From Stats to Brackets:
Applying Machine Learning to NCAA Predictions

Molly O'Connor

Advisor: Jodi Fasteen

Carroll College, Helena MT

Table of Contents

Abstract

Predicting outcomes in the NCAA Men's Basketball Tournament is notoriously difficult due to frequent upsets, inconsistent team performance, and the single-elimination format of March Madness. This project focuses on building a data-driven model specifically for first-round games by leveraging historical tournament results from 2008–2025 alongside KenPom efficiency metrics spanning 2002–2025. The modeling process began with Dean Oliver's "Four Factors" (effective field goal percentage, offensive rebounding rate, turnover rate, and free throw rate) and was expanded to include tempo-adjusted statistics, tournament seeding, and engineered interaction terms such as efficiency per possession and seed–efficiency interactions. A Random Forest classifier was chosen to capture nonlinear relationships among team characteristics, and systematic feature selection ultimately identified seven highly predictive variables, including adjusted efficiency margin, efficiency per possession, seed effects, shooting advantages, and possession-based metrics.

After sequential hyperparameter tuning, the final model achieved a test accuracy of 79.82%, substantially outperforming several baselines, including random guessing (50%), always selecting the higher seed (71.64%), and prior bracket strategies (75%). Statistical testing confirmed that these improvements were significant. To make the model accessible and practical, it was exported to JavaScript and deployed within a web-based graphical interface, allowing users to input team statistics and receive real-time predictions for first-round matchups. Overall, the results demonstrate that combining historical data, advanced basketball metrics, and machine learning can yield robust and interpretable predictions for March Madness. Future extensions could expand the model to later tournament rounds and incorporate dynamic factors such as team momentum, injuries, or coaching strategies.

Introduction

Each March, millions of fans attempt to do the impossible: predict the outcomes of the NCAA Men's Basketball Tournament. Despite the widespread enthusiasm, and the friendly but fierce bracket competitions that have become a tradition in many families, including my own; consistently forecasting game results remains notoriously difficult. Upsets, seeding surprises, and unpredictable performance swings create an environment where even experts struggle to gain an edge.

In recent years, however, advances in sports analytics and machine learning have opened new possibilities for understanding the factors that influence tournament success. Data-driven models can evaluate team efficiency, pace, shooting quality, and other performance indicators far beyond what traditional rankings or "gut feeling" can capture.

This project develops and evaluates a predictive model for first-round March Madness results, using historical tournament outcomes alongside KenPom efficiency metrics. The goal is not only to build a model with high predictive accuracy, but also to determine which statistical features meaningfully contribute to winning. By systematically exploring feature selection, feature engineering, tree-based models, and hyperparameter tuning, this project aims to identify an optimal set of predictors and produce a model that performs competitively against common bracket-picking heuristics (such as always choosing the higher seed).

Background

The NCAA Men's Basketball Tournament, widely known as *March Madness*, is one of the most popular and unpredictable sporting events in the United States. Each spring, 68 college basketball teams compete in a single-elimination tournament to determine the national champion. The first round alone features 32 games played over just two days, producing upsets, dramatic finishes, and bracket-breaking surprises that capture national attention. Millions of fans fill out brackets attempting to predict the winners, even though the odds of creating a perfect bracket are astronomically low (1 in 9.2 quintillion odds!). These odds are so low that there is a free trip to mars, burgers for life, and a million dollar prize waiting for anyone who successfully creates a perfect March Madness bracket (Fore, 2025). The combination of high stakes, volatility, and widespread participation makes March Madness an ideal setting for applying data science and predictive modeling.

To understand how teams perform and what factors influence winning, sports analysts rely on a variety of statistical systems. One of the most foundational frameworks is Dean Oliver's Four Factors, which summarize the core components of efficient basketball performance:

- **Effective Field Goal Percentage (eFG%)** – measures shooting efficiency while accounting for the added value of three-point shots.
- **Turnover Percentage (TO%)** – captures how often a team loses possession before attempting a shot.
- **Offensive Rebound Percentage (OR%)** – reflects a team's ability to extend possessions by retrieving missed shots.
- **Free Throw Rate (FTR)** – indicates how effectively a team generates scoring opportunities at the free-throw line (Poropudas, 2023).

These factors are widely considered the most direct statistical drivers of wins and serve as a foundation for many analytical models.

In addition to box-score statistics, modern college basketball analysis often uses tempo-adjusted metrics developed by Ken Pomeroy ("KenPom"), whose system has become a standard reference among analysts (Cunningham, 2025). KenPom ratings provide measures such as:

- **Adjusted Offensive Efficiency (AdjO)** – points scored per 100 possessions, adjusted for opponent strength.
- **Adjusted Defensive Efficiency (AdjD)** – points allowed per 100 possessions, similarly adjusted.
- **Adjusted Tempo (AdjT)** – the number of possessions a team typically plays per game.
- **Adjusted Efficiency Margin (AdjEM)** – the difference between AdjO and AdjD; one of the strongest indicators of overall team quality (Pomeray, 2005).

Because tournament matchups involve teams with contrasting styles, paces, and schedules, tempo-adjusted statistics offer a more accurate comparison than raw points or win-loss records.

Tournament seeding is another important concept. Each region ranks its teams from 1 to 16, with lower seeds (e.g., 1–4) generally representing stronger teams. While seeding provides a rough prediction of outcomes, upsets are common, especially in the first round, making seeds alone imperfect predictors.

Finally, predictive modeling in sports often benefits from techniques such as:

- **Feature selection**, which identifies the most useful variables and removes redundant or noisy ones .
- **Feature engineering**, which creates new variables that capture interactions or ratios not present in the raw data.
- **Tree-based models**, such as decision trees or random forests, which handle nonlinear relationships and provide interpretable feature importance measures (Kavlakoglu, 2025).

Together, these concepts form the foundation for developing a machine learning model capable of predicting first-round March Madness outcomes. With an understanding of the tournament's structure, the statistical frameworks behind team evaluation, and the types of variables commonly used in sports analytics, the next step is selecting an appropriate modeling technique. For this project, I employed a Random Forest classifier, a tree-based ensemble method well-suited for complex, nonlinear relationships often present in sports performance data. Before describing the data preparation and feature engineering steps, it is helpful to first explain how Random Forests operate and why they are particularly effective for this type of predictive task.

Random Forest Algorithm

A random forest is a machine learning method that makes predictions by combining the results of many smaller models called decision trees.

What is a decision tree?

A decision tree is a model that works like a flowchart. It starts with a big group of data and splits it into smaller and smaller groups based on different features. For example, in this project when predicting March Madness first-round winners, a decision tree might start by asking:

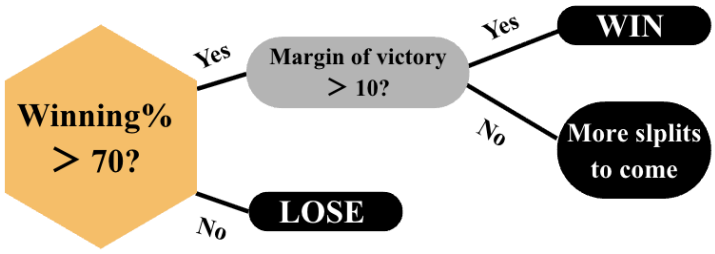
Does the team have a winning percentage above 70%?

- If yes, follow one branch.
- If no, follow another.
-

From there, the tree could split again:

For teams with winning percentages above 70%, is their average margin of victory more than 10 points?

- If yes, predict "Win."
- If no, keep splitting based on other stats like turnovers or three-point shooting.



The process continues until the tree reaches a point where it feels confident making a prediction (a leaf). Meaning the result is either "Win" or "Lose."

The problem is that a single decision tree can be unstable. If you change the training data even slightly, you might end up with a very different tree, and therefore very different predictions. This sensitivity is called high variance. A random forest solves this problem by building not just one tree, but hundreds (sometimes thousands) of them, and then combining their predictions. This approach helps "average out" the instability of individual trees.

The Process for Random Forest

1. **Random sampling of data:** Instead of using the entire dataset to build one tree, the algorithm creates many different "mini datasets." Each mini dataset is made by randomly selecting rows from the original data with replacement. This means that some rows might appear multiple times in one dataset, while others might not appear at all. This technique is called **Bootstrapping**.
    - *Example:* If we have 100 basketball teams, one mini dataset might include Team A three times, and leave out Team B completely, and so on.

2. **Random feature selection:** When each tree is being built, it doesn't use all the features (like rebounds, steals, shooting percentage, etc.) at every split. Instead, it picks a random subset of features to consider. This step prevents all the trees from looking alike and helps reduce correlation between them.
    - *Example:* One tree might decide splits based on rebounds and turnovers, with another tree focusing on free throw percentage and assists.

3. **Building many decision trees:** Each of these randomly sampled datasets and feature subsets produces its own tree. Every tree learns slightly different patterns because it's looking at the problem from a different angle.

4. **Aggregation (majority vote):** Once all the trees are built, the forest makes predictions on new data. Each tree gives its vote ("This team will win" or "This team will lose"), and the forest goes with the majority.
    - *Example:* If 70 out of 100 trees say "Win" and 30 say "Lose", the forest predicts "Win".

Why Random Forests are Powerful

- **Reduced overfitting:** A single decision tree can "memorize" the training data too closely, which makes it bad at predicting new data. By averaging many trees, the random forest smooths out these quirks.
- **Better accuracy:** With many trees, the forest can capture a wider variety of patterns in the data.
- **Robustness:** Random forests work well even when some features are noisy, irrelevant, or missing (Soni, 2023).

Random Forest Example

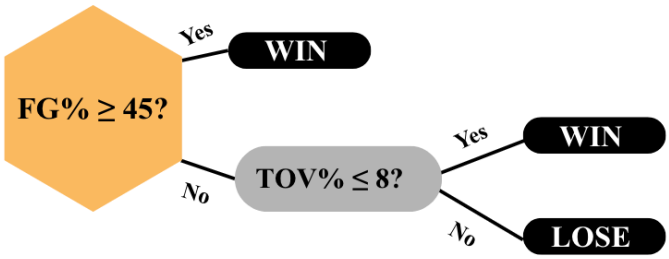
Let's start with a smaller subset of basketball data:

ID	FG%	FT%	Rebounds	Assists	TOV%	Prediction
0	**52**	**74**	**45**	**12**	**15**	Win
1	**44**	**67**	**28**	**16**	**24**	Lose
2	**43**	**82**	**32**	**24**	**8**	Win
3	**40**	**70**	**24**	**8**	**11**	Lose

Suppose we grow a decision tree on this data. The tree will split on features to try and separate Wins from Losses. Let's say the tree learns this simple rule.

Root: **FG% ≥ 45 ?**

- **Yes** → Predict **Win**
- **No** → Check **TOV% ≤ 8?**
  - **Yes** → Predict **Win**
  - **No** → Predict **Lose**



This single tree does a good job, but remember: a different sample of data might make the tree split differently and give different results.

Now let's say we create a small forest using three trees, each on a random bootstrap sample of the data and with random feature choice.

Step 1: Bootstrap samples

- **Tree A data sample:** IDs [0 1, 2, 2]
- **Tree B data sample:** IDs [1, 1, 3, 0]
- **Tree C data sample:** IDs [2, 3, 3, 1]

Step 2: Random feature subsets

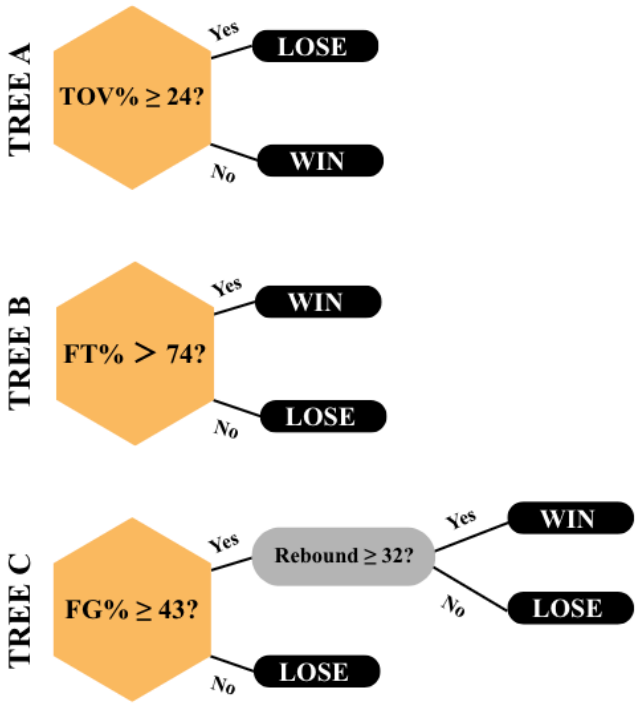
- **Tree A uses:** FG%, TOV%
- **Tree B uses:** Assists, FT%
- **Tree C uses:** Rebounds, FG%

Step 3: Build the trees

- **Tree A:** Sees sample [0, 1, 2, 2] and only looks at FG% and TOV%. It might learn:
  - If TOV% ≥ 24 → Predict **Lose**
  - Else → Predict **Win**

- **Tree B:** Sees sample [1,1,3,0] and only looks at Assists and FT%. It might learn:
  - If FT% $>$ 74 → Predict **Win**
  - Else → Predict **Lose**
- **Tree C:** Sees sample [2, 2, 3, 1] and only looks at Rebounds and FG%. It might learn:

  Root: **FG%** ≥ 43 **?**
  - **No** → Predict **Lose**
  - **Yes** → Check if **Rebound** ≥ 32?
    - **Yes** → Predict **Win**
    - **No** → Predict **Lose**



Step 4: Making predictions with the Random Forest

Once all the trees are trained, we can use the random forest to predict new outcomes. The process is straightforward. Take the new data point - for example, a team with the following stats:

FG%	FT%	Rebounds	Assists	TOV%	Prediction
46	78	30	14	12	TBD

Then, we pass the stats through each tree according to the rules the tree learned:

- **Tree A (**features: FG%, TOV%):
    - Rule: If TOV% ≥ 24 → Predict Lose, Else → Predict Win
    - So, because TOV% = 12, which is less than 24 → **Tree A Predicts Win**
- **Tree B (**features: Assists, FT%**)**
    - Rule: If FT% > 74 → Predict Win, Else → Predict Lose
    - So, because FT% = 78, which is greater than 24 → **Tree B Predicts Win**
- **Tree C (**features: Rebounds, FG%**)**
    - Rule: If FG% < 43 → Predict Lose, Else → If Rebound ≥ 32→ Predict Win, Else → Predict Lose
    - So, because FG% = 46, then we look at Rebounds, and because Rebounds = 30 which is less than 32 → **Tree C Predicts Lose**

Now, we aggregate the predictions:

- Tree A → **Win**
- Tree B → **Win**
- Tree C → **Lose**

The majority vote is Win, so the random forest predicts that this team is likely to **Win.**

How the Decision Tree makes a rule:

A decision tree decides its rules by systematically splitting the data based on the features that best separate the outcomes. At each step in the tree, the algorithm evaluates all available features such as field goal percentage, turnovers, rebounds, or assists and tests different threshold values for each feature. For example, it might consider whether FG% ≥ 45 provides a meaningful separation between Wins and Losses. To determine which split is "best," the tree calculates a measure of impurity or an entropy, a scientific concept that measures a system's disorder (CSRC, 2010). These metrics quantify how mixed the outcomes are within a group: a perfectly "pure" group would contain only Wins or only Losses, while a mixed group would contain both. The feature and threshold that result in the most pure separation become the rule for that branch of the tree.

Once a split is made, the tree repeats this process recursively on each resulting subset. For instance, after splitting teams with FG% ≥ 45, the tree may then consider turnovers within this group to further separate Wins from Losses. This continues until the subsets are sufficiently pure or another stopping criterion is reached, such as a minimum number of observations in a leaf. The end result is a hierarchical structure of decisions, where each internal node represents a condition on a feature, each branch represents the outcome of that condition, and each leaf represents a predicted class, such as "Win" or "Lose."

Mathematically, the choice of each split aims to maximize the information gain or reduction in impurity. Information gain measures how much uncertainty about the outcome is reduced by

splitting the data on a particular feature. A feature that perfectly separates Wins from Losses provides the maximum information gain, while a less informative feature results in smaller gain. By repeatedly selecting the feature and threshold that maximize this measure, the tree constructs a series of rules that systematically reduce uncertainty about the outcome (Simic, 2022). This process allows the tree to model complex relationships between features and predictions, even when interactions between features are nonlinear or hierarchical.

However, a single decision tree can be sensitive to small changes in the training data, leading to high variance in predictions. Random forests address this by building many decision trees on different random subsets of the data and features, then aggregating their predictions. Each tree in the forest follows the same process of selecting splits based on impurity reduction, but because they see slightly different data and consider different features at each split, the ensemble of trees produces more stable and accurate predictions (Soni, 2023). Essentially, the forest averages out the variability of individual trees, making the model robust and reliable.

In conclusion, the random forest algorithm proves especially useful for predicting basketball game outcomes because it reduces the weaknesses of relying on a single decision tree. A lone tree might make unstable predictions depending on which teams or stats it sees during training, but a forest of many trees, each trained on different bootstrap samples and random subsets of features balances out these instabilities. In our March Madness example, individual trees focused on different aspects of team performance, such as shooting percentages, rebounds, assists, and turnovers. While one tree might have leaned heavily on free throw percentage, another considered turnovers, and a third combined shooting and rebounding. When we introduced a new team's stats, the trees disagreed at times, but the random forest combined their "votes" to give a more reliable overall prediction. Instead of depending on just one statistic, the random forest integrates many, producing a more stable and accurate forecast. For basketball predictions, this makes the method especially valuable: it captures the complexity of the game, accounts for variation across teams, and reduces the risk of overfitting to past performance. Ultimately, random forests give us a stronger, data-driven way to anticipate which teams are most likely to win.

The Process

Data Collection and Preparation

To build a predictive model for first-round March Madness outcomes, I first assembled and cleaned two complementary datasets:

1. March Madness Game Results (2008-2025) – containing tournament outcomes, seeds, team names, and round-by-round performance.

2. KenPom Metrics (2002–2025) – providing efficiency ratings and advanced statistics for every Division I team.

Although both datasets include valuable information, they differ in structure, naming conventions, and update timing, making careful preprocessing essential.

Data Filtering and Cleaning

Because post-tournament statistics can leak information about game outcomes, I restricted the KenPom dataset to pre-tournament values only. I also limited the results dataset to games from 2008–2025, which provided a sufficiently large range of seasons with consistent KenPom coverage. Team names varied across datasets (e.g., abbreviations, hyphens, or alternate spellings), so I standardized names and manually resolved mismatches. Any teams that could not be reliably aligned were removed to avoid incorrect merges.

Merging Datasets

After cleaning, I merged the datasets on team name and season, creating a combined table containing both:

- team-level efficiency metrics
- tournament seeding
- the round in which each team exited

This resulted in a unified dataset that could be used for modeling, feature engineering, and exploratory analysis. I saved the merged file as a master dataset to streamline later experimentation.

Feature Understanding and Exploration

Before modeling, I familiarized myself with all available variables. I created descriptive summaries to organize key KenPom concepts such as efficiency margin, tempo, and shooting

metrics. This helped guide the selection of features that might meaningfully influence first-round tournament outcomes. Full details of all features are provided in the appendix (see *Variable Descriptions, page 35*)

Train-Test Split

To evaluate the model's predictive performance on unseen data, the dataset was divided into training and test subsets (70% training, 30% testing). The training set was used to fit the models and identify important features, while the test set was held out to provide an unbiased estimate of accuracy. This separation is critical in machine learning, as it prevents overfitting and ensures that model performance reflects its ability to generalize to new games rather than just memorizing historical results. By maintaining this structure, all subsequent modeling steps, including feature selection, engineering, and evaluation, were performed on the training data, with the test set serving as the benchmark for assessing real-world predictive capability.

Dean Oliver's "Four Factors"

Initially, I selected a subset of predictive variables based on Dean Oliver's "Four Factors" philosophy, which he expanded upon in 2004 in *Basketball on Paper* (Oliver). These factors, effective field goal percentage (eFGPct), offensive rebound percentage (ORPct), free throw rate (FTRate), and turnover percentage (TOPct), represent key strategies historically emphasized by coaches to influence game success.

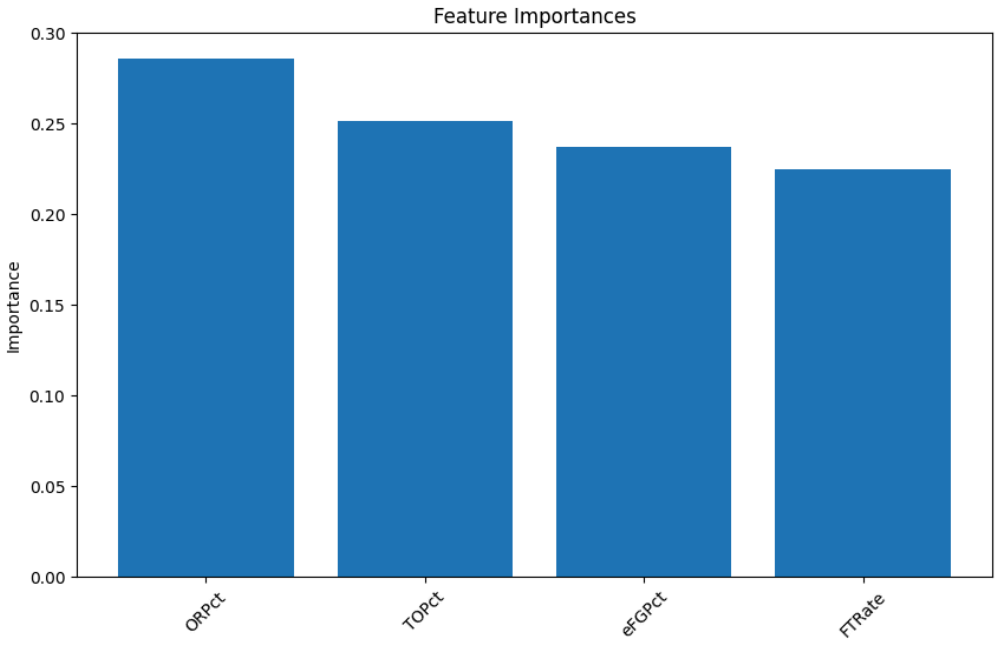


Figure 1.1: Feature Importance for Dean Oliver's "Four Factors"

Training a Random Forest model using only these four features produced an initial test accuracy of 60%. Because this is a binary classification problem (win vs. loss), the baseline accuracy (what we would expect if the model had no predictive power and simply guessed) is **50%**. In other words, any accuracy meaningfully above 50% indicates that the model is learning some signal beyond random chance. With that context, the 60% accuracy suggests that these four metrics provide some predictive value, but their power is still limited. **Figure 1.1** illustrates the relative importance of each feature in the model. Although ORPct emerged as the most influential variable, all four factors contributed relatively evenly, with importance values within approximately 5% of one another. Recognizing that team seed is an important factor in bracket outcomes, I added seed as a feature. This adjustment increased the model's accuracy to **69%**.

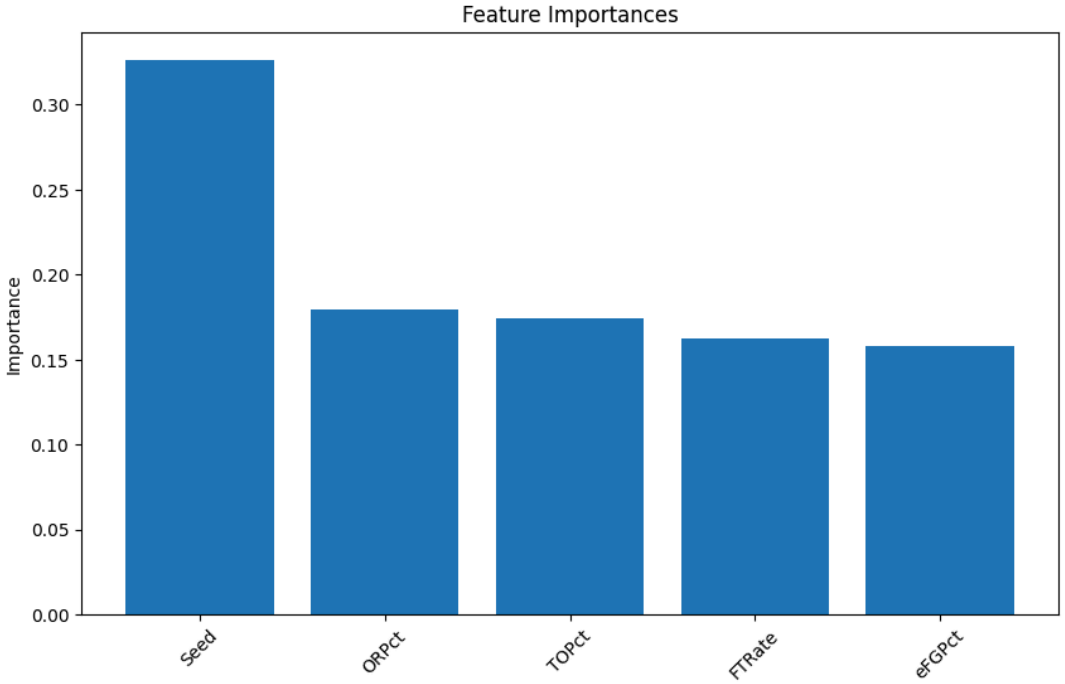


Figure 1.2: Feature Importance for Dean Oliver's "Four Factors" + Tournament Seed

Figure 1.2 demonstrates the significant impact of tournament seed on predictive performance. When seed was added to the model, it emerged as the most important feature, accounting for approximately 34% of the model's importance, while the relative contribution of the original four factors dropped below 20% each. This highlights the strong influence of seeding on first-round outcomes. Subsequently, I expanded the model to include all available variables available in the dataset.

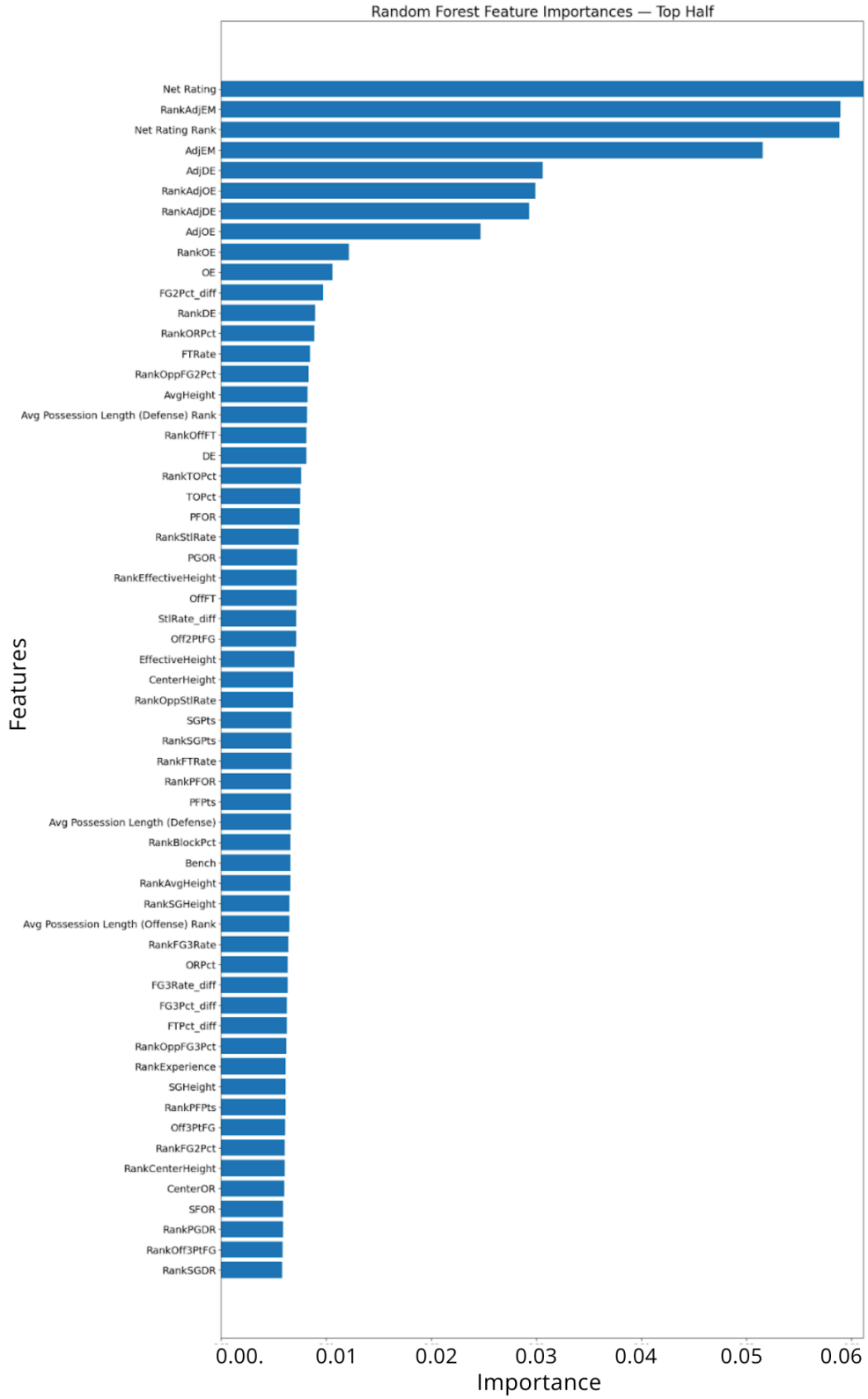


Figure 1.3: Feature Importance for all variables (first half)

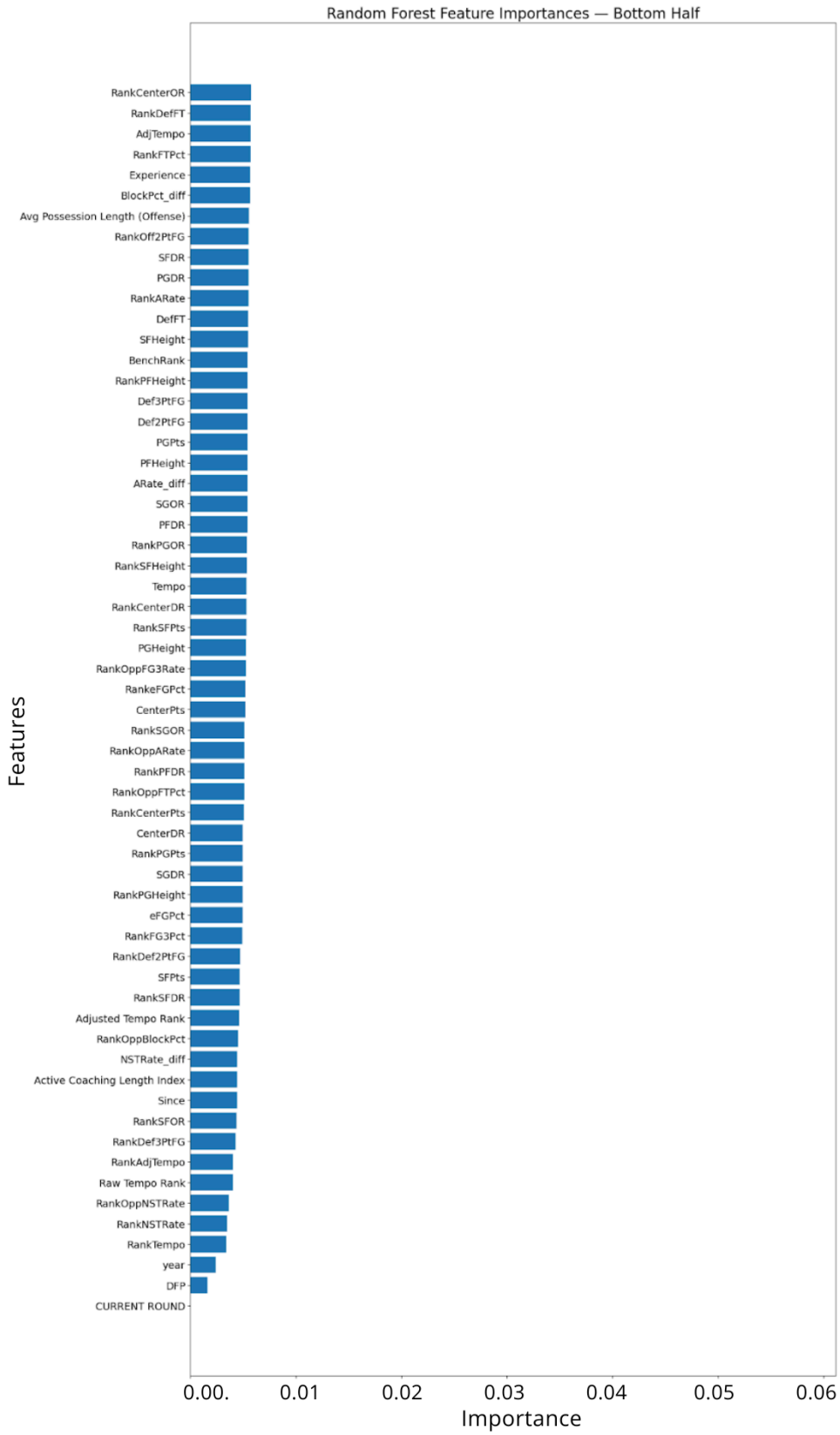


Figure 1.4: Feature importance for all variables (second half)

The model's accuracy further improved to **75%**, indicating that the inclusion of additional features provided meaningful predictive information. However, as **Figure 1.3 and Figure 1.4** shows, the full dataset contains 132 variables, a large number that is hard to work with, which also introduces potential noise that may not improve the model's performance. To determine the optimal number of features, I implemented a loop that tested the model performance across every possible subset of top features ranked by importance. The highest accuracy, **77.49%**, was achieved using the top 8 features.

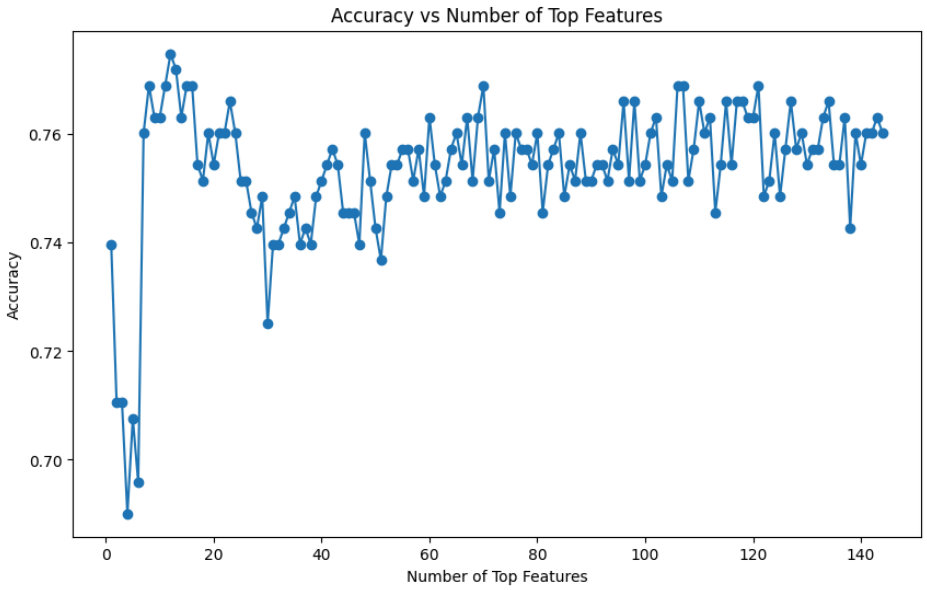


Figure 1.5: Exploring Accuracy vs the Number of Features Used

Figure 1.5 illustrates how model performance changes as different features are added or removed. The highest accuracy, **77.49%**, was achieved using the top eight features highlighting the benefit of focusing on the most informative variables.

The top eight features identified were:

1. **Net Rating** – Overall point differential per 100 possessions, capturing a team's net performance.
2. **AdjEM** – Adjusted efficiency margin, representing a team's overall strength after accounting for opponent quality.
3. **AdjEM Rank** – Ranking of teams based on adjusted efficiency margin, showing relative team strength.
4. **Net Rating Rank** – Ranking of teams based on net rating, reflecting overall performance compared to others.
5. **AdjOE** – Adjusted offensive efficiency, measuring points scored per 100 possessions against average opponents.

6. **AdjDE** – Adjusted defensive efficiency, measuring points allowed per 100 possessions against average opponents.
7. **AdjOE Rank** – Ranking of teams based on adjusted offensive efficiency, indicating relative scoring efficiency.
8. **AdjDE Rank** – Ranking of teams based on adjusted defensive efficiency, indicating relative defensive strength.

Building on this, the next step was to further refine the feature set by removing less meaningful or redundant variables, in order to reduce noise and improve model interpretability.

Feature Refinement

Recognizing that including too many variables can reduce model accuracy, the next step was to further refine the feature set by removing less meaningful or redundant features. For instance, each statistic in the dataset had a corresponding rank variable, indicating how a team compared to all Division I programs in that category. A team with the highest offensive efficiency would receive a rank of 1, making these variables largely redundant with the original statistics. I made a new data set with all rank variables removed, and ran a random forest including all variables in the new data set. The graph can be seen on the following page:

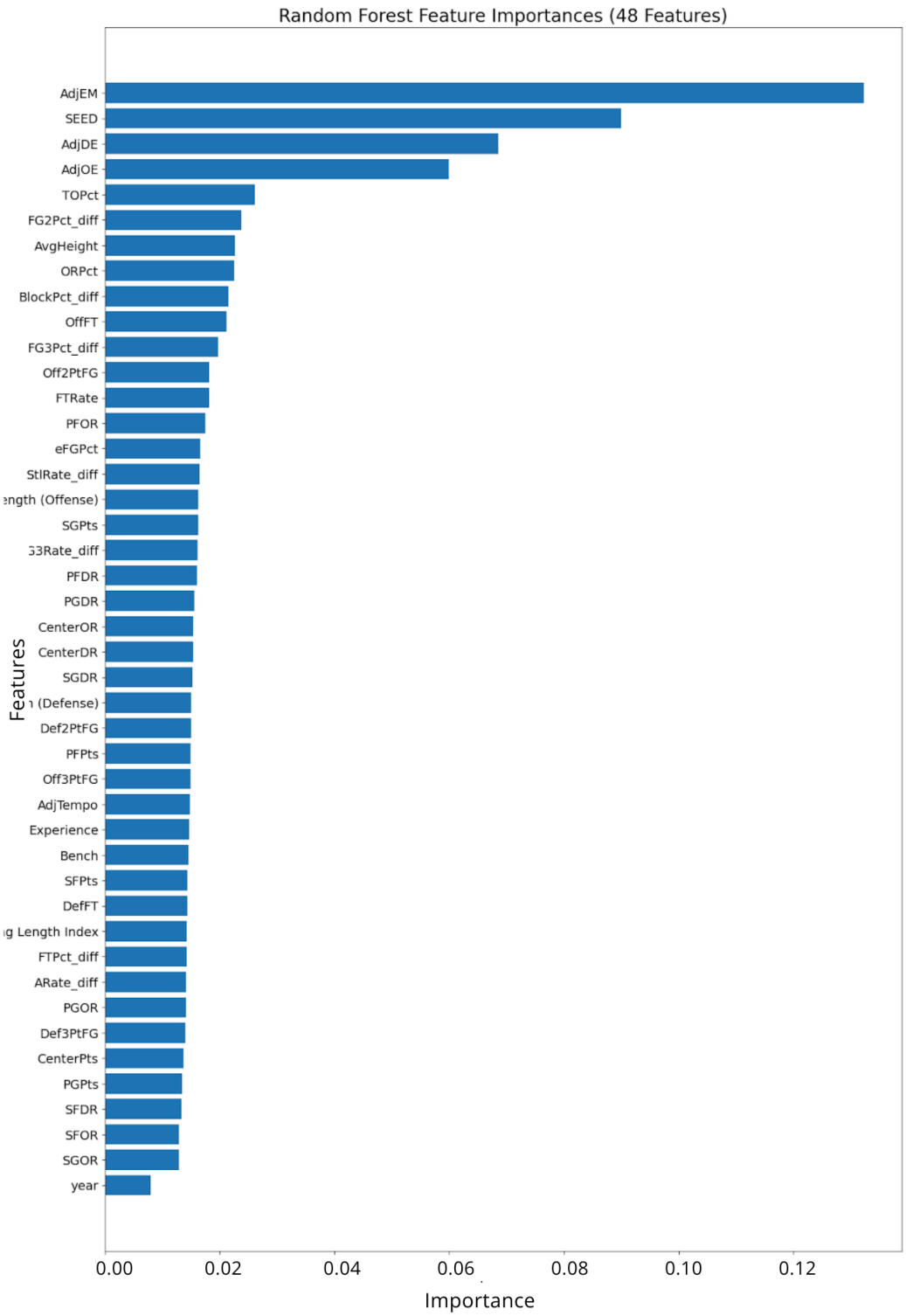


Figure 1.6: Feature Importance for all variables (rank not included)

Using this refined dataset, we achieved a new accuracy of **76.608%**. I then took it a step further by removing additional variables that showed strong correlations with others. For example, AdjEM measures a team's overall performance and represents the expected point differential per 100 possessions against an average Division I team. It is calculated by subtracting a team's Adjusted Defensive Efficiency (AdjD) from its Adjusted Offensive Efficiency (AdjO). Since AdjEM already incorporates both AdjO and AdjD, those variables were removed to avoid redundancy.

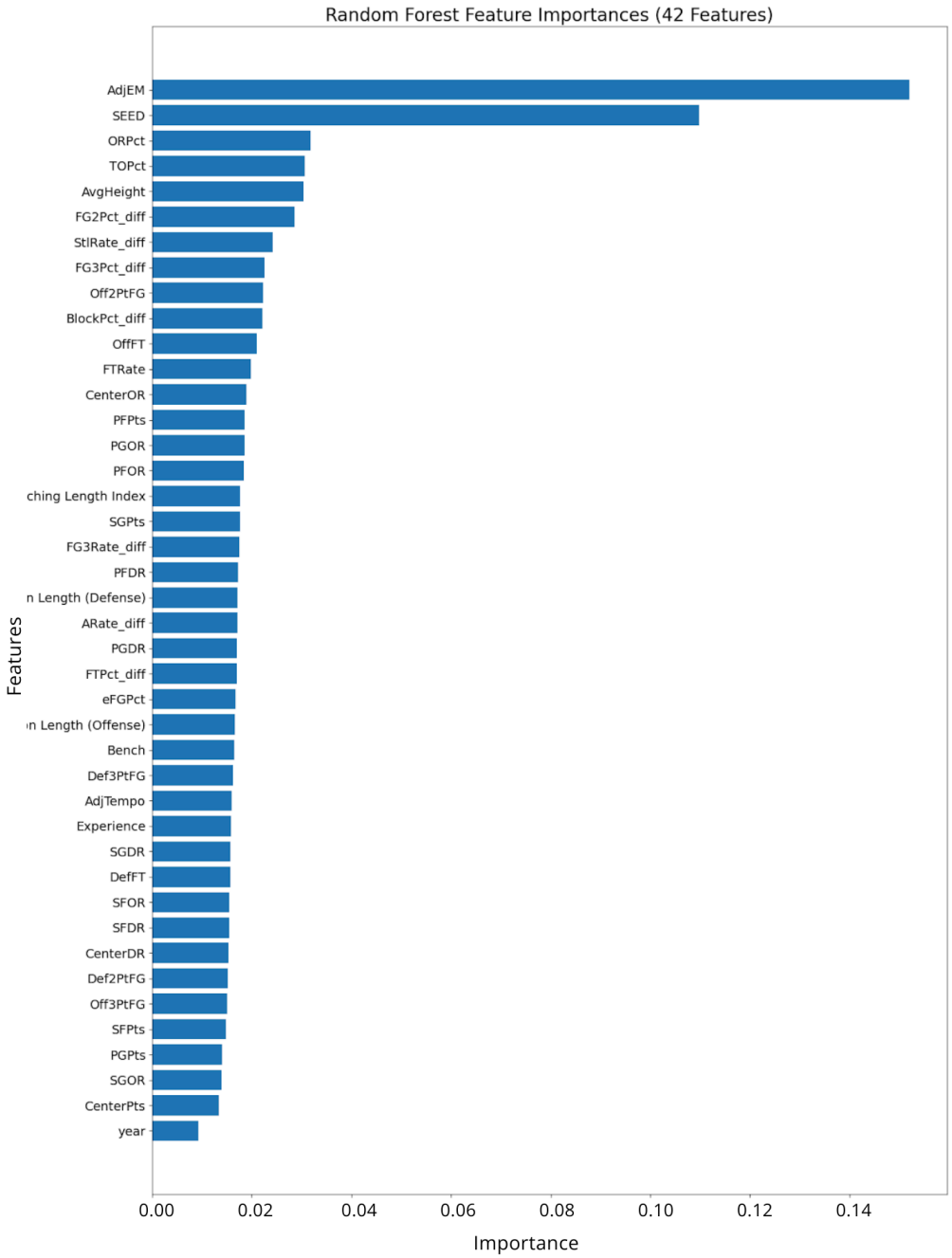


Figure 1.7: Feature Importance for all variables (least redundant data set)

Creating a Random Forest using all the features in the revised dataset resulted in an accuracy of **77.485%**, the highest among the three datasets tested (all features, no rank variables, and no rank variables with redundant features removed). Given the strong performance and the reduced complexity, the dataset with the fewest variables was selected for all subsequent modeling steps.

Feature Engineering

To further enhance the model's predictive power, several engineered features were created to capture interactions and relationships that were not represented in the original dataset:

1. **AdjEM/AdjTempo Ratio:** Calculated as the adjusted efficiency margin divided by adjusted tempo (AdjEM / AdjTempo), this feature reflects a team's efficiency per possession, providing a normalized measure of scoring and defensive effectiveness relative to pace.
2. **Height × Experience Interaction**: By multiplying average team height with average experience, this feature captures the combined advantage of taller, more seasoned teams, which are often more effective in rebounding, defense, and situational decision-making.
3. **Seed × AdjEM Interaction:** This interaction models how a team's tournament seed interacts with its overall strength. Stronger teams with higher seeds may be more likely to perform as expected, while lower-seeded teams might face greater difficulty overcoming stronger opponents.

These engineered features were added to the refined dataset, creating a richer set of variables designed to improve the model's ability to distinguish between competing teams. By incorporating interactions and normalized metrics, the model can better capture subtle patterns that influence game outcomes.

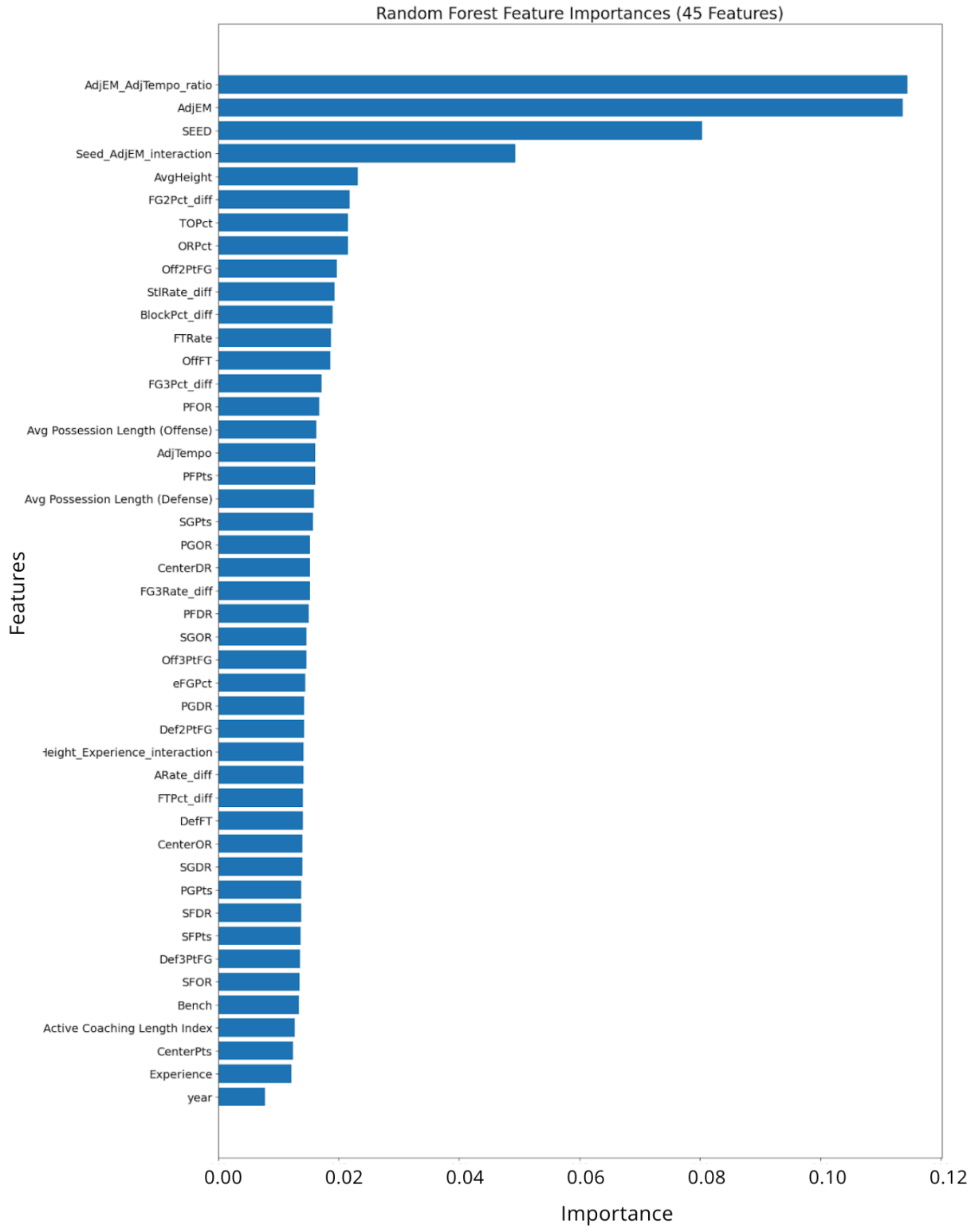


Figure 1.8: Feature importance on enhanced data set

Applying a Random Forest model to this enhanced feature set allowed the model to leverage these additional signals, resulting in an improved test accuracy of **77.7%**. This increase demonstrates that the engineered features provided meaningful predictive information beyond the original variables. As with earlier modeling steps, the next goal was to identify the optimal subset of features to maximize predictive performance while minimizing noise. To achieve this, a systematic search was conducted: the model was evaluated on every possible subset of the top-ranked features, based on their importance in the Random Forest. This process revealed that using the top seven features produced the highest test accuracy of **79.53%**, demonstrating that a carefully selected, smaller set of variables can outperform models that include all features.

The **top seven features** identified were:

9. **AdjEM_AdjTempo_ratio** – efficiency per possession, capturing scoring and defensive effectiveness relative to pace.
10. **AdjEM** – adjusted efficiency margin, a measure of overall team strength.
11. **SEED** – tournament seed, reflecting expectations based on historical performance and rankings.
12. **Seed_AdjEM_interaction** – models how team strength interacts with tournament seed.
13. **FG2Pct_diff** – difference in two-point field goal percentage between competing teams.
14. **ORPct** – offensive rebound percentage, indicating second-chance scoring opportunities.
15. **TOPct** – turnover percentage, reflecting ball security and efficiency.

Hyperparameter Tuning

After identifying the top seven features that produced the strongest predictive accuracy, the next step was to optimize the Random Forest model itself. Although Random Forests are powerful algorithms, their performance depends heavily on a set of internal settings called hyperparameters. These are not learned from the data; instead, they control how the model is built. For example, how many decision trees it uses or how deep those trees are allowed to grow. Adjusting these settings can either improve accuracy or cause the model to overfit the noise in the data. To systematically search for the best-performing combination, I used a technique called Grid Search Cross-Validation. Grid Search CV works by testing every possible combination of chosen hyperparameter values, evaluating each version of the model, and selecting the one that performs best. Rather than guessing which settings might work well, Grid Search ensures that the model is tuned in a structured, unbiased way.

The following hyperparameters were selected because they directly influence the shape and behavior of the Random Forest:

- **Number of trees (n_estimators)** – More trees can improve performance but require more computation.
- **Maximum depth of each tree (max_depth)** – Limits how many splits a tree can make; shallow trees reduce overfitting.
- **Minimum samples required to split a node (min_samples_split)** – Controls how easily a branch forms.
- **Minimum samples at each leaf (min_samples_leaf)** – Prevents nodes from becoming too small and overly specific.

I used a sequential, one-at-a-time approach for hyperparameters rather than a full grid search so I could better understand the parameters individually. First, I investigated the number of trees (n_estimators) while leaving all other hyperparameters at their default values (which are used automatically by Python's RandomForestClassifier: max_depth=none, min_samples_split=2, min_samples_leaf=1). Once the optimal number of trees was selected, I held that value constant and moved on to tune the next parameter, such as max_depth. This process was repeated for each subsequent hyperparameter.

For each of these hyperparameters, I defined several values for Grid Search to test. For example, I tested models with anywhere from 100 to 600 trees and tree depths ranging from shallow to unlimited.

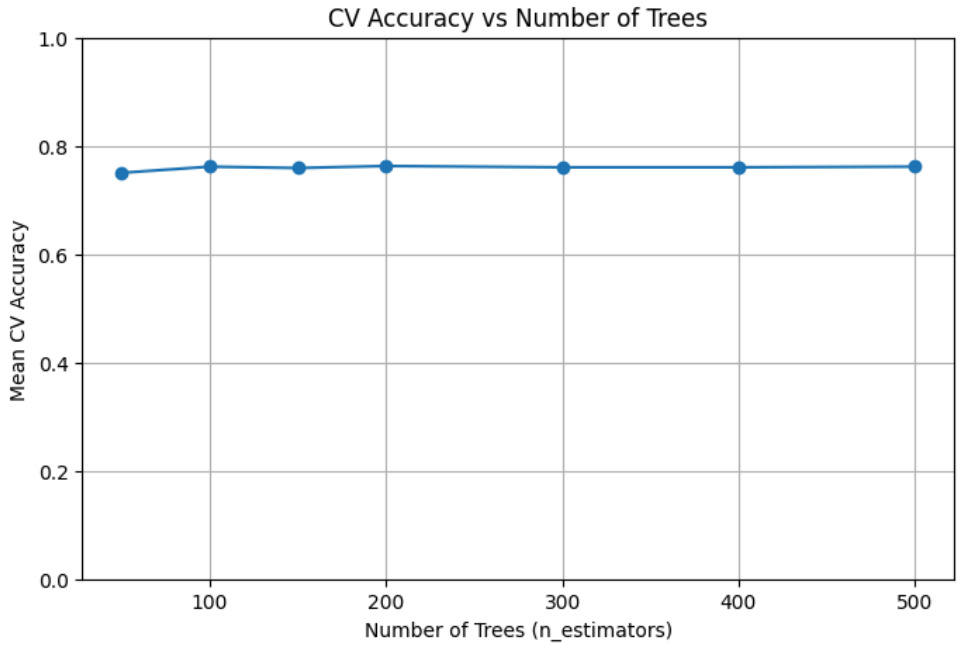


Figure 1.9: Number of Trees effect on Model Accuracy

Examining **Figure 1.9**, most of the variation appears to occur between 75% and 80% accuracy. Zooming in on this range allows us to better understand the subtle differences in model performance.

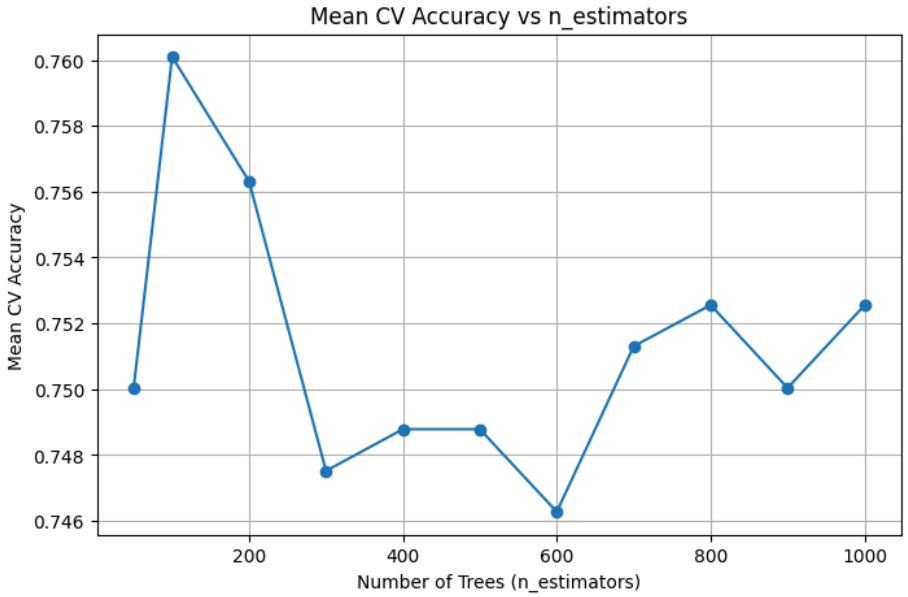


Figure 1.10: Number of Trees effect on Model Accuracy (zoomed in)

The first parameter we examined was n_estimators, which represents the number of decision trees included in the Random Forest. In our tuning process, we evaluated a range of values for n_estimators and plotted each one against the model's performance. Based on the resulting graph, the model achieved its highest accuracy when using **100 trees**. Moving forward, I used 100 as the n_estimators and investigated max_depth next.

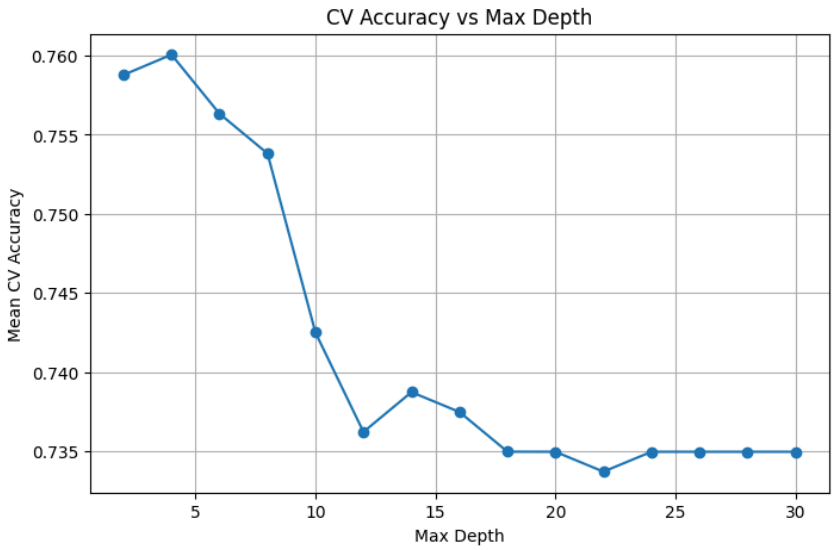


Figure 1.11: Max Tree Depth effect on Model Accuracy

Next, we examined the max_depth parameter, which controls how deep each decision tree is allowed to grow, essentially determining how many splits and branches the model can make. Our graph plotting accuracy across different depth values shows a peak at a max_depth of **4**. At this depth, the model captures enough complexity to improve predictions without becoming overly specialized. Beyond 20, accuracy no longer increases and instead begins to level off, suggesting that deeper trees do not provide additional benefit for this dataset.

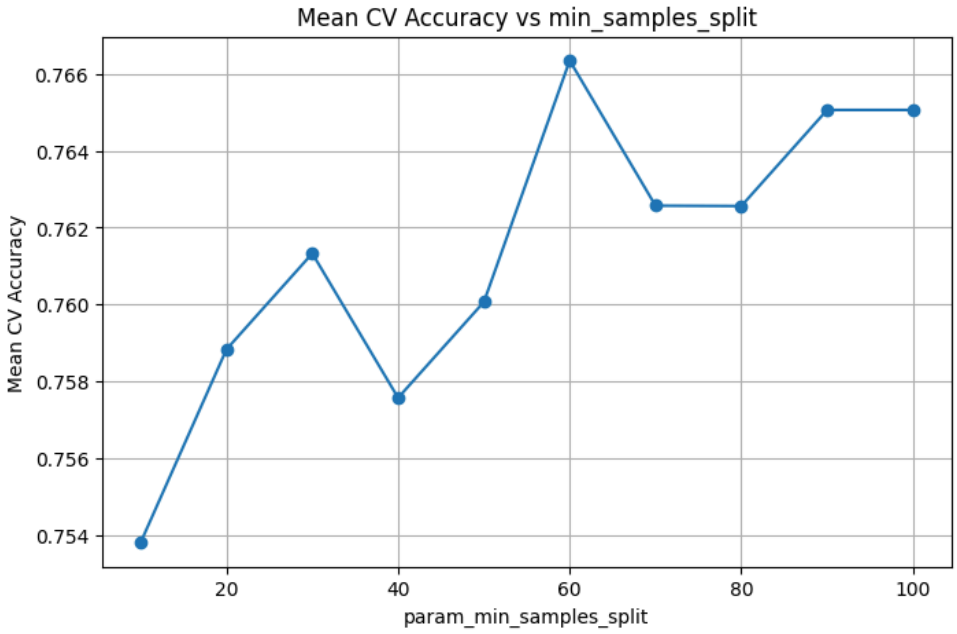


Figure 1.12: Minimum Samples Split effect on Model Accuracy

Next, we analyzed the min_samples_split parameter, shown in **Figure 1.12**. This parameter controls the minimum number of samples required for a node to split and form additional branches. Smaller values allow the tree to split more freely, which can capture subtle patterns but also increases the risk of overfitting. Larger values restrict splitting, helping the model generalize better but potentially overlooking important structure. Our results indicate that the model achieves its highest accuracy at a min_samples_split of **60**.

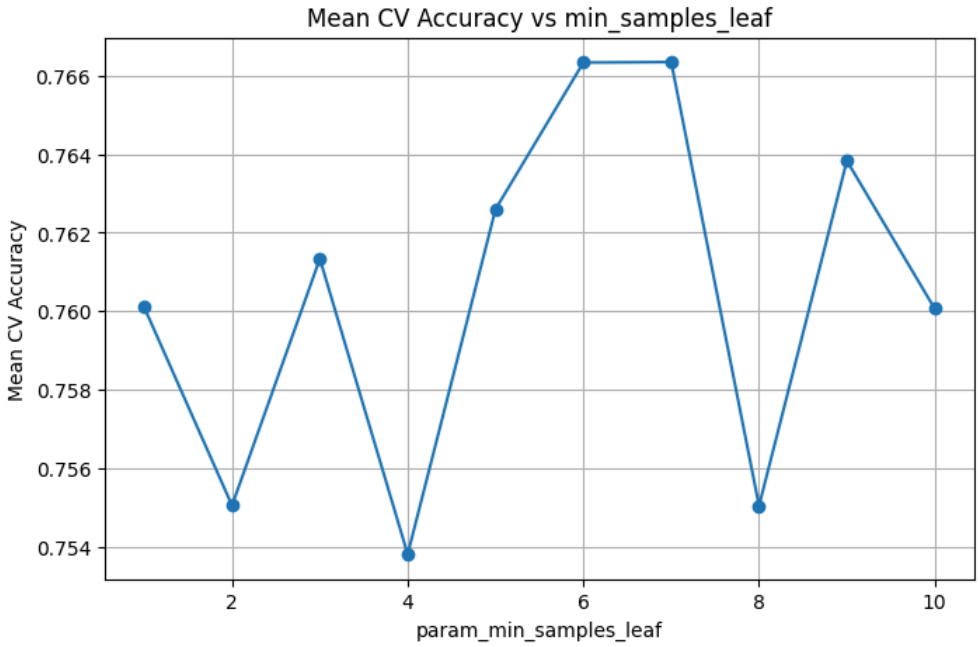


Figure 1.13: Minimum Samples Leaf effect on Model Accuracy

Finally, we examined the min_samples_leaf parameter, which determines the minimum number of samples required for a terminal node (a leaf). This parameter influences how "fine-grained" the model's final decisions can be. Smaller values allow the tree to create very specific leaves, which may lead to overfitting, while larger values force the model to generalize more broadly across the data.As shown in our results, the model achieved its highest accuracy when min_samples_leaf was set to **7**.

Together, these tuning experiments reveal how each hyperparameter shapes the model's complexity, generalization ability, and overall predictive strength. By systematically evaluating the impact of the number of trees, tree depth, and node-size constraints, we were able to identify the combination that consistently produced the highest accuracy on the validation data. The tuning results showed clear performance peaks at:

- **n_estimators = 100**
- **max_depth = 4**
- **min_samples_split = 60**
- **min_samples_leaf = 7**

Hyperparameter tuning was performed by adjusting parameters one at a time rather than using a full grid search. This approach made it easier to isolate the effect of individual parameters on model performance, but it does not account for interactions between parameters. In practice, Random Forest hyperparameters are interdependent; for example, a change in the number of estimators may alter which maximum tree depth performs best. A grid search evaluates parameter combinations simultaneously and is therefore better suited to capturing these interactions. This interdependence is what makes hyperparameter tuning particularly challenging. As a result, while the selected parameters improved model performance, a more comprehensive grid search could potentially identify better-performing combinations.

However, using these optimized hyperparameters, I retrained the Random Forest model on the training data and evaluated it on the test set to confirm its performance. This fully tuned model achieved a test accuracy of **79.82%**, improving slightly over the previous untuned models. The modest gain is typical: most of the predictive power was already captured with careful feature selection and initial parameter choices.

Model Evaluation & Comparison

1. Random Forest vs. Always Pick Team A
The naïve strategy of always picking Team A achieved an expected accuracy of 50%. Comparing this to the Random Forest's 79.82% accuracy, the z-test resulted in a Z-statistic of 13.7438 and a p-value of 0.0000, indicating a highly significant improvement. This demonstrates that the model captures real predictive patterns in the data rather than performing well by chance.

2. Random Forest vs. Higher Seed Strategy
Using tournament seeding alone to predict winners gave an accuracy of 71.64%. The Random Forest still outperformed this approach, with a Z-statistic of 3.7716 and a p-value of 0.0001, confirming that combining multiple features and engineered variables improves predictions beyond simple seed-based logic.

3. Random Forest vs. Previous Bracket Picks
My bracket picks from last March achieved 75% accuracy in the first round. Even against this stronger baseline, the Random Forest showed a statistically significant improvement, with a Z-statistic of 2.2233 and a p-value of 0.0131. While the gain is smaller, it reinforces that the model generalizes better than personal heuristics or historical picks.

Overall, these tests indicate that the Random Forest's higher accuracy is statistically significant across all comparisons, validating its ability to predict first-round outcomes more effectively than simpler or historical strategies.

Deploying the Model to a Web Application

Once the Random Forest model was fully trained and tuned, the next step was to make it accessible for real-world use. To achieve this, I exported the model from Python to JavaScript, allowing it to be embedded directly into a GUI-based web application. This approach enables users to interact with the model in a familiar, browser-based environment without needing to run Python or install specialized software.

The model export process involved converting the trained Random Forest into a format compatible with JavaScript, preserving all decision trees, feature splits, and thresholds. This ensures that the predictions made in JavaScript are identical to those generated in Python. After export, I integrated the model into a web interface where users can input team statistics for a given matchup, such as adjusted offensive efficiency, player height, and seeding. The application then outputs a predicted winner for the first-round game, along with associated probabilities, in real-time. This deployment demonstrates the practical utility of the model: it is not only a tool for analysis in a research context but also a user-friendly application that can assist fans, or analysts in making data-driven predictions.

Limitations and Future Work

While the Random Forest model performed well for first-round predictions, there are several opportunities for improvement. The model currently focuses on historical team statistics and does not account for dynamic factors such as team momentum, injuries, or coaching strategies, which can influence outcomes. Additionally, upsets in March Madness often have complex causes, and understanding these patterns could help refine predictive features.

Future work could explore these factors, incorporating variables that capture streaks, player availability, or psychological and situational effects. Expanding the model to predict outcomes in later rounds would also provide a more comprehensive tournament tool, as games beyond the first round often display different dynamics and feature importance. By investigating these additional variables and tournament stages, the model could become both more accurate and more insightful, helping to explain why certain upsets occur and improving its practical application for fans and analysts alike.

Conclusion

This project developed a Random Forest model to predict first-round March Madness outcomes. By carefully selecting and engineering features, and tuning hyperparameters, the model achieved a test accuracy of 79.82%, outperforming simple baselines such as always picking the higher seed or relying on previous bracket results. Statistical tests confirmed that these improvements are meaningful, demonstrating that the model captures real patterns in team performance.

The model was also made practical by exporting it to JavaScript and integrating it into a GUI-based website, allowing users to make predictions in real time. While the current model focuses on first-round games, future work could expand predictions to later rounds, incorporate additional factors such as team momentum, and explore the underlying reasons for upsets.

Overall, this project demonstrates the exciting potential of machine learning in sports, showing how data-driven models can reveal new insights, enhance understanding of complex competitions, and create engaging, interactive experiences for fans and analysts alike.

References

Cunningham, Nate. “KenPom Rankings Explained: Who Is Ken Pomeroy and What Do His Rankings Mean?” *SI*, Sports Illustrated, 17 Mar. 2025, www.si.com/college-basketball/kenpom-rankings-explained-who-is-ken-pomeroy-what-do-rankings-mean.

CSRC Content. “Entropy - Glossary | CSRC.” *Csrc.nist.gov*, csrc.nist.gov/glossary/term/Entropy.

Fore, Preston. “A Free Trip to Mars, Free Burgers for Life, and $1 Million: What a Perfect March Madness Bracket Could Win You as Warren Buffett and Elon Musk Chip In.” *Fortune*, 18 Mar. 2025, fortune.com/2025/03/18/march-madness-perfect-bracket-billionaires-elon-musk-warren-buffett-prizes/.

GeeksforGeeks. “Random Forest Algorithm in Machine Learning.” *GeeksforGeeks*, 22 Feb. 2024, www.geeksforgeeks.org/machine-learning/random-forest-algorithm-in-machine-learning/

“Introduction to Oliver’s Four Factors.” *Squared Statistics: Understanding Basketball Analytics*, 6 Sept. 2017, squared2020.com/2017/09/05/introduction-to-olivers-four-factors/.

Kavlakoglu, Eda. “The 2025 Guide to Machine Learning.” *IBM*, 2025, www.ibm.com/think/machine-learning#605511093.

Oliver, Dean. *Basketball on Paper: Rules and Tools for Performance Analysis*. Potomac Books, 2004.

Pomeray, Ken. *Stats Explained*. 1 Aug. 2005, kenpom.com/blog/stats-explained/.

Poropudas, Jirka, and Topi Halme. "Dean Oliver's Four Factors Revisited." *ArXiv.org*, 2023, arxiv.org/abs/2305.13032.

Melkonyan, Lilit. "Statistics Decision Tree Definition and Examples." *Plat.AI*, 5 Apr. 2023, plat.ai/blog/statistic-decision-tree-definition/.

Normalized Nerd. "Random Forest Algorithm Clearly Explained!" *Www.youtube.com*, 21 Apr. 2021, www.youtube.com/watch?v=v6VJ2RO66Ag.

Simic, Milos. "Information Gain in Machine Learning | Baeldung on Computer Science." *Www.baeldung.com*, 17 Feb. 2022, www.baeldung.com/cs/ml-information-gain.

Soni, Brijesh. "Why Random Forests Outperform Decision Trees: A Powerful Tool for Complex Data Analysis." *Medium*, 15 Mar. 2023,

medium.com/@brijesh_soni/why-random-forests-outperform-decision-trees-a-powerful-tool-for-complex-data-analysis-47f96d9062e7.

Appendix

Variable Descriptions

Variable	Description
year	Season year of the data.
Short Conference Name	Conference name the team belongs to.
Adjusted Temo	Basketball performance or tournament metadata variable: estimate of the speed at which a team would play against an average opponent
Adjusted Tempo Rank	Rank of the metric 'Adjusted Tempo', relative to all Division I teams.
Raw Tempo	Raw Tempo: the unadjusted, basic measure of how many possessions a team uses per 40 minutes
Raw Tempo Rank	Rank of the metric 'Raw Tempo', relative to all Division I teams.
Adjusted Offensive Efficiency	Adjusted Offensive Efficiency: points scored per 100 possessions adjusted for opponent quality.
Adjusted Offensive Efficiency Rank	Rank of the metric 'Adjusted Offensive Efficiency', relative to all Division I teams.
Raw Offensive Efficiency	Raw Offensive Efficiency: unadjusted points scored per 100 possessions.
Raw Offensive Efficiency Rank	Rank of the metric 'Raw Offensive Efficiency', relative to all Division I teams.
Adjusted Defensive Efficiency	Adjusted Defensive Efficiency: points allowed per 100 possessions adjusted for opponent quality.
Adjusted Defensive Efficiency Rank	Rank of the metric 'Adjusted Defensive Efficiency', relative to all Division I teams.
Raw Defensive Efficiency	Raw Defensive Efficiency: unadjusted points allowed per 100 possessions.
Raw Defensive Efficiency Rank	Rank of the metric 'Raw Defensive Efficiency', relative to all Division I teams.
Avg Possession Length (Offense)	Basketball performance or tournament metadata variable: Avg time of possesion (Offense).
Avg Possession Length (Offense) Rank	Rank of the metric 'Avg Possession Length (Offense)', relative to all Division I teams.
Avg Possession Length (Defense)	Basketball performance or tournament metadata variable: Avg time of possesion (Defense).
Avg Possession Length (Defense) Rank	Rank of the metric 'Avg Possession Length (Defense)', relative to all Division I teams.
eFGPct	Effective Field Goal Percentage: shooting efficiency weighted for 3-point value.
RankeFGPct	Rank of the metric 'eFGPct', relative to all Division I teams.
TOPct	Turnover Percentage: percent of possessions ending in turnovers.
RankTOPct	Rank of the metric 'TOPct', relative to all Division I teams.
ORPct	Offensive Rebound Percentage: percent of missed shots rebounded by the offense.
RankORPct	Rank of the metric 'ORPct', relative to all Division I teams.
FTRate	Free Throw Rate: free throw attempts per field goal attempt.
RankFTRate	Rank of the metric 'FTRate', relative to all Division I teams.
OffFT	Basketball performance or tournament metadata variable: Number of free throws shot
RankOffFT	Rank of the metric 'OffFT', relative to all Division I teams.
Off2PtFG	Two-Point Field Goal Percentage: accuracy on shots inside the arc.
RankOff2PtFG	Rank of the metric 'Off2PtFG', relative to all Division I teams.
Off3PtFG	Three-Point Field Goal Percentage: accuracy on shots beyond the arc.
RankOff3PtFG	Rank of the metric 'Off3PtFG', relative to all Division I teams.
DefFT	Basketball performance or tournament metadata variable: Number of free throws awarded to offensive team
RankDefFT	Rank of the metric 'DefFT', relative to all Division I teams.
Def2PtFG	Two-Point Field Goal Percentage: accuracy on shots inside the arc.
RankDef2PtFG	Rank of the metric 'Def2PtFG', relative to all Division I teams.
Def3PtFG	Three-Point Field Goal Percentage: accuracy on shots beyond the arc.
RankDef3PtFG	Rank of the metric 'Def3PtFG', relative to all Division I teams.
FTPct	Free Throw Percentage: accuracy from the free-throw line.
RankFTPct	Rank of the metric 'FTPct', relative to all Division I teams.
BlockPct	Block Percentage: percent of opponent shots blocked.
RankBlockPct	Rank of the metric 'BlockPct', relative to all Division I teams.
OppFG2Pct	Opponent 2-Point Percentage: defensive effectiveness inside the arc.
RankOppFG2Pct	Rank of the metric 'OppFG2Pct', relative to all Division I teams.
OppFG3Pct	Opponent 3-Point Percentage: defensive effectiveness against threes.
RankOppFG3Pct	Rank of the metric 'OppFG3Pct', relative to all Division I teams.
OppFTPct	Free Throw Percentage: accuracy from the free-throw line.
RankOppFTPct	Rank of the metric 'OppFTPct', relative to all Division I teams.
OppBlockPct	Opponent Block Percentage: percent of shots blocked by opponents.
RankOppBlockPct	Rank of the metric 'OppBlockPct', relative to all Division I teams.
FG3Rate	Three-Point Attempt Rate: proportion of total shots taken from three.
RankFG3Rate	Rank of the metric 'FG3Rate', relative to all Division I teams.
OppFG3Rate	Opponent 3-Point Percentage: defensive effectiveness against threes.
RankOppFG3Rate	Rank of the metric 'OppFG3Rate', relative to all Division I teams.
ARate	Assist Rate: percentage of field goals that are assisted.
RankARate	Rank of the metric 'ARate', relative to all Division I teams.
OppARate	Opponent Assist Rate: opponent share of assisted field goals.
RankOppARate	Rank of the metric 'OppARate', relative to all Division I teams.
StlRate	Steal Rate: fraction of defensive possessions ending in a steal.
RankStlRate	Rank of the metric 'StlRate', relative to all Division I teams.
OppStlRate	Opponent Steal Rate: rate at which opponents record steals.
RankOppStlRate	Rank of the metric 'OppStlRate', relative to all Division I teams.
NSTRate	Non-Steal Turnover Rate: turnovers committed that are not steals.
RankNSTRate	Rank of the metric 'NSTRate', relative to all Division I teams.
OppNSTRate	Opponent Non-Steal Turnover Rate.
RankOppNSTRate	Rank of the metric 'OppNSTRate', relative to all Division I teams.
AvgHeight	AvgHeight: average player height for that position or team.
RankAvgHeight	Rank of the metric 'AvgHeight', relative to all Division I teams.
CenterHeight	CenterHeight: average player height for that position or team.
RankCenterHeight	Rank of the metric 'CenterHeight', relative to all Division I teams.
PFHeight	PFHeight: average player height for that position or team.
RankPFHeight	Rank of the metric 'PFHeight', relative to all Division I teams.
SFHeight	SFHeight: average player height for that position or team.

RankSFHeight	Rank of the metric 'SFHeight', relative to all Division I teams.
SGHeight	SGHeight: average player height for that position or team.
RankSGHeight	Rank of the metric 'SGHeight', relative to all Division I teams.
PGHeight	PGHeight: average player height for that position or team.
RankPGHeight	Rank of the metric 'PGHeight', relative to all Division I teams.
EffectiveHeight	EffectiveHeight: average player height for that position or team.
RankEffectiveHeight	Rank of the metric 'EffectiveHeight', relative to all Division I teams.
Experience	Average years of experience weighted across lineups.
RankExperience	Rank of the metric 'Experience', relative to all Division I teams.
Bench	Bench usage: percentage of minutes played by non-starters.
BenchRank	Rank of the metric 'Bench', relative to all Division I teams.
CenterPts	Scoring contribution from the Center position group.
RankCenterPts	Rank of the metric 'CenterPts', relative to all Division I teams.
PFPts	Scoring contribution from the PF position group.
RankPFPts	Rank of the metric 'PFPts', relative to all Division I teams.
SFPts	Scoring contribution from the SF position group.
RankSFPts	Rank of the metric 'SFPts', relative to all Division I teams.
SGPts	Scoring contribution from the SG position group.
RankSGPts	Rank of the metric 'SGPts', relative to all Division I teams.
PGPts	Scoring contribution from the PG position group.
RankPGPts	Rank of the metric 'PGPts', relative to all Division I teams.
CenterOR	Center Offensive Rebounds.
RankCenterOR	Rank of the metric 'CenterOR', relative to all Division I teams.
PFOR	Basketball performance or tournament metadata variable: Offensive Rebound contribution from the PF group.
RankPFOR	Rank of the metric 'PFOR', relative to all Division I teams.
SGOR	Basketball performance or tournament metadata variable: Offensive Rebound contribution from the SG group.
RankSGOR	Rank of the metric 'SGOR', relative to all Division I teams.
PGOR	Basketball performance or tournament metadata variable: Offensive Rebound contribution from the PG group.
RankPGOR	Rank of the metric 'PGOR', relative to all Division I teams.
CenterDR	Center Defensive Rebounds.
RankCenterDR	Rank of the metric 'CenterDR', relative to all Division I teams.
PFDR	Basketball performance or tournament metadata variable: Defensive Rebound contribution from the PF group.
RankPFDR	Rank of the metric 'PFDR', relative to all Division I teams.
SGDR	Basketball performance or tournament metadata variable: Defensive Rebound contribution from the SG group.
RankSGDR	Rank of the metric 'SGDR', relative to all Division I teams.
PGDR	Basketball performance or tournament metadata variable: Defensive Rebound contribution from the PG group.
RankPGDR	Rank of the metric 'PGDR', relative to all Division I teams.
Net Rating	Net Rating: offensive rating minus defensive rating.
Net Rating Rank	Rank of the metric 'Net Rating', relative to all Division I teams.
Mapped Conference Name	Conference name the team belongs to.
team	Short team name identifier.
Current Coach	Current head coach and coaching tenure information.
Full Team Name	Basketball performance or tournament metadata variable: Full Team Name.
Since	Basketball performance or tournament metadata variable: Since.
Active Coaching Length	Current head coach and coaching tenure information.
Active Coaching Length Index	Current head coach and coaching tenure information.
Seed	Tournament seed (1–16).
Region	Tournament region assignment.
Post-Season Tournament	Postseason tournament classification (NCAA, NIT, etc.).
Post-Season Tournament Sorting Index	Postseason tournament classification (NCAA, NIT, etc.).
Vulnerable Top 2 Seed?	Indicator for whether a top-2 seed is considered upset-prone.
Tournament Winner?	Indicator for whether the team won the NCAA Tournament.
Tournament Championship?	Basketball performance or tournament metadata variable: Tournament Championship?.
Final Four?	Indicator for whether the team reached the Final Four.
Top 12 in AP Top 25 During Week 6?	Basketball performance or tournament metadata variable: Top 12 in AP Top 25 During Week 6?.
BY YEAR NO	Basketball performance or tournament metadata variable: BY YEAR NO.
BY ROUND NO	Tournament round information.
TEAM NO	Basketball performance or tournament metadata variable: TEAM NO.
SEED	Tournament seed (1–16).
ROUND	Tournament round information.
CURRENT ROUND	Tournament round information.
SCORE	Points scored by the team in the game.
round_label	Tournament round information.