March Madness Predictions

Jackson Gasperack & Sawan Pandita

2025-12-06

Table of contents

1 Introduction

1.1 Motivation

Every year, millions of people fill out their bracket in hopes of being the person that correctly predicts the winner of that year's Men's Basketball National Championship. However, it is nearly impossible to do this, the odds for predicting the first round correctly is 1 to 3500 on its own. The odds for filling out an entire perfect bracket are 1 to 2.9 quintillion assuming each team has a 50/50 chance of winning or losing. That's not the case however, as statistics, seedings, and rosters influence how people choose which team will win each game. But, even with more robust models, like FiveThirtyEight's mentioned in their article, the odds are still a staggering 1 in 2 billion (Paine & Voice, 2017). It is so unpredictable to pick a single winner that we even had to modify our research question since our models began to underfit. To make this unpredictable choice a little more clear we wanted to create a model that could predict which teams would be most likely to make the Final Four.

1.2 Dataset Description

We imported Lundberg's dataset, College Basketball Dataset, from Kaggle. This notebook is comprised of 13 csv files, one for each year from 2013-2025 March Madness, and one main file that ranges all the years. In each separate year file, the variables comprised of the following for the season that lead up to that postseason:

- **TEAM**: School name
- **CONF**: Division I Conference school is apart of
- **G**: Games played
- **W**: Games won
- **ADJOE**: Adjusted Offensive Efficiency (Pts/100 possessions)
- **ADJDE**: Adjusted Defensive Efficiency (Pts allowed/100 possessions)
_ **BARTHAG**: Power rating (Chance of beating D1 team)
- **EFG_O**: Effective FG percentage
- **EFG_D**: Effective FG percentage allowed
- **TOR**: Turnover rate
- **TORD**: Turnover rate allowed (Steal rate)
- **ORB**: Offensive rebound rate
- **DRB**: Offensive rebound rate allowed
- **FTR**: Free throw percentage
- **FTRD**: Free throw percentage allowed
- **2P_O**: Two-point FG percentage
- **2P_D**: Two-point FG percentage allowed
- **3P_O**: Three-point FG percentage
- **3P_D**: Three-point FG percentage allowed
- **ADJ_T**: Adjusted tempo (possessions/40 mins)
- **WAB**: Wins above bubble (Wins after the March Madness qualification cutoff)
- **POSTSEASON**: Postseason round of elimination
- **SEED**: Seed in March Madness

In the 2025 dataset, 3PR and 3PRD were added, they weren't specified in the data card but I assumed it was just 3P_O and 3P_D with different names that the creator accidentally added. RK in the 2020 dataset is the ranking of the teams from where the data was originally scraped, since there was not a postseason in 2020 due to COVID. Finally, in the large dataset the year of each row was also added since it's the accumulation of all the separate year datasets.

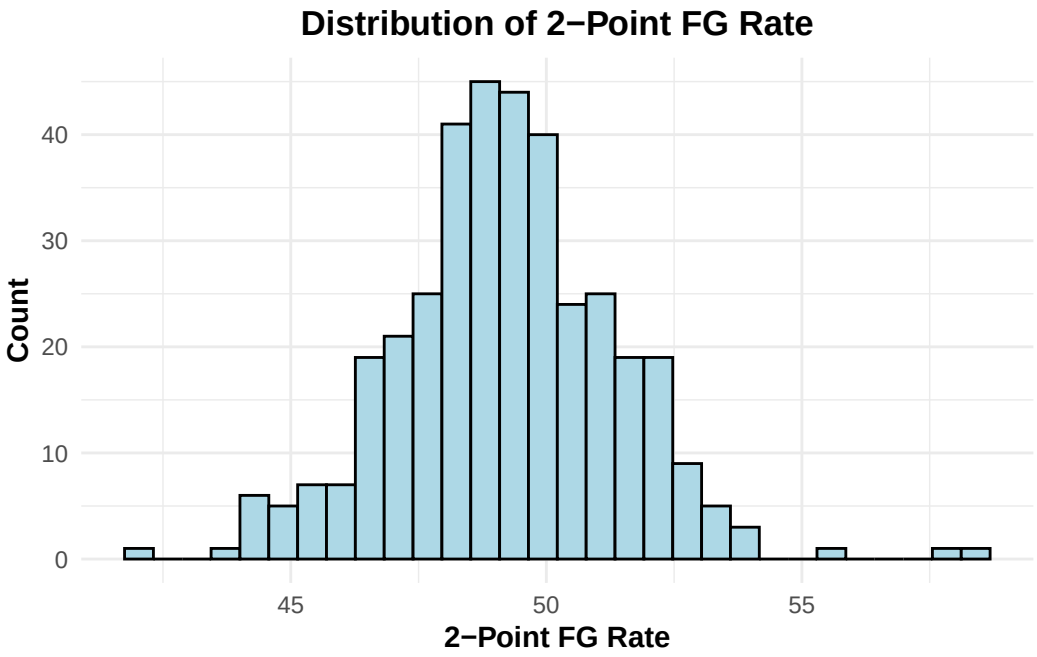
2 Data Analysis

2.1 Data Cleaning

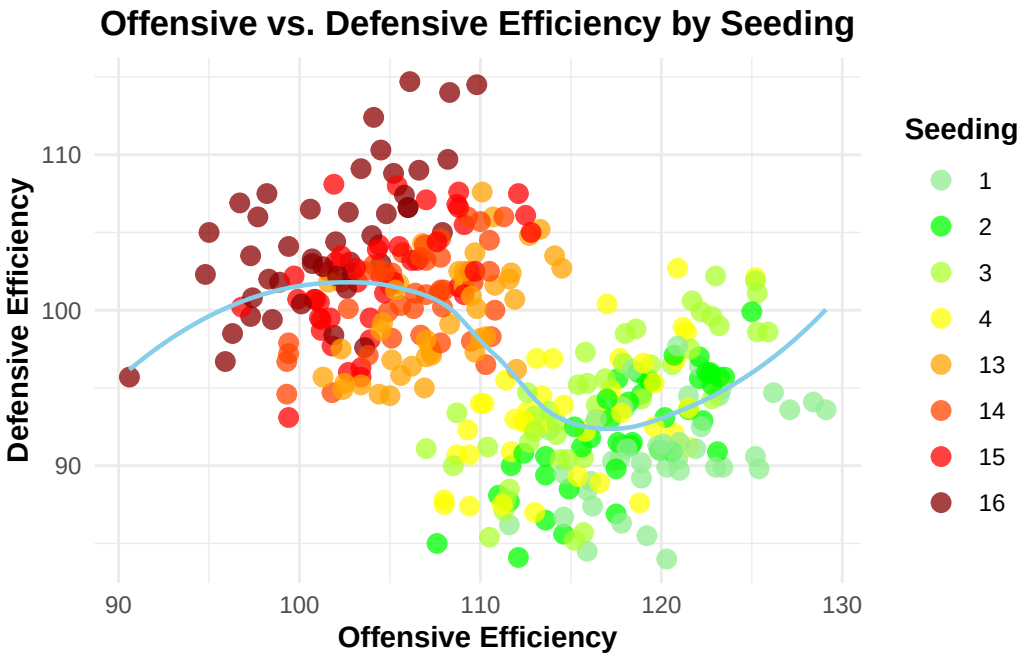
After we loaded our dataset into the R environment, there were several steps that we needed to take before constructing our models. The first thing we did was make sure all types of all tables were corresponding to each other, for example the seed columns of some datasets were character types instead of numeric. Then we printed the structure with the function str() to see what type of variables we were dealing with. After getting a good idea of the data we were dealing with we then wanted to check for how many unique values were in each column to establish categories. We found that there were no null values in terms of statistics, which in our case meant we didn't need to worry about them because we'd be using statistics to predict our outcome. When looking through the 13 datasets, we wanted to make sure all variables were consistent. Since 2020 didn't have a postseason because of COVID and this dataset was made before the 2025 postseason, those two datasets should have one less variable than the rest. We also renamed some variables and removed other unnecessary ones. The 2025 dataset had a rank column and a seed column which seemed redundant, it also had two additional statistics included that weren't in any other dataset, so we removed those. After this process we moved onto finding relationships between our variables.

2.2 Distributions and Correlations

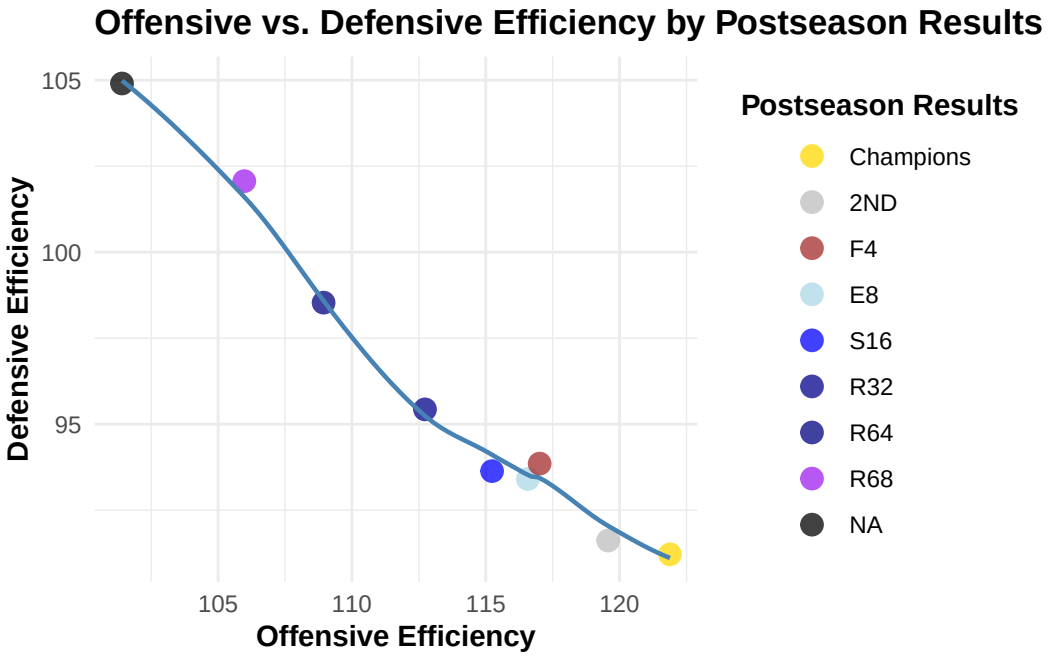
To start, we separated our offensive and defensive statistics in order to view their distributions separately. We then made a list corresponding these statistics' variable names to their nicer formatted names. After finding the average of the statistics of all teams since 2013, we were finally ready to plot our distributions. When finishing this process for both sides of the ball, we found that all of our variables were normally distributed for the most part. As an example, here's the distribution of the two-point FG rate variable.



After finding distributions we wanted to see how our variables interacted with each other. With our research question looking to predict which teams will make the final four, we wanted to see how seedings and postseason results would correlate with offensive and defensive efficiency metrics. After making a table of the offensive and defensive efficiencies for each seeded team in a given year, we wanted to check the counts to ensure consistency. When doing this however, we were met with an error as some seeds had too many and some had not enough since there should be four of each. We realized that this was because of the play-in teams where eight teams of low seedings play in games before the actual tournament in order to qualify. So with some research, we found which teams lost in the play-in games and set their seed as -1. We also found that some teams were just incorrectly seeded on accident so we fixed those as well. After completing that we plotted how efficiency metrics are affected by the seeding of the team, which shows that lower seeds are better defensively whereas higher seeds are better offensively.



This makes sense as basketball has turned very offensive in the last decade. In the 2010's, offensive production was 50% more important towards winning games than defensive production (Wilco, 2018). With the game, in general, only becoming more centered around three-point shooting we can assume this trend is either the same or has only increased more since 2018. We then took every other statistic and found their correlation to adjusted efficiency for offense and defense. The highest indicators were effective field goal rate and two-point rate. Finally, we grouped all teams by their postseason result and averaged the offensive and defensive efficiencies for each seed across all years. This showed the same trend as the last plot. The early round exits are better defensive performers but the final eight teams usually have better offensive performance.



3 Feature Selection

In order to choose the best set of features, we ran three different subset selection algorithms. First, we created a full and null model with isFinalFour as the target. Then, we used the step function to do forward selection going from the null to the full model. Secondly, we used another step function to do backward selection going from the full to the null model. Finally, we used the step function a third time to do stepwise selection which goes both ways. After printing their summaries, the forward selection model performed the best with an AIC of 209.49.

After seeing the p-values, we decided to remove BARTHAG since it didn't seem to contribute much. We did a plug-in method afterwards in order to add more features to reduce overfitting while also maintaining a relatively low AIC. The model we came up with actually ended up being better with an AIC of 209.16. So, the feature set that gave us the best AIC is:

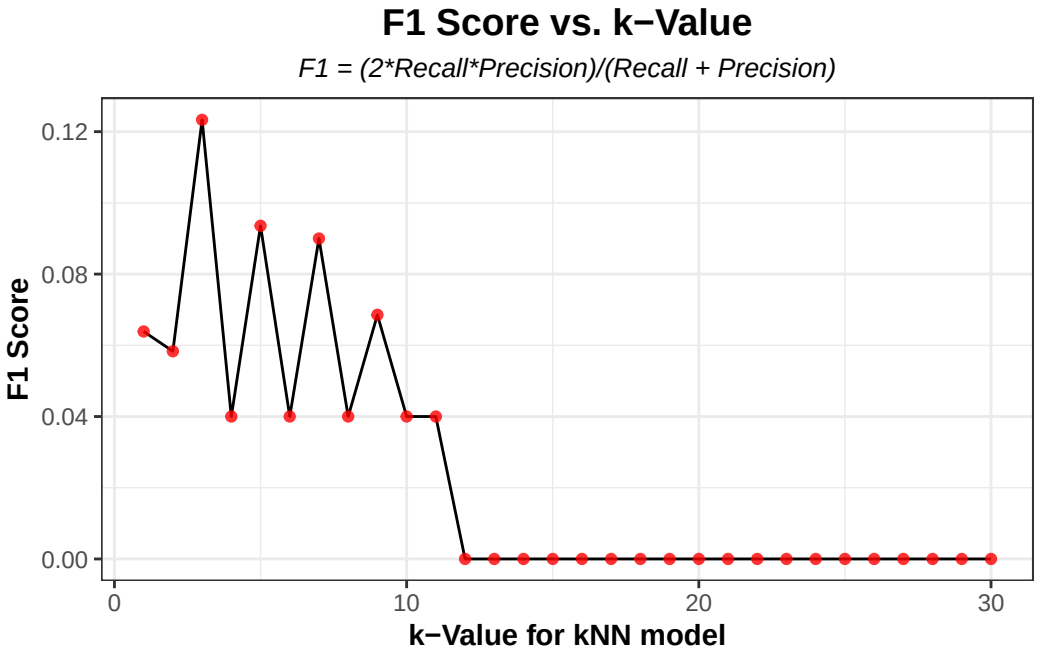
- **Wins**
- **Adjusted Offensive Efficiency**
- **Adjusted Defensive Efficiency**
- **Two-point rate**
- **Two-point rate allowed**
- **Turnover rate**
- **Steal rate**
- **Free Throw rate**

This Logistic Regression model is not included in any model evaluations as it performed worse than our baseline model which you will later see also performed poorly.

4 Model Construction

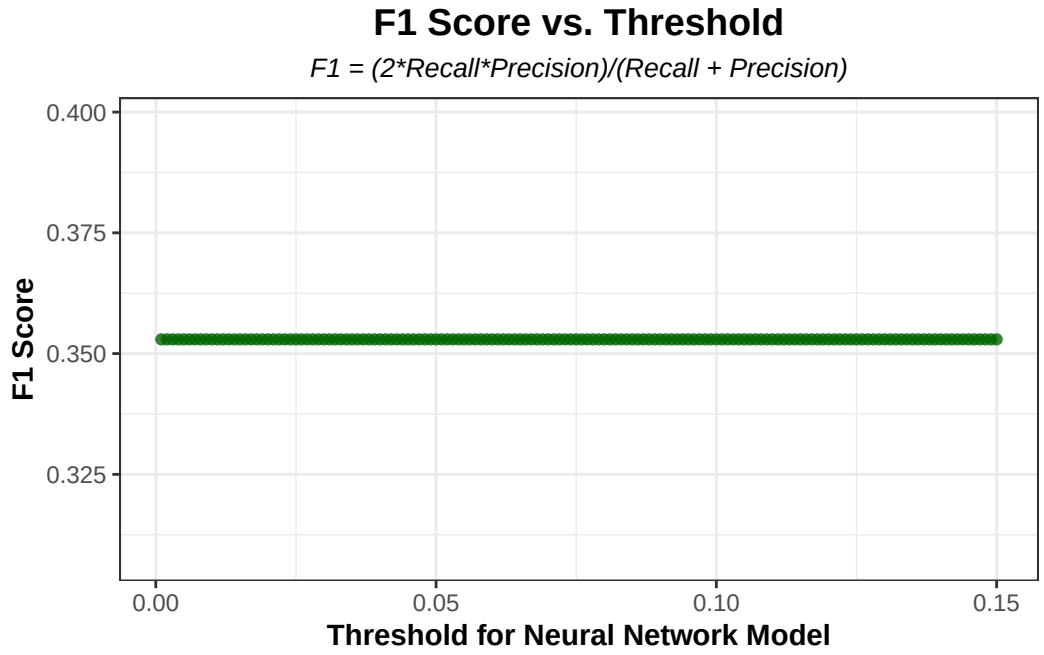
4.1 k-Nearest Neighbors

For this model, I wanted to do two forms of resampling since it usually underperforms when compared to models like Neural Network and Random Forest. In the outer loop I did a 30-fold Cross Validation technique to compute the best possible F1 score. We focused on F1 score instead of accuracy since our success rate is a very small 1.1% which means accuracy will always be high and therefore can be misleading. Inside that loop we also did a 10-fold cross validation on our training and testing sets for each value of k, since kNN is not as computationally expensive as any of our other models. Through this rigorous approach we found that $k = 3$ maximized the F1 score of the model at around 0.123. Even though it had the highest accuracy with 99.2%, this model could only identify the positive class slightly more than 12% of the time.



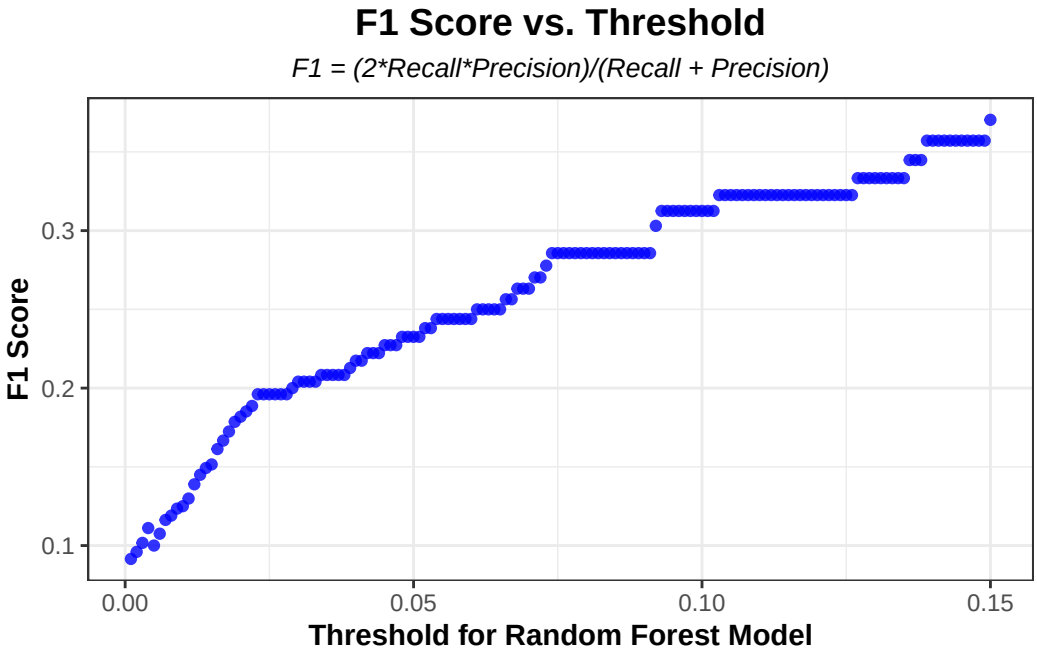
4.2 Neural Network

For the neural network we didn't use any other resampling techniques than k-fold cross validation to find the threshold that produced the best F1 value. This is because they have many activation functions in a specified amount of hidden layers as opposed to just finding the k-Nearest x-values like the previous model did. In this case we fit a model of size 5 with 1000 iterations and entropy to predict the best possible Final Four combination. After going through thresholds from 0.001 to 0.15 we found the threshold of around 0.13 to maximize F1 with a score of 0.35 which is considered good in a scenario like this when the predicted class is so uncertain. This model still posted a great accuracy of 97.4% and identified the positive class three times better than kNN. It's also worth noting that the neural network model does have the possibility to outperform both Random Forest and XGBoost. Yet, not only are those performances inconsistent but it will mostly also have a worse F1 score than both.



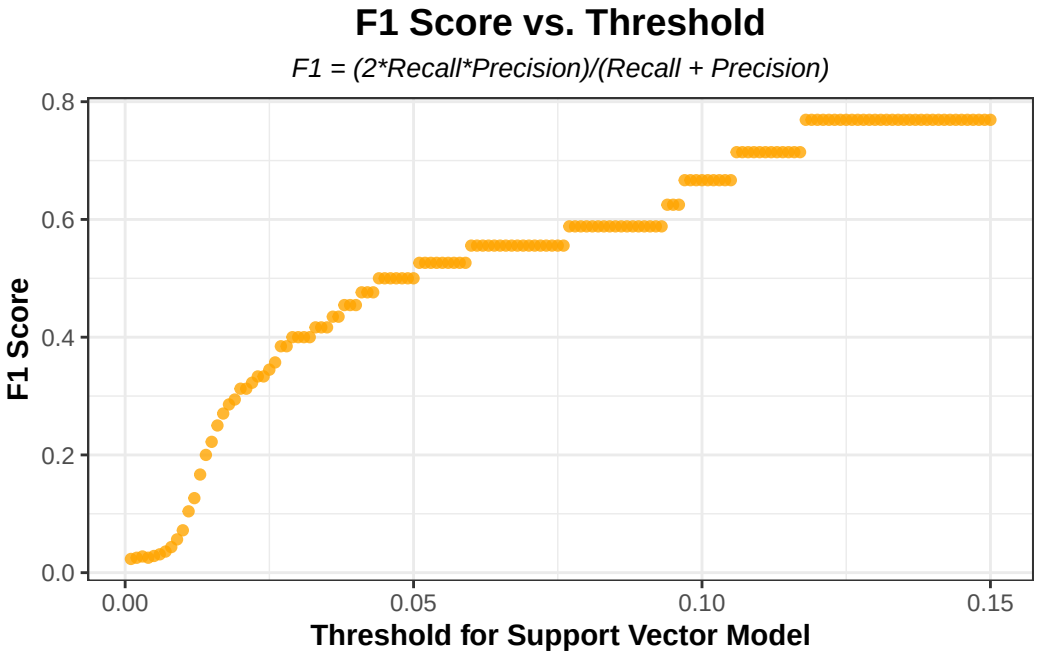
4.3 Random Forest

For the random forest model we did the same technique that we did for the neural network. Due to the complexity of the model, we only had a for loop go through the previously stated thresholds. We fit a random forest model of 5000 trees which had a 1% OOB estimate of error rate. After going through these thresholds we found that a threshold of around 0.15 maximized the F1 score for this model. We were able to achieve a F1 score of 0.37 along with an accuracy of around 97% which improves on the previous model's F1 score while maintaining the high accuracy. We also saw that the F1 score kept increasing past 0.15 so we tried to increase the threshold up to 0.3. However when we did that we realized that even though F1 scores kept increasing, less and less teams were counted as successes which led to limited findings. So, we decided to cap it at 0.15 in order to interpret our model's predictions more accurately.



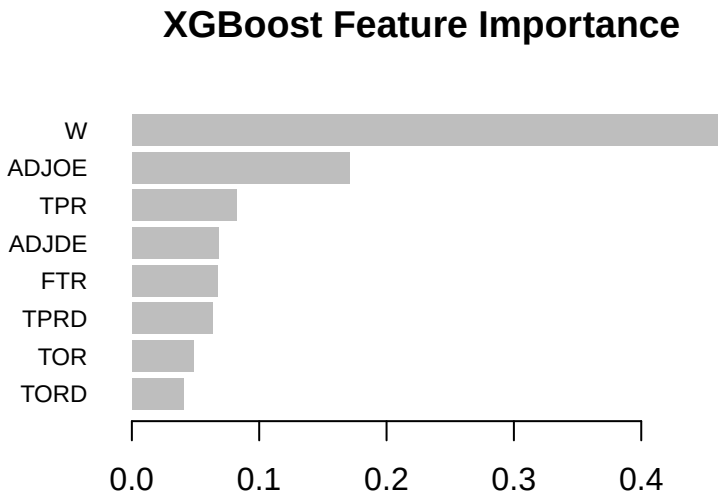
4.4 SVM

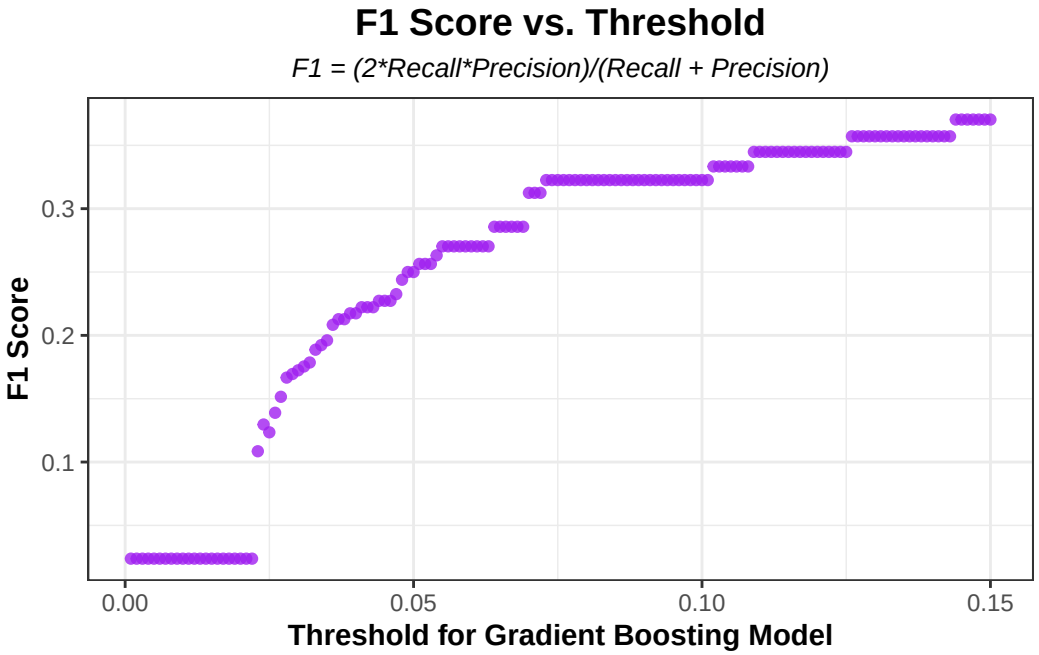
For the Support Vector Machine, I utilized the e1071 package to create a non-linear classification model. I specifically chose the radial basis function (RBF) kernel because the relationship between basketball statistics and winning is rarely linear. Similar to the other models, I enabled probability generation to allow for ROC curve analysis later. After training the model on our selected features, I tuned the decision threshold to maximize the F1 score. Interestingly, the SVM model produced the highest F1 score of the entire project at 0.769 with an accuracy of 96.9%. While these numbers look incredible on paper, the Area Under the Curve (AUC) was 0.772, which is significantly lower than the tree-based models (Random Forest and XGBoost). This discrepancy suggests that while the SVM found a very specific decision boundary that worked well for the test set's positive cases, it might not be as robust globally as the gradient boosting methods.



4.5 XGBoost

I implemented XGBoost (Extreme Gradient Boosting) as a bonus method because it is widely considered the industry standard for tabular classification problems. Unlike Random Forest, which builds trees independently, XGBoost builds trees sequentially to correct the errors of previous trees. I performed a grid search to tune the hyperparameters, resulting in a max_depth of 3, an eta (learning rate) of 0.1, and a subsample of 0.6. Keeping the tree depth shallow (3) helped prevent overfitting, which is a common risk with this dataset given the class imbalance. The model achieved the highest AUC of all our models at 0.958 and a strong F1 score of 0.4. As seen in the Feature Importance plot, the model heavily favored Wins (W) and Adjusted Offensive Efficiency (ADJOE), confirming that in modern college basketball, elite offense is a stronger predictor of deep tournament runs than defense.



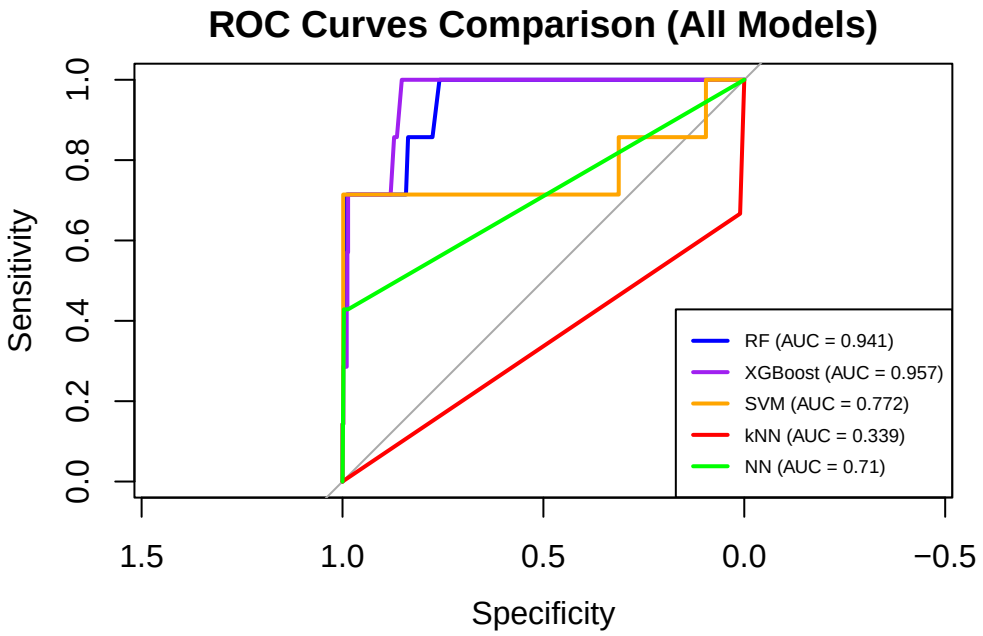


5 Results

To definitively compare our five models, we plotted their Receiver Operating Characteristic (ROC) curves on a single graph. The ROC curve illustrates how well the model separates positive classes (Final Four teams) from negative classes.

Table 1: Metrics Computed for Each Model

F1 Score	Accuracy	Area Under Curve	Balanced Accuracy	Model
0.370	0.971	0.941	0.844	Random Forest
0.370	0.971	0.957	0.844	Gradient Boosting
0.769	0.995	0.772	0.856	Support Vector Machine
0.353	0.981	0.710	0.708	Neural Network
0.123	0.992	0.339	0.500	k-Nearest Neighbors

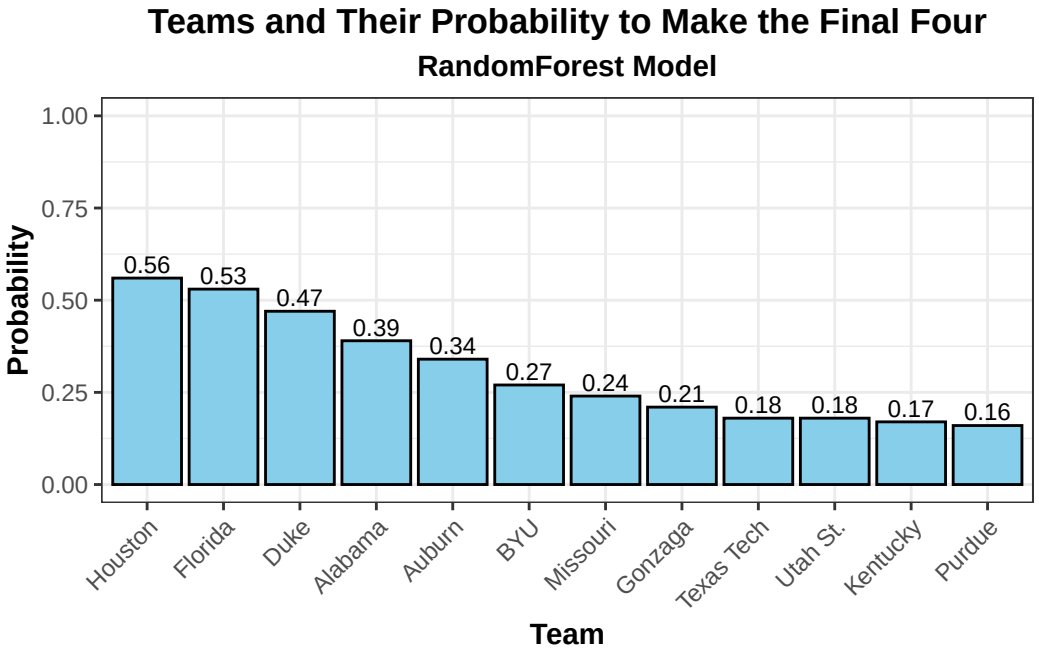


- **XGBoost (Purple Line)**: Performed the best with an AUC of *0.96*, hugging the top-left corner the tightest.
- **Random Forest (Blue Line)**: A close second with an AUC of *0.94*.
- **Neural Network (Green Line)**: Performed well with an AUC of *0.71*.
- **SVM (Orange Line)**: Showed a "blocky" step function with an AUC of *0.77*, indicating less granular probability estimates.
- **kNN (Red Line)**: Performed poorly, falling below the diagonal line with an AUC of *0.34* which performs worse than random guessing of 0.5.

While SVM had the highest raw F1, we believe XGBoost is the superior model due to its high AUC and consistent probabilities.

6 Conclusion

6.1 RandomForest 2025 Predicitions



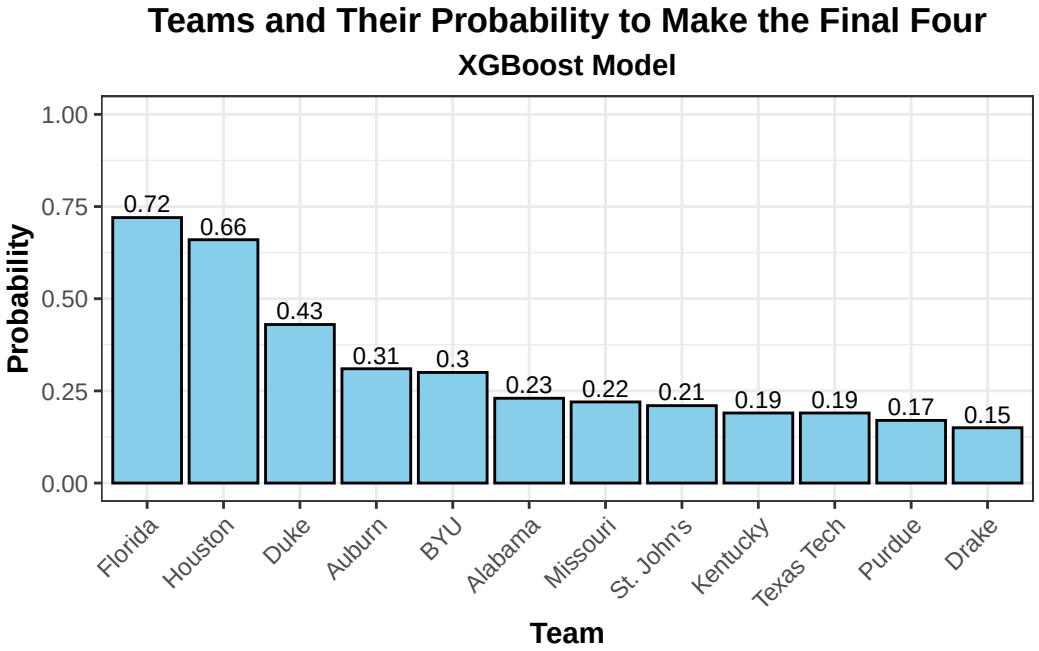
My RandomForest model accurately predicted three out of the four real 2025 final four teams. The fourth actual team that did not have a top four probability was actually ranked fifth by my model, meaning that even if it's an edge case my model can accurately predict elite contenders. The other teams predicted by my model that didn't end up making the final four ended up with the following eliminations:

- Alabama **DEF** by Duke *[Elite Eight]*
- Texas Tech **DEF** by Florida *[Elite Eight]*
- Purdue **DEF** by Houston *[Sweet 16]*
- BYU **DEF** by Alabama *[Sweet 16]*
- Kentucky **DEF** by #2 Tennessee *[Sweet 16]*
- Gonzaga **DEF** by Houston *[Round 2 by 5 points]*

So looking more carefully at the edge cases, my model correctly predicted 6 of 8 elite eight teams. It also predicted teams that were clearly powerhouses like Gonzaga, Kentucky, and BYU who just got an unlucky schedule and were beaten by another predicted team earlier in the tournament. Unfortunately, my model is not susceptible to upsets as there were three highly favored teams that lost in the first or second round.

- Missouri **DEF** by Drake *[Round 1]* **UPSET**
- Utah State **DEF** by UCLA *[Round 1]* **UPSET**
- St. John's **DEF** by Arkansas *[Round 2]* **UPSET**

6.2 XGBoost 2025 Predicitions



My XGBoost model actually predicted the exact same top four teams as Jackson's Random Forest model: Houston, Florida, Duke, and Alabama. However, my model was correct and much more confident in its top pick, giving Florida a 72% probability and Houston came in second with a 66% probability of making the Final Four compared to the Random Forest's 56%. Auburn was also the fifth most likely team in my model, meaning both models were good at successfully determining elite contenders. Just like Jackson's model, mine struggled with upsets, as Missouri and St. John's were included in this model as well despite losing in the first two rounds. My model also identified new potential Final Four contenders in Arizona and Drake, both of whom upset highly favored teams Missouri and Oregon, respectively. With Arizona also making the Sweet 16 this just goes on to show how this model is good at predicting top-tier competitors. Since the predictions were so similar to the Random Forest model, it confirms that these teams were the statistical favorites,

even if the actual tournament results were different due to upsets or the unpredictability of the tournament itself.

6.3 Dataset Conclusions

After fully completing model construction and analysis, we found that XGBoost was slightly more accurate in its 2025 predictions and therefore was the best model overall. From both the Random-Forest and XGBoost variable importance plots we can see that wins was easily the most dominant variable and the best way to see if a team would make the final four or not. We can also say efficiency metrics like ADJOE and ADJDE were dominant predictors of postseason success. Whereas, defensive statistics like TPRD and TORD were not as important when trying to predict teams probability of making the final four.

7 Appendix

7.1 Datasets

Sundberg, A. (2025, March 20). College Basketball Dataset. Kaggle. https://www.kaggle.com/datasets/andrewsundberg/college-basketball-dataset

7.2 Sources

NCAA. (2025, April 15). Browse every NCAA bracket since 1939 with stats and records. NCAA. https://www.ncaa.com/basketball-men/d1/every-ncaa-bracket-1939-today-tournament-stats-records

Paine, N., & Voice, J. (2017, March 14). The odds you'll fill out a perfect bracket. FiveThirtyEight. https://fivethirtyeight.com/features/the-odds-youll-fill-out-a-perfect-bracket/

Wilco, D. (2018, November 11). Nine Years of College Basketball Data Show Offense Matters More in March. NCAA. https://www.ncaa.com/news/basketball-men/article/2018-11-10/nine years-college-basketball-data-show-offense-matters-more

7.3 R Packages

----- tidyverse -----
Welcome to the {tidyverse}.
Hadley Wickham, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, Alex Hayes, Lionel Henry, Jim Hester, Max Kuhn, Thomas Lin Pedersen, Evan Miller, Stephan Milton Bache, Kirill Müller, Jeroen Ooms, David Robinson, Dana Paige Seidel, Vitalie Spinu, Kohske Takahashi, Davis Vaughan, Claus Wilke, Kara Woo, Hiroaki Yutani.
2019.
Journal of Open Source Software.
4.
43.
1686.
10.21105/joss.01686.

----- ggplot2 -----
Hadley Wickham.
ggplot2: Elegant Graphics for Data Analysis.
Springer-Verlag New York.
2016.
978-3-319-24277-4.
https://ggplot2.tidyverse.org.

----- caret -----
Building Predictive Models in R Using the caret Package.
28.
https://www.jstatsoft.org/index.php/jss/article/view/v028i05.
10.18637/jss.v028.i05.
5.
Journal of Statistical Software.
Kuhn, Max.
2008.
1-26.

----- glmnet -----
Regularization Paths for Generalized Linear Models via Coordinate Descent.
Jerome Friedman, Trevor Hastie, Robert Tibshirani.
Journal of Statistical Software.
2010.
33.
1.
1--22.
10.18637/jss.v033.i01.

----- randomForest -----
Classification and Regression by randomForest.
Andy Liaw, Matthew Wiener.
R News.
2002.
2.
3.
18-22.
https://CRAN.R-project.org/doc/Rnews/.

----- FNN -----
FNN: Fast Nearest Neighbor Search Algorithms and Applications.
Alina Beygelzimer [aut] (cover tree library), Sham Kakadet [aut] (cover tree library), John Langford [aut] (cover tree library), Sunil Arya [aut] (ANN library 1.1.2 for the kd-tree approach), David Mount [aut] (ANN library 1.1.2 for the kd-tree approach), Shengqiao Li <lishengqiao@yahoo.com> [aut, cre].
2024.
R package version 1.1.4.1.
https://CRAN.R-project.org/package=FNN.
10.32614/CRAN.package.FNN.

----- pROC -----
pROC: an open-source package for R and S+ to analyze and compare ROC curves.
Xavier Robin, Natacha Turck, Alexandre Hainard, Natalia Tiberti, Frédérique Lisacek, Jean-Charles Sanchez, Markus Müller.
2011.
BMC Bioinformatics.
12.
77.

----- nnet -----
Modern Applied Statistics with S.
W. N. Venables, B. D. Ripley.
Springer.
Fourth.
New York.
2002.
ISBN 0-387-95457-0.
https://www.stats.ox.ac.uk/pub/MASS4/.

----- PRROC -----
Area under Precision-Recall Curves for Weighted and Unweighted Data.

Jens Keilwagen, Ivo Grosse, Jan Grau.
2014.
9.
3.
PLOS ONE.

----- xgboost -----
xgboost: Extreme Gradient Boosting.
Tianqi Chen <tianqi.tchen@gmail.com> [aut], Tong He <hetong007@gmail.com>
[aut], Michael Benesty <michael@benesty.fr> [aut], Vadim Khotilovich
<khotilovich@gmail.com> [aut], Yuan Tang <terrytangyuan@gmail.com>
[aut] (ORCID: <https://orcid.org/0000-0001-5243-233X>), Hyunsu Cho
<chohyu01@cs.washington.edu> [aut], Kailong Chen [aut], Rory Mitchell [aut],
Ignacio Cano [aut], Tianyi Zhou [aut], Mu Li [aut], Junyuan Xie [aut], Min Lin
[aut], Yifeng Geng [aut], Yutian Li [aut], Jiaming Yuan <jm.yuan@outlook.com>
[aut, cre].
2025.
R package version 1.7.11.1.
https://CRAN.R-project.org/package=xgboost.
10.32614/CRAN.package.xgboost.

----- Matrix -----
Matrix: Sparse and Dense Matrix Classes and Methods.
Douglas Bates [aut] (ORCID: <https://orcid.org/0000-0001-8316-9503>),
Martin Maechler <mmaechler+Matrix@gmail.com> [aut, cre] (ORCID:
<https://orcid.org/0000-0002-8685-9910>), Mikael Jagan [aut] (ORCID:
<https://orcid.org/0000-0002-3542-2938>).
2025.
R package version 1.7-4.
https://CRAN.R-project.org/package=Matrix.
10.32614/CRAN.package.Matrix.

----- e1071 -----
e1071: Misc Functions of the Department of Statistics, Probability Theory Group
(Formerly: E1071), TU Wien.
David Meyer <David.Meyer@R-project.org> [aut, cre] (ORCID:
<https://orcid.org/0000-0002-5196-3048>), Evgenia Dimitriadou
[aut, cph], Kurt Hornik <Kurt.Hornik@R-project.org> [aut] (ORCID:
<https://orcid.org/0000-0003-4198-9911>), Andreas Weingessel [aut], Friedrich
Leisch [aut].
2024.
R package version 1.7-16.
https://CRAN.R-project.org/package=e1071.
10.32614/CRAN.package.e1071.

```
----- kableExtra -----
kableExtra: Construct Complex Table with 'kable' and Pipe Syntax.
Hao Zhu <haozhu233@gmail.com> [aut, cre] (ORCID:
<https://orcid.org/0000-0002-3386-6076>).
2024.
R package version 1.4.0.
https://CRAN.R-project.org/package=kableExtra.
10.32614/CRAN.package.kableExtra.
```