related factors (season disruptions, modified schedules, no fans, etc.) and anamolous poor performances from some of the traditionally good teams (Michigan State, Duke, Kentucky, etc.).



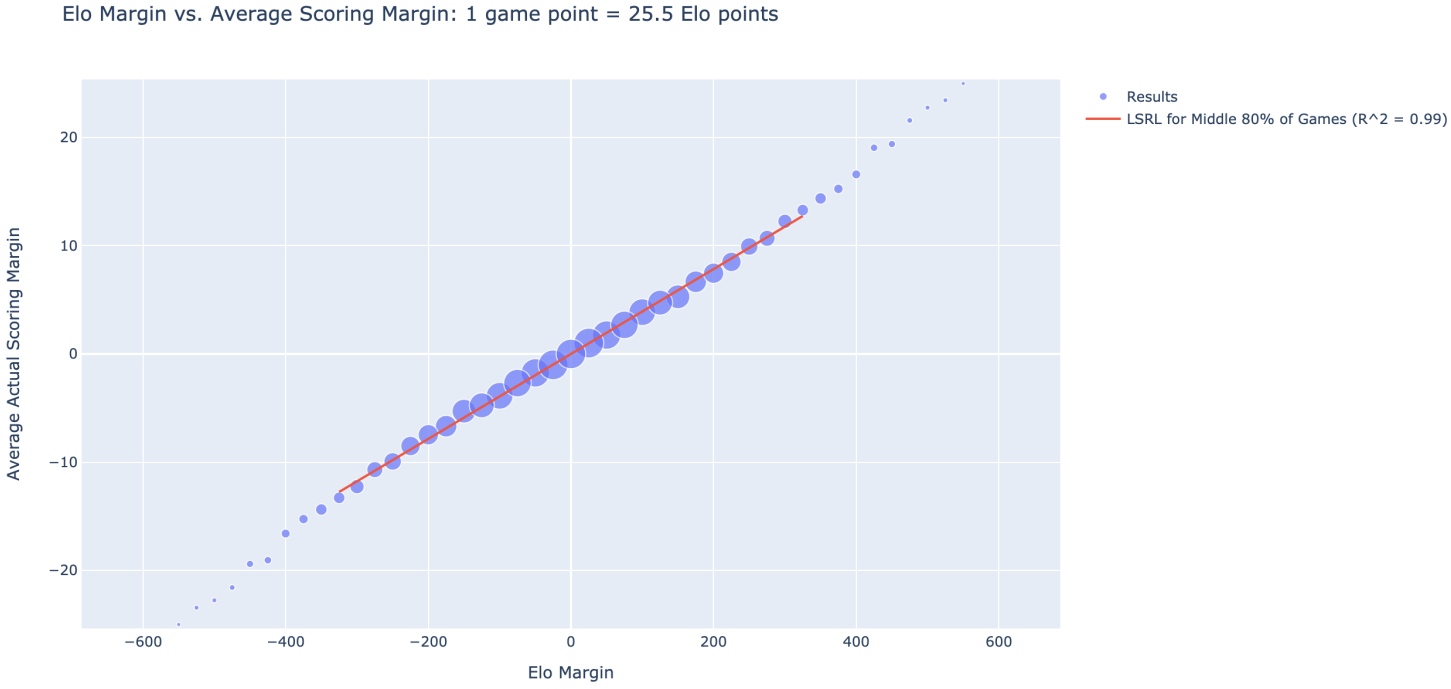
As mentioned in the methodology section, Elo ratings are meant to be a closed system. Ideally, the total number of points in the system does not change. However, so many new teams join the system that the total number of points is not stable over time. The trend of decreasing average Elo rating over time, shown below, is not surprising since so many teams enter the Elo rating system with their default 1050 rating and play just one game (i.e. the MITs of the NCAA world who enter the system just to play one game against Harvard every year bring the overall rating average down). This is a trend to keep an eye on over time. Perhaps, it makes sense to add in some cleaning at the end of every season that removes teams who only played one game.



The chart below shows the Elo rating distribution as of games through 3/12/2021. There are a lot of teams not pictured who are clustered around the 1050 mark because they joined the system late and don't play enough to distribute themselves. This chart shows a distribution centered close to 1500, which is what we should expect for well-sorted teams. 1500 is supposed to be the average. As expected, some teams really distinguish themselves in the long right tail (Gonzaga in this snapshot).



In order to predict point spreads from Elo ratings, we need to determined how many points of Elo difference equal 1 point of final score difference. To do this, I plotted Elo rating differences versus the average resulting score difference in those games. The size of the bubbles is proportional to the number of games in the dataset had that Elo difference. It's a good sign that a clear trend emerges in this plot. That indicates we can actually convert Elo point differences to point spreads. Towards the left and right tails of the plot, you can see some curvature. But, for the majority of games, a straight line can help explain the relationship. I fit a linear regression to the middle 80% of the data. In general, the slope of this line indicates that every 25.5 points of Elo difference corresponds to a 1 point difference in predicted point spread. In other words, a team that is 50 Elo points better should be favored by about 2 points in the game.



Lastly, to visualize the accuracy of the Elo model's predictions, I plotted predicted win probability versus actual win percentage. For example, there were 855 games in the dataset where the model predicted a team's win probability was ~53% (rounded to the nearest 1%). In reality, 450 of those teams won their games. That is a win percentage of 52.6% `(450/855)` - which means the model's prediction was about right. Over time, teams given an X% chance of winning should win about X% of their games. This plot shows that relationship to be mostly true. Plotted with a linear line of best fit, the R-Squared value is ~0.99.