Predicting Early NBA Career Survivability Using Pre-Draft College Statistics

Christian Ferdinand Wiethüchter

Dissertation written under the supervision of professor Nicolò Bertani

Dissertation submitted in partial fulfilment of requirements for the MSc in Business Analytics, at the Universidade Católica Portuguesa, 11.09.2025.

Abstract

This study examines whether publicly available college basketball statistics can predict early NBA career outcomes, a period both formative for players and financially crucial for franchises under rookie-scale contracts. A dataset was assembled covering all drafted college players from 2002–2019, combining season-level college performance with subsequent NBA results. Because most prospects play multiple seasons before declaring for the draft, season-level predictions were systematically aggregated to the player level to reflect the actual unit of draft-day decision-making.

Five outcome labels captured increasing levels of early-career success: surviving the rookie contract, securing rotation or starter roles, and surpassing minimum impact thresholds in Win Shares and Value Over Replacement Player. An end-to-end pipeline embedded preprocessing, grouped cross-validation, and fold-safe aggregation, ensuring reproducibility and preventing information leakage. Regularized linear models served as baselines, while Random Forests and Gradient Boosted Trees benchmarked non-linear performance.

The results show that college box-score data contains predictive signal, though modest in strength for most labels. Tree-based methods outperformed linear models: Random Forests were strongest for durability-oriented outcomes such as rotation roles and four-year survival, while boosting captured rarer ceiling outcomes like starters and high-impact contributors. Aggregation to the player level proved essential, with simple averaging often sufficient. Feature importance highlighted class year, games played, assists, and shooting efficiency as consistent though limited predictors.

While box scores alone cannot identify future stars with high precision, they provide a systematic, reproducible baseline that helps reduce draft risk by flagging players most likely to contribute early.

Keywords: NBA Draft, College Basketball Statistics, Machine Learning, Sports Analytics, Player Aggregation, Early Career Prediction

Title: Predicting Early NBA Career Survivability Using Pre-Draft College Statistics

Author: Christian Ferdinand Wiethüchter

Resumo

Este estudo investiga se estatísticas públicas do basquetebol universitário podem prever os resultados iniciais de carreira na NBA, um período formativo para os jogadores e financeiramente crucial para as franquias devido aos contratos de *rookie*. Foi construído um conjunto de dados que abrange todos os jogadores universitários selecionados no draft entre 2002 e 2019, combinando desempenho por temporada na universidade com os resultados subsequentes na NBA. Como a maioria dos candidatos joga várias temporadas antes de se declarar para o draft, as previsões ao nível da temporada foram agregadas ao nível do jogador, refletindo a unidade real de decisão no dia do draft.

Foram definidos cinco rótulos de sucesso inicial: sobrevivência ao contrato de *rookie*, papéis de rotação ou de titular, e mínimos em métricas avançadas como *Win Shares* e *Value Over Replacement Player*. Um pipeline completo incluiu pré-processamento, validação cruzada agrupada e agregação sem fuga de informação, garantindo reprodutibilidade. Modelos lineares regularizados serviram como linha de base, enquanto Random Forests e *Gradient Boosted Trees* representaram abordagens não lineares.

Os resultados mostram que as estatísticas universitárias contêm sinal preditivo, ainda que modesto. Métodos baseados em árvores superaram os lineares: Random Forests foram mais eficazes em resultados de durabilidade, enquanto métodos de *boosting* captaram melhor papéis de titular e impacto elevado. A agregação ao nível do jogador revelou-se essencial, muitas vezes bastando uma média simples. Embora estatísticas de *box score* não identifiquem futuros astros com alta precisão, fornecem uma base sistemática que ajuda a reduzir riscos de draft, sinalizando jogadores com maior probabilidade de contribuir cedo.

Palavras-chave: Draft da NBA, Estatísticas do Basquetebol Universitário, Aprendizado de Máquina, Sports Analytics, Random Forests, Gradient Boosted Trees, Agregação de Jogadores, Previsão de Carreira Inicial

Título: Draft da NBA, Estatísticas do Basquetebol Universitário, Aprendizado de Máquina, Análise de Desempenho Esportivo, Agregação de Jogadores, Previsão de Carreira Inicial

Autor: Christian Ferdinand Wiethüchter

Acknowledgements

First, I would like to thank my professors at the Business Analytics Masters Program at Catolica Lisbon SBE, for all they have taught us. Writing this thesis has made me truly appreciate the depth of knowledge and skills I have gained over the past two years.

I am especially grateful to Professor Filipa Reis, Professor Nicolò Bertani, and Professor Miguel Godinho de Matos for the passion they brought to the classroom every day, for their dedication to our learning as well as challenging us to prove ourselves on their high academic standards.

A special thank you to my thesis supervisor, Professor Nicolò Bertani. His support was invaluable whenever I encountered complications, and his trust in my ability to push forward made this entire undertaking possible.

I also have to thank my mother, who, for 2 years listened to all my problems on the phone or otherwise, steadfastly believing into my capability to solve them.

Finally, I want to thank Lisbon and its people. This city has been an amazing place to live and learn among people that have become some of my best friends.

Contents

Abstract i

Resumo ii

Acknowledgements iii

Table of Contents iv

List of Figures v

List of Tables vi

List of Abbreviations vii

1 Introduction 1

2 Literature Review 3

2.1 The Draft 3

2.2 Systematic Errors in Talent Evaluation 4

2.3 Analytics in Sports 4

2.4 Predictive Modelling and Player Success 5

3 Data 8

3.1 Data Sources and Description 8

3.2 Data Preparation 9

3.3 Final Data Description 10

4 Methodology 14

4.1 Feature Engineering 14

4.2 Label Engineering 15

4.3 Model Family Selection 17

4.4 Training and Validation 18

4.5 Aggregating to the Player Level 19

4.6 Evaluation Strategy 19

5 Results 21

5.1 Model Performance 21

5.2 Aggregation 22

5.3 Confusion Matrices 23

5.4 Feature Importance 24

5.5 Discussion of Key Results 25

6 Conclusion 28

6.1 Summary 28

6.2 Limitations 29

6.3 Future Work 29

Appendix A Additional Figures 32

Appendix B Code Excerpts 33

List of Figures

1 *Data pipeline.* . . . 10
2 *Drafted per year and matched share.* . . . 10
3 *Roster composition.* . . . 11
4 *3PA trend by draft year.* . . . 12
5 *College scoring and shooting.* . . . 12
6 *NBA career outcomes.* . . . 13
7 *Win Shares distributions.* . . . 17
8 *VORP distributions.* . . . 17
9 *Precision–Recall curve for Rotation (test set).* . . . 23
10 *Confusion matrices of winning models (test set).* . . . 24
A.1 *Sports-Reference player page.* . . . 32

List of Tables

1 *Feature groups and engineering treatments.* 15
2 *Engineered NBA outcome labels.* 16
3 *Boosted vs. Random Forest on all labels (test set).* 23
4 *Permutation feature importance for Rotation (test set).* 25

List of Abbreviations

Abbreviation	Meaning
BAA	Basketball Association of America
NBA	National Basketball Association
VORP	Value Over Replacement Player
WS	Win Shares
LightGBM	Light Gradient Boosting Machine
SVM	Support Vector Machine
SHAP	SHapley Additive exPlanations
LIWC-22	Linguistic Inquiry and Word Count, 2022 version
ROC-AUC	Receiver Operating Characteristic – Area Under the Curve
3PA	Three-Point Attempts
BPM	Box Plus/Minus
VIF	Variance Inflation Factor
OLS	Ordinary Least Squares
PR-AUC	Precision–Recall Area Under the Curve
XGBoost	Extreme Gradient Boosting
CatBoost	Categorical Boosting
PFI	Permutation Feature Importance
EPM	Estimated Plus-Minus
RAPM	Regularized Adjusted Plus-Minus
WAR	Wins Above Replacement

1 Introduction

When the BAA (Basketball Association of America), the predecessor to today's NBA (National Basketball Association), held the first draft in 1947, it was not foreseeable what kind of spectacle and media coverage the event would one day attract. What started as an industry meeting between professionals in a Detroit hotel has grown into an event held in a packed NBA stadium, becoming one of the largest televised events on the U.S. sports calendar. Yet the underlying idea has remained the same. Because youth sports in the United States are largely organized at the school and college level, professional franchises don't operate their own youth development systems. Therefore the draft serves as the primary entry mechanism for new talent, allocating the top picks each year to the weakest teams in order to maintain competitive balance.

The highly orchestrated nature of the draft process, however, contrasts with the high degree of uncertainty of career outcomes. Many highly regarded college players go on to disappoint in the NBA, which points to persistent difficulties in evaluating talent, despite the professionalization of U.S. college sports. And whereas the draft outcomes or first year performance of players have been studies to some degree, little has been done to explore player success over the first 4 years, which are the most financially crucial for any franchise, with highest relevance to rebuilding teams.

In order to close this gap, an approach was developed that builds directly on the structure of available data and the requirements of decision-making on draft day. College basketball statistics were chosen as the foundation, since they are consistently recorded across players and readily available to teams. To predict early professional success, outcome measures were defined that capture both durability, whether players remain in the league and contribute consistent minutes—and upside, reflected in impact metrics such as VORP (Value Over Replacement Player) and WS (Win Shares). The multi season nature of most college careers required that modelling account for multiple seasons per player, leading to the use of grouped splits and later aggregation to the player level. From these considerations, a modelling framework was designed that combines regularised linear models for interpretability with tree ensembles for capturing non-linear patterns, embedded in pipelines that ensure reproducibility and guard against information leakage.

The analysis shows that college statistics contain predictive information that can be used to improve the evaluation of prospects. While linear models offered limited predictive power, tree-based methods delivered consistent gains, with boosted ensembles strongest at the season level and Random Forests particularly effective once predictions were aggregated to the player level. Reliable signal was found for outcomes that capture immediate contribution (`rotation`) and medium-term durability (`survival`), both best modelled with Random Forests. Indicators of upside, such as `impact4_vorp`, produced usable but less stable predictions, with boosted models performing best. The remaining labels, `starter` and `impact4_ws`, achieved only limited reliability, with thresholded performance in particular revealing their weaknesses. Overall,

these results confirm that box-score data, though not sufficient for high-precision forecasts, provide a meaningful baseline that extends the scope of prior work and can inform draft decisions in a systematic way.

The remainder of this thesis is organized as follows. Chapter 2 reviews the relevant literature on draft prediction and related work. Chapter 3 describes the data sources and the preparation steps taken to construct the analytical dataset. Chapter 4 outlines the methodology, including the feature engineering and modeling approaches. Chapter 5 presents the results, while Chapter 6 concludes with a discussion of implications, limitations, and directions for future research.

2 Literature Review

2.1 The Draft

It is assumed that the reader is familiar with the basic rules of basketball, a five-on-five team sport played over four 12-minute quarters, with unlimited substitutions and scoring through 1-, 2-, or 3-point field goals or free throws. The draft system however has its own unique rules and processes. As this mechanism plays the most important part in shaping team composure and success, it is important to understand both how it works and shapes team strategies, as well as what research has been done into to decisions and team and player outcomes of the draft.

While the top teams of each NBA season enter the playoffs to compete for the championship, the 14 teams that did not make the cut see their season end, and make up the bottom 14 of the league's 30 teams. These teams then enter the draft lottery, which distributes the first 14 of 60 total picks in the draft through a weighted system, according to reverse final placement in the league. The worst teams get the highest chances for the very best picks, although even the 14th-worst team still has a slight chance of receiving the first pick. This setup is intended to discourage teams from losing on purpose toward the end of the season (”tanking”), in order to secure one of the very top draft positions.

The draft pool is mainly comprised of first- to fourth- (and final) year college players, along with a small number of international players and, in rare cases, players directly entering the draft from high school. On draft day, all teams select their players across two rounds of 30 picks each.

The structure of rookie contracts in the NBA adds an additional economic dimension to the draft process. Salaries for first-year players are not individually negotiated but are fixed by the league and tied directly to the draft position at which a player is selected. These contracts usually run for 2 years fixed, plus an option for two more years which the holder of the rookie contract can execute at will. This leaves decision-making power almost entirely in the hands of the franchise, which is also always allowed to trade a contract away, leaving the player with little influence over salary or mobility during this period. For the teams, every draft pick therefore represents not only a decision on the immediate player impact on the team, but also a fixed financial committent over a predetermined period of time.

This means, that if a drafted player exceeds expectations, the added value generated for the team during this contract period is far greater than the salary paid. Since the NBA is regulated by a salary cap on a team level, such surplus value is of particular importance, as it allows competitive teams to balance expensive star contracts with productive yet inexpensive rookie-scale deals. For rebuilding teams, on the other hand, players on a rookie contract serve as strategic assets, they may be retained as long-term cornerstones or traded for established veterans and additional draft picks, depending on the team's situation. In either case, the utility of a draft pick depends not only on immediate rookie-year performance but also on whether a player can deliver consistently across the full four-year contract window. This is especially true since the first draft picks tend to be executed by the worst teams, which depend more heavily on the mak-

ing the correct choice. Despite the central role of this perspective in roster construction and financial planning, systematic analysis of medium-term success among draft entrants remains limited.

2.2 Systematic Errors in Talent Evaluation

For one, there always remain some elements of human decision making, showing the biases which come with any such decision, especially in an environment, which traditionally has relied heavily on the subjective ”feeling” of a scout when evaluating a player. Scouts still tend to overvalue factors such as total points, player size and the pedigree of the college conference (league) the player is playing in, while undervaluing less obvious factors such as efficiency and ball control (Sailofsky, 2018). This is likely due to an availability bias, where easily observable attribute are overvalued even of the observer supposedly knows better.

Additionally, teams and scouts remain heavily influenced by the early mock draft rankings, which are shared in the media in the months leading up to the actual draft. Decisions makers tend to overvalue these mock draft rankings, even if more recent analyses and data point into another direction, showing some anchoring bias which directly influences draft day decision making (Ichniowski & Preston, 2021).

There also seems to be a tendency to evaluate player performance unequally once they have been acquired via a draft pick. Players with a high draft number tend to be evaluated more graciously while also given more chances to prove themselves. This sunk cost effects leads to an overvaluation of the players draft number, independent of actual on field productivity (Miguel et al., 2019)

These factors together show, that there remains a significant degree of heuristic-induced bias in team decision-making, where visible but often misleading signals such as scoring volume or early draft reputation outweigh more reliable indicators of future performance.

2.3 Analytics in Sports

In order to challenge these biases in decision-making, sports analytics emerged as an attempt to replace intuition with data, reshaping professional basketball and player evaluation in particular. How spectators consume the game, how it is presented, and how the industry operates has undergone a profound transformation driven by this rise of analytics. Its roots lie in sabermetrics, a term coined by baseball analysts in the late 20th century and popularized through the book and film Moneyball, which demonstrated that objective statistical analysis could uncover player value often overlooked by traditional scouting methods. The success of this approach showed that data could provide a significant competitive advantage, prompting other major sports, including basketball, to fully embrace analytics.

This philosophical shift quickly spread to basketball, revolutionizing team strategy and player evaluation. Modern basketball analytics complements traditional box score numbers

(like points or rebounds) with more sophisticated metrics to capture the finer details of the game. Professional leagues, such as the NBA, have invested heavily in player-tracking technologies like SportVU, which uses cameras to generate granular data on player and ball movements (NBA Communications, 2016). This has enabled the creation of advanced analytics that quantify shot efficiency by location, defensive impact, and overall player value. For example, analytics helped popularize the modern ”pace-and-space” offense by demonstrating the high-efficiency returns of shots taken near the rim or from the three-point line, a key strategic insight derived from analyzing a wide range of on-court actions (Rincon, 2024).

This increasing availability of data has made a data-driven approach increasingly relevant to all aspects of the sport, including player evaluation for the draft. However, the wealth of rich data available for professionals is not yet fully mirrored at the college level. This introduces a key ambiguity. while machine learning models can now predict outcomes with high accuracy using extensive professional data, the challenge for research at the collegiate level is to determine which of the more limited, publicly available metrics, including fundamental box score data, are the most reliable predictors of future success.

2.4 Predictive Modelling and Player Success

Against this backdrop, an increasing strand of research has turned to systematic modelling approaches, aiming to predict the effects surrounding the draft.

Patton et al. (2020) showed quite impressively the power of large-scale data analytics for predicting whether a player would make it to the NBA. Going beyond college box scores, they used a dataset comprised of ball- and player-tracking data from college games, spanning over 650,000 possessions to capture the actual plays being performed on the court. They trained a LightGBM (Light Gradient Boosting Machine) model both on box score data as well as on their tracking-derived features. Their results showed a 20% improvement over the sole use of box score data.

They demonstrated the added value of richer input data for predictive modelling, particularly when enriching box scores with actions that directly relate to game outcomes. There are, however, practical limitations to this approach, as video tracking is far from being available for every game, and building the required infrastructure is costly. Nevertheless, they established that innovating with data and moving beyond the box score can improve predictions in the basketball space, even though their models were limited to forecasting whether or not a player would be drafted, and did not move beyond that.

Others have tried to solve the same prediction task by using more of the tools of machine learning.

de Araújo Costa et al. (2024) used season level college data to examine the potential of interpretable machine learning models for predicting whether college players would reach the NBA. Using data from 2009 to 2021, their dataset contained over 65,000 player-game records

and a wide range of features, including traditional box score statistics and advanced efficiency metrics. They compared several modelling approaches, such as decision trees, logistic regression, support vector machines, and neural networks, with a particular emphasis on maintaining transparency and interpretability.

To improve predictive accuracy and reduce dimensionality, they combined these models with a genetic algorithm for feature selection. Their results showed that logistic regression paired with the genetic algorithm achieved the strongest balance, reaching an accuracy of around 82%. Importantly, the study demonstrated that interpretable white-box models performed on par with more complex black-box methods such as SVM (Support Vector Machine) and neural networks, suggesting that higher complexity does not necessarily translate into better predictions in this setting.

They further used SHAP (SHapley Additive exPlanations) values to highlight which attributes contributed most strongly to predictive performance. Advanced efficiency and possession-adjusted statistics consistently emerged as key predictors, while raw scoring totals played a less important role. This seems to support the view of (Sailofsky, 2018) that teams often overvalue visible output while undervaluing underlying efficiency. As a limitation, their study relied exclusively on college game performance data and did not integrate fitness or psychological measures. Nevertheless, de Araújo Costa et al. (2024) established that interpretable machine learning can determine whether a college player would make it to the NBA with an accuracy high enough to be useful to decision makers.

Farrell et al. (2025) combined both approaches, by integrating psychological measures into models of draft success. Drawing on college player interviews from 1992 to 2024, they applied the LIWC-22 (Linguistic Inquiry and Word Count, 2022 version) tool to extract linguistic indicators of cognitive, emotional, and social traits, which were then combined with college box score statistics and physical attributes. This approach made it possible to incorporate aspects of mindset and communication style into predictive models that traditionally rely only on performance data.

Their models showed that psychological features on their own achieved modest accuracy, but when combined with college and physical data, entry prediction rose to around 87%. Importantly, their results highlighted that players who expressed themselves in more present- and future-focused language, and who used less complex sentence structures, were more likely to reach and remain in the NBA.

While Farrell et al. (2025) extended prediction by adding psychological dimensions to traditional performance metrics, Foutzopoulos et al. (2025) shifted the focus further along the career timeline, examining how early professional performance in a player's sophomore season relates to the likelihood of sustaining a ten-year NBA career, which qualifies players for the full NBA-retirement scheme. Using draft cohorts from 1999 to 2006, and validating their models on players drafted in 2013–2014, they investigated which early professional signals were most indicative of sustained longevity.

The authors employed ridge logistic regression to address multicollinearity among sophomore box score features such as minutes, points, rebounds, assists, steals, and age. Their best models achieved an ROC-AUC (Receiver Operating Characteristic – Area Under the Curve) of around 0.79, suggesting a reasonably strong ability to distinguish between players who would retire earlier and those who would last a decade or more. Importantly, minutes played and overall usage emerged as the strongest predictors, alongside basic production measures like scoring and rebounding.

While their study was limited to sophomore-year NBA data rather than pre-draft or college performance, it demonstrated that early-career playing time and usage patterns can provide meaningful signals of long-term survival.

Taken together, the existing literature demonstrates a good understanding in predicting NBA player success. Studies have shown that draft decisions are affected by systematic biases, that richer data sources and machine learning methods can improve predictions, that psychological dimensions add explanatory power beyond performance metrics, and that early professional statistics can be used to forecast very long careers. The focus of previous work, however, has largely been to predict whether a player makes it to the NBA, which has been well studied, as well as either very short-term or very long-term success metrics. While this is interesting and important in its own right, the likely impact a player has on his team in the first four years, when his value is bound to a team at a fixed price, remains largely unexplored.

This thesis focuses on this rookie-scale contract horizon and examines player contribution across the first four years in the league, by answering the question: *To what extent can college basketball statistics predict player success during the first four years of an NBA career?* Little attention has been paid to this intermediate window, yet this period is of particular interest because it combines two elements. On the one hand, it often marks the decisive phase of a player's development and integration into the league. On the other hand, it is economically unique, as player salaries and contracts are fixed and thus create the potential for substantial surplus value. By centring on this time frame, the study seeks to extend on existing research and provide decision-makers with evidence that is directly aligned with the realities of roster construction and salary-cap management.

3 Data

3.1 Data Sources and Description

To answer the question of player outcomes over their first few career years one needs to obtain sufficient data on both the players' pre-draft performance as well as their NBA career development. This chapter describes the data used, how it was obtained and how it was prepared for analysis.

The aim for the final dataset is to combine information on players' college basketball performance as well as their subsequent NBA careers. In order to have full control of scope, quality, and the understanding of the material, the data collection process was developed end-to-end, relying on original sourcing wherever possible. This approach also provides full transparency on how the data was obtained and prepared.

The main source of data is the Sports Reference network of websites. Whereas the NBA's official platform, NBA Stats, publishes quite extensive data on current professional players, it does not provide any information on college performance. By contrast, Basketball Reference (Sports Reference LLC, 2025a) and its sister site Sports Reference College Basketball (Sports Reference LLC, 2025b), the network's college basketball hub, provide detailed information on current and past players' full careers. An exemplary screenshot of how a sports reference player page looks can be found in the appendix (Figure A.1). These sites, with their official aim to democratise data, are well known among enthusiasts and have served as a starting point for prior research, as in Miguel et al. (2019).

This work focuses on the draft cohorts of 2002 to 2019, inclusive. The upper bound was chosen so each player could have had the opportunity to establish themselves in the NBA for at least five full seasons. The lower bound was chosen since Sports Reference provides a near-complete track record of college players starting with the 1998 season, giving a full history for every college player who declared for the 2002 draft, including seniors. These 18 drafts are comprised of 1,074 players in total, at an average of 59.66 per draft. Of these 1,074, 730 could be matched to college player data, resulting in 2,381 rows of season-level observations. The remaining 32% of players could not be matched to college players. They either came from international backgrounds or entered directly from high school and therefore lacked college statistics. This proportion is about what is expected from draft composition in the past 25 years.

The data were collected through a series of custom-built web scrapers written in Python, using the requests and beautiful soup packages. In the first step, complete draft lists were scraped from basketball reference for each year. From there, the name was extracted to build the URL for the individual player sites, from which the career statistics were obtained. Afterwards, another web scraper was set up to repeat the same process for the players' per-season college data from Sports Reference. This, however, proved challenging, as the large-scale automated queries triggered an automated Cloudflare protection which blocked conventional scraping methods. After an iterative process, the issue was resolved using the Cloudscraper package, which was able to

bypass the JavaScript-based protection with careful request handling and additional delays.

The outcome of this data sourcing procedure was two datasets, which combined give the full picture of the current data on the college and NBA careers of the 2002 to 2019 drafted college athletes.

After an initial round of modelling, a second, smaller dataset was collected containing players' height and weight at draft. These data were also scraped from sports-reference.com/cbb and stored with a new, second version of the data in order to explore its impact on modelling. Moving forward the two datasets will be called "Pre" and "Post".

3.2 Data Preparation

Since the data we obtained was scraped directly from the HTML, it was raw and had to be cleaned and prepared extensively before being suitable for further usage.

The first stage of preparation focused on filtering and restructuring. The scraped college dataset contained rows that did not represent single seasons, such as career summaries, multi-year aggregates, placeholders for transfer years or other scraping artefacts. These were removed so that only valid season-level observations remained, and implausible values, such as rows mistakenly attributed to earlier players with the same name, were dropped or marked as missing.

The second stage addressed standardisation. Variable names were harmonised, numeric fields converted into consistent types, and categorical information such as team, conference, position, and class year brought into a consistent format. An identifier was added to make each player-season uniquely recognisable.

The final stage was the integration across sources. College season data and NBA career summaries were merged into a single dataset, year-by-year totals and the matched share with the college sample are shown in *Figure 2: Drafted per year and matched share*. It has to be added that this data preparation was not a one-off exercise but required successive passes, with issues discovered during exploratory checks leading to refinements. Intermediate versions of the dataset were preserved to ensure reproducibility. The final data is internally consistent and suitable for descriptive analysis. The full preprocessing and integration workflow is shown in *Figure 1: Data pipeline*.

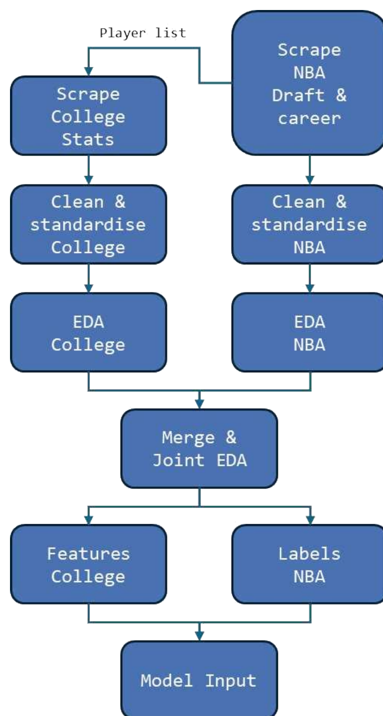


Figure 1: *Data pipeline.*

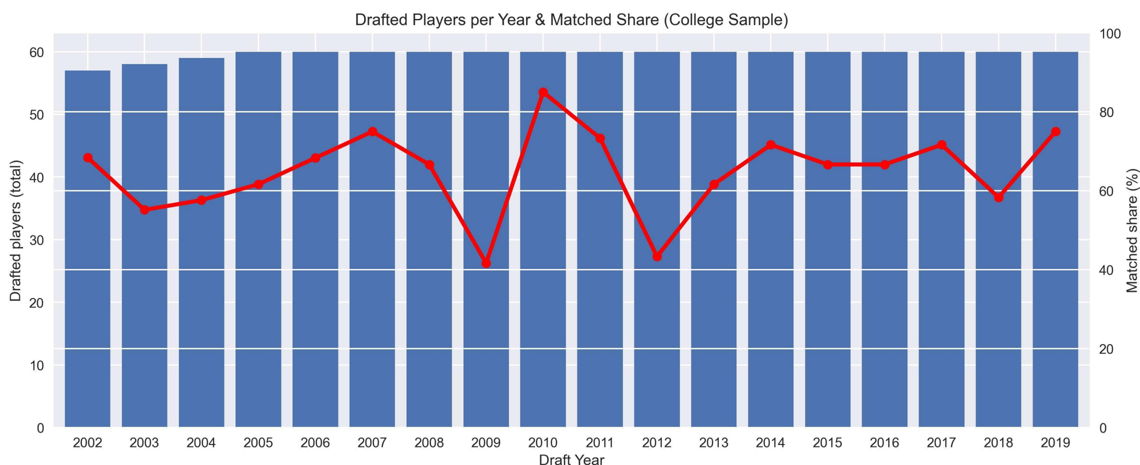


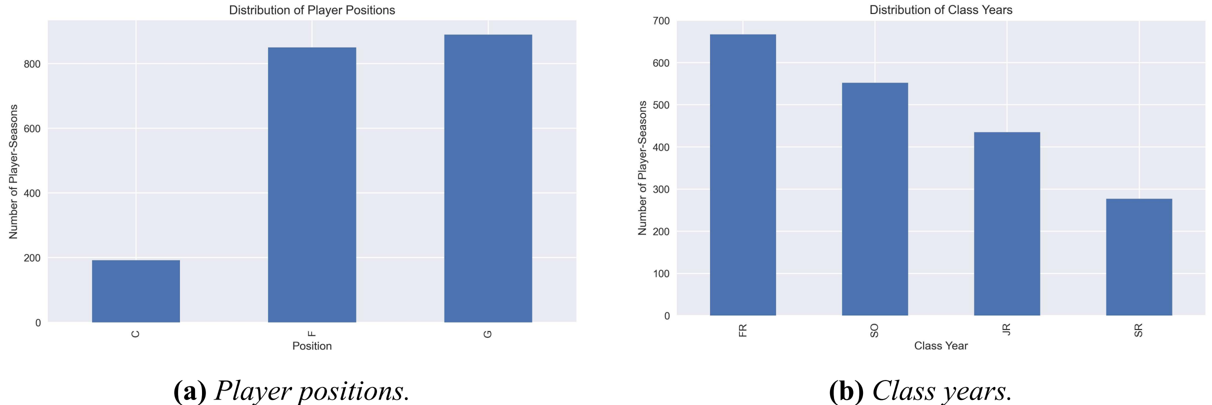
Figure 2: *Drafted per year and matched share.*

3.3 Final Data Description

The final cleaned dataset contains 1,931 rows, representing the college season data of 695 individual players. Each row corresponds to a single college player's performance in a single season, as well as his NBA statistics on a career level. This allows a comprehensive view of both pre-draft performance and total career outcomes. The variables in the dataset fall into one of three groups. Identifiers, including player name, season year, college team, conference, class year, position and draft year, or binary award indicator. Next, college statistics cover standard box score and shooting measures on a per-game basis, such as points, rebounds, assists, field goal

percentage, three-point percentage, and free-throw percentage. The NBA outcomes include variables such as years played, games played, Win Shares, as well as a binary indicator of whether the player ever appeared in the league, as a small but noticeable percentage of drafted players do not actually appear in an NBA game.

Upon first inspection of the data, some patterns are quite clearly visible. Guards are more active on the perimeter, recording higher assist counts and more three-point attempts, while forwards and centres have the best rebounding and field goal percentages. Seniors typically show higher per-game averages than underclassmen, reflecting both development during the college years as well as the tendency of elite players declaring for the draft after one or two seasons; see *Figure 3a: Player positions* and *Figure 3b: Class years*.



(a) *Player positions.* **(b)** *Class years.*

Figure 3: *Roster composition.*

Across draft years, there is a clear upward trend in three-point attempts, accompanied by a modest rise in three-point accuracy as well as a gradual decline in rebounding averages. This is very much in line with changes in the game observed during the time frame of our draft years, a pattern which shows what has been described in the literature as the three-point revolution (Levin et al., 2024); see *Figure 4: 3PA trend by draft year*.

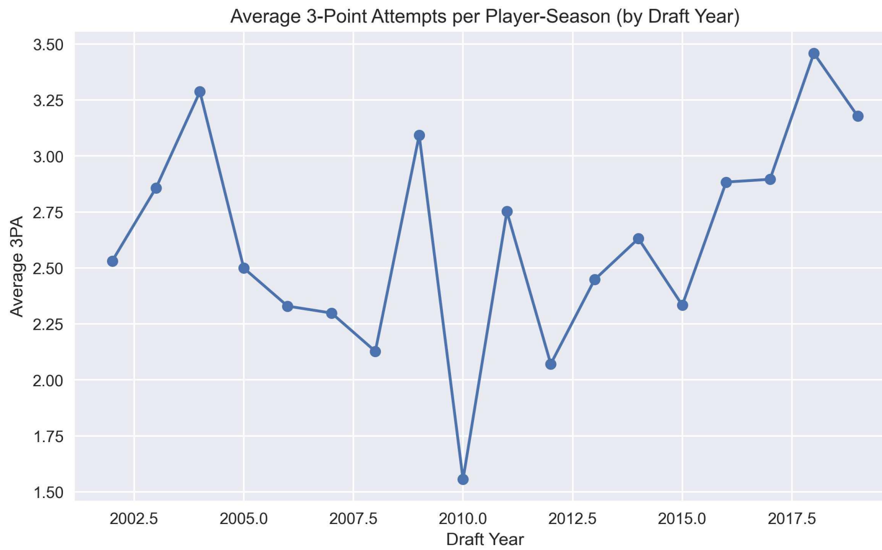
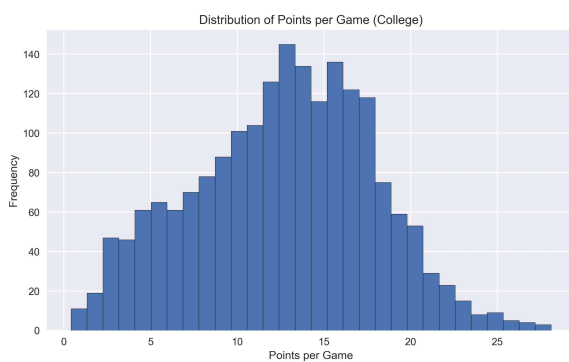
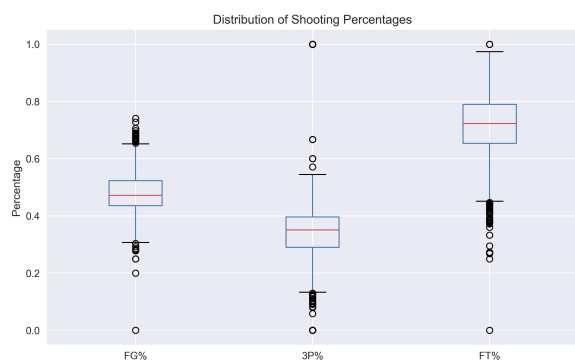


Figure 4: *3PA trend by draft year.*

The skewed distributions also highlight the presence of a small number of star players alongside a majority of more modest contributors. Counting statistics such as points, rebounds, and assists are strongly right-skewed, with a majority of players performing only modestly and a long tail of elite players; see *Figure 5a: College points per game*. Shooting percentages are more symmetrically distributed than counting statistics and they cluster around league averages; see *Figure 5b: Shooting percentages*. Differences across player roles are clearly visible, with centres and forwards recording higher overall efficiency due to their proximity to the basket, while guards exhibit lower field goal percentages but greater variability in three-point accuracy. Free-throw shooting is the most stable of the three measures, showing a relatively tight distribution across players and a gradual improvement over time.



(a) *Points per game.*

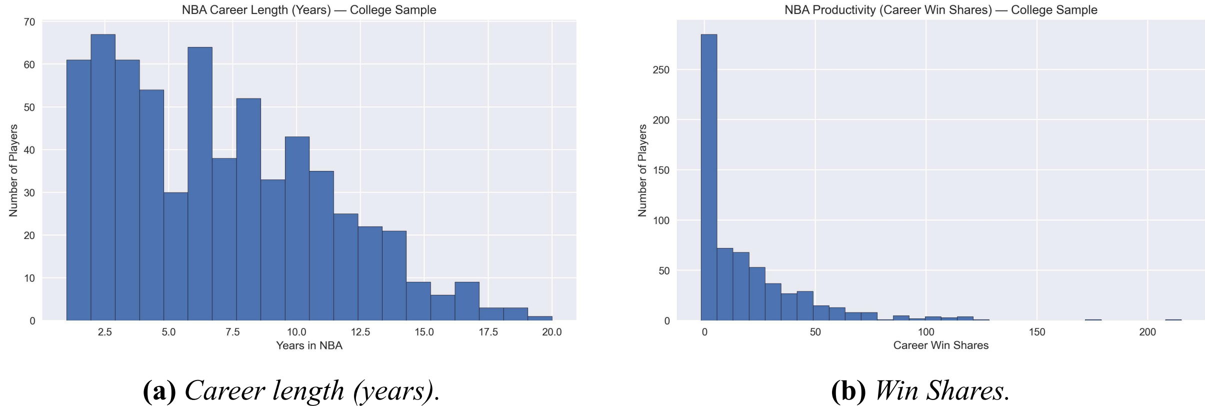


(b) *Shooting percentages (FG%, 3P%, FT%).*

Figure 5: *College scoring and shooting.*

NBA career outcomes again show a skewed pattern. Most players appear in the league for only a few seasons, while a smaller group achieve very long careers. Games played and Win Shares follow the same shape, with modest totals for the majority and a long tail of highly productive players; see *Figure 6a: NBA career length* and *Figure 6b: NBA Win Shares*. BPM (Box

Plus/Minus), an advanced box score metric that estimates a player's overall impact relative to league average, is distributed more evenly around zero, with many replacement-level contributors and a smaller set of players showing strongly positive or negative impacts.



(a) *Career length (years).*

(b) *Win Shares.*

Figure 6: *NBA career outcomes.*

The patterns described here establish the context for later analysis. They highlight how the data reflect known differences between positions and class years, as well as broader trends in basketball over time. While the dataset is already consistent and informative, minor refinements may still be applied as the project develops.

4 Methodology

4.1 Feature Engineering

In preparing the data for analysis, several transformations called feature engineering were required to ensure that the resulting features were suitable for use in predictive models. The main objectives of the feature preparation were to reduce redundancy, make players comparable despite differences in minutes or roles, and to translate categorical information into a form usable for analysis. The transformations were aimed at increasing interpretability, while at the same time avoiding any leakage of NBA outcomes into the predictor set.

Several steps were taken to achieve this. Redundant raw totals were removed in favour of rate- or percentage-based measures, and the main counting statistics were normalised to per-40 values (40 min is equivalent to one full 20 min + 20 min college game) to account for varying playing time. Shooting percentages were kept but filtered by minimum-attempt thresholds, in order to avoid misleading values caused by very small samples. In addition, a number of efficiency and style indicators were created, such as assist-to-turnover ratio, free-throw rate, and three-point attempt share. These are known as ”advanced statistics” in basketball. Categorical variables, including playing position, class year, and awards, were represented through one-hot encoding.

Multicollinearity was monitored by means of correlation checks and VIF (Variance Inflation Factor). In cases of perfect or near-perfect overlap, only one variable was kept. Adjustments for skewness, such as scaling or log transformations, were not carried out at this stage but postponed to the modelling pipeline, to ensure they were fitted only on training data. For the post-expanded dataset, player height (in cm) and weight (in kg) at draft were also included as raw features, requiring no further transformation.

Table 1: *Feature groups and engineering treatments.*

Feature group	Treatment
Counting statistics	Converted to per-40 values; raw totals dropped to ensure comparability across players
Shooting percentages	Retained only when based on sufficient attempts (3PA $\geq$ 50, 2PA $\geq$ 100, FTA $\geq$ 50); values below thresholds set to missing
Derived ratios	Assist-to-turnover ratio, free-throw rate (FTA/FGA), and three-point attempt share (3PA/FGA) constructed; overlapping rebound shares reduced to offensive rebound share only
Categorical variables	Encoded using one-hot dummies (positions G/F/C, class FR–SR, award flag); reference categories dropped to prevent multicollinearity
Anthropometrics	Height (cm) and weight (kg) at draft included as raw features without further transformation
Multicollinearity	Pairwise correlation and VIF checks; redundant variables (e.g., duplicate totals, DRB share) removed
Deferred transformations	Scaling and log transforms postponed to modelling pipeline to prevent data leakage

4.2 Label Engineering

Having constructed a set of features that enable prediction, the next step was to define the outcome variables, in line with our goal to predict the performance of NBA players during the first crucial years of their career. To prevent any leakage from their college careers, outcome labels were constructed from NBA career data only. The aim was to capture both the productivity of the career and its length, in a way that answers our question of early career success. For this purpose, originally three binary and two continuous measures were created.

The three binary measures were designed in a hierarchical and nested way, ranging from moderate to serious achievement. The first binary label was a survival indicator ”`survival`”, set to one if a player played four or more seasons in the NBA, meaning he survived his full rookie contract span. The second, a “`rotation`” label, was introduced for players who survived the full four season as well as averaged at least 15 minutes per game, indicating that they had found a solid role in their team. The third, a “`starter`” label, was applied to players who survived the full four season as well as averaged at least 25 minutes per game, indicating a regular starting role with substantial playtime. These definitions are summarised below in Table 2.

In addition, two continuous measures were constructed to capture performance across the

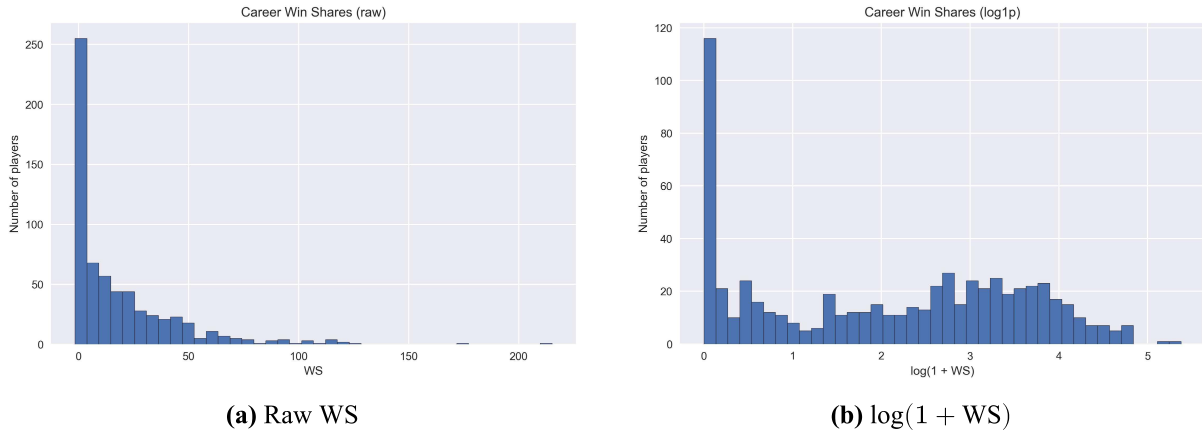
career. Career WS (Win Shares) is an advanced statistic estimating a player's total contribution to wins based on box-score statistics, always zero or larger. VORP (Value Over Replacement Player), by contrast, reflects contributions relative to a replacement-level baseline, with the baseline set to zero. Both measures are heavily right-skewed, as a small number of elite players with very long careers accumulate extremely high values of WS and VORP, the distributional shape and the effect of the transformations are illustrated in *Figure 7: Win Shares distributions* and *Figure 8: VORP distributions*.

However, upon first inspection the two regressors were dropped due to poor performance. Instead, two new binary labels were added "`impact4_ws`" and "`impact4_vorp`", which each combined the 4 year minimum survival with a WS minimum of 15, and a VORP of 1. These additions ensure that both durability and minimum impact are encoded directly in the binary targets.

The final dataset therefore contains three binary indicators of career survival at different levels, as well as two continuous measures of impact, raw and logged. This results in five distinct but complementary outcome variables. All five were retained for the initial modelling, with the option to narrow the focus at a later point.

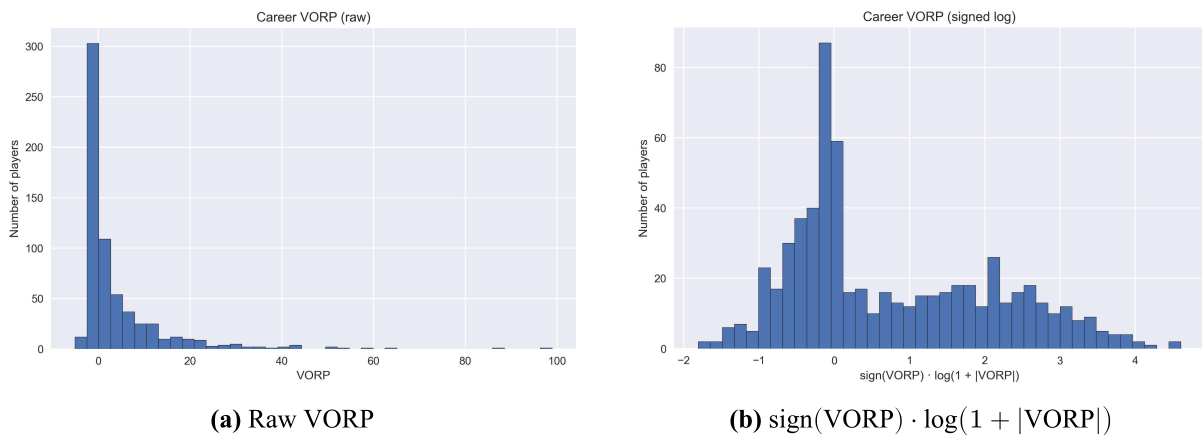
Table 2: *Engineered NBA outcome labels.*

Label	**Type**	**Definition / Notes**
`survival`	Binary	= 1 if player appeared in at least four NBA seasons (≈64.5% prevalence); corresponds to rookie contract length
`rotationr`	Binary	= 1 if player averaged ≥15 minutes per game (≈54.2% prevalence); indicates a sustained role in the rotation
`starter`	Binary	= 1 if player averaged ≥25 minutes per game (≈22.4% prevalence); reflects regular starter responsibility
`impact4_ws`	Binary	= 1 if player both survived 4+ years and accumulated at least 15 career Win Shares (≈36.1% prevalence); captures sustained presence with tangible box-score value
`impact4_vorp`	Binary	= 1 if player both survived 4+ years and accumulated at least +1.0 career VORP (≈39.3% prevalence); captures sustained presence with positive impact relative to replacement level



(a) Raw WS   **(b)** $\log(1 + \mathrm{WS})$

Figure 7: *Win Shares distributions.*



(a) Raw VORP   **(b)** $\mathrm{sign}(\mathrm{VORP}) \cdot \log(1 + |\mathrm{VORP}|)$

Figure 8: *VORP distributions.*

4.3 Model Family Selection

With the feature and label sets established, the next step was to select suitable model families for the predictive tasks. A broad range of methods was considered at the start, but only a subset was deemed useful for implementation. Predominant reasons were the structure of the dataset, the objectives of the study, and the balance between interpretability and predictive performance.

The dataset is comprised of 1,931 season-level observations for 695 unique players, with one to four observations per player. This size rules out approaches that depend on very large data volumes, and the repeated-measures structure requires care to avoid information leakage between training and test sets. Additionally, many features are correlated or constructed as ratios, which demands a certain tolerance of the models tolerant to multicollinearity. Interpretability was also prioritised, so that predictive results can be linked back to player characteristics in a way that increases understanding of the key predictive features.

Against this background, several model families were deemed not practical for this scenario. OLS (Ordinary Least Squares) and logistic regression were not pursued, as both are unstable in the presence of multicollinearity and lack regularisation. Highly flexible but data-hungry

methods such as deep neural networks or transformer-based tabular models were also excluded, given the limited sample size and the risk of overfitting. Other approaches such as k-Nearest Neighbours, Naïve Bayes, and kernel SVMs were likewise set aside, either because they scale poorly or because stronger alternatives exist within the same category.

The first pass of modelling therefore focussed on three representative families. Ridge and Elastic Net regression were chosen as a regularised linear baseline, both to test whether the engineered features carry predictive signal and to allow coefficients to be interpreted as directional effects. Random Forests were included as a non-linear benchmark, since they capture interactions between features while remaining robust to noise and resistant to overfitting in smaller datasets. Finally, Gradient Boosted Trees were selected because they have consistently shown the strongest performance on tabular data and are expected to deliver among the best results in this setting.

Together, these three families cover the spectrum from linear to highly non-linear, which allows me to assess not only whether predictive signal exists but also in what form it appears. Hyperparameters will be tuned through cross-validation and all models will be run within reproducible pipelines so that preprocessing and transformations remain consistent. All models produced season-level probabilities that were subsequently aggregated to the player level using predefined schemes.

4.4 Training and Validation

To ensure that predictive performance was assessed fairly, models were trained and validated within a cross-validation framework. The key challenge here is that the dataset contains multiple season-level entries for the same player. With a traditional approach this would create information leakage, as a player could appear both in the training and test sets.

To avoid this, all splits were grouped either by player or by draft year, so that no player was represented in more than one fold. A hold-out test set was created using a grouped shuffle split with a fixed random seed to guarantee reproducibility. Within the training set, grouped cross-validation was applied to tune hyperparameters and assess model stability across folds.

All preprocessing steps were embedded within scikit-learn pipelines to ensure that transformations were fitted only on the training folds and then applied to the validation or test data. This pipeline design prevents information leakage from scaling, logarithmic transformations, or winsorisation.

For classification tasks, class imbalance was handled through class-weight adjustments in the loss function, which allowed minority classes such as “starter” to be represented more fairly during training. No resampling methods, such as oversampling or undersampling, were applied. With a dataset of moderate size and only the “starter” label showing notable imbalance at about twenty-two percent positive cases, resampling was likely to either reduce the effective sample size or lead to overfitting. The original distributions were therefore preserved, with class weights

and evaluation metrics robust to skew providing the more appropriate strategy. All aggregation steps were subsequently performed fold-safely (within each validation fold's slice) to avoid leakage; the resulting player-level probabilities were then evaluated on the holdout set with thresholds selected from cross-validation.

4.5 Aggregating to the Player Level

Because each player can appear in multiple college seasons, the base models produced probabilities at the season level. To evaluate outcomes on the level of the player, these were combined into a single probability estimate per player. All aggregation was carried out in a fold-safe manner, so that for cross-validation each player's seasons were aggregated only within the validation fold. The resulting player-level probabilities were then evaluated on the holdout set using thresholds selected from cross-validation.

Several aggregation schemes were considered. The first was a simple average across all seasons, assigning equal weight to each. A second approach weighted season probabilities by the number of games, while a third weighted them by total minutes, giving greater influence to seasons with more on-court exposure.

As an alternative to weighted averaging, we also applied a probabilistic or-rule. Here the player-level probability is computed as

$$p_{\text{player}} = 1 - \prod_{i=1}^{n} (1 - p_i),$$

so that the estimate reflects the chance that the outcome occurred in at least one of the player's seasons. This method captures the idea that a single strong season can already be predictive of professional impact.

Finally, a stacked meta-learner was implemented. From the set of season-level probabilities per player, a small gradient boosting model was trained on summary features such as the mean, maximum, minimum, standard deviation, number of seasons, weighted averages by games and minutes, and a top-two average. This meta-learner produced a player-level prediction by learning which summaries of the season probabilities were most informative.

Evaluation was carried out using ROC-AUC, PR-AUC (Precision–Recall Area Under the Curve), and confusion-matrix-derived precision, recall and F1. In addition, calibration curves, ROC curves and precision–recall curves were inspected for each aggregation mode.

4.6 Evaluation Strategy

Model performance was assessed with a range of metrics suited to the binary outcome variables. Accuracy, precision, recall and the F1-score were inspevcted to capture different aspects of predictive balance. The ROC-AUC and PR-AUC were also included. These metrics provide more

reliable insight in imbalanced settings, where overall accuracy can be misleading. ROC-AUC reflects how well the model ranks cases across thresholds, while PR-AUC focuses directly on the trade-off between precision and recall when identifying positive cases. To evaluate the calibration of predicted probabilities, log loss and the Brier score were calculated to check whether predicted likelihoods match observed frequencies.

In addition to these aggregate metrics, confusion matrices were produced for each label. A confusion matrix tabulates the counts of correct and incorrect predictions, distinguishing between true positives, false positives, true negatives and false negatives. This allows a direct inspection of which types of errors the models tend to make. For example, a model may achieve reasonable recall but at the cost of many false positives, or vice versa. Since confusion matrices depend on operating at a fixed decision threshold τ, they highlight how models that rank well overall may still collapse when thresholded. To complement this threshold-specific view, PR–AUC curves were inspected for all labels, illustrating the full trade-off between precision and recall across τ. Together, these tools provide both a global view of model ranking ability and a practical view of classification performance at chosen cut-offs.

Evaluation was carried out on the held-out test set created from grouped splits. This ensured that the reported metrics reflect true generalisation performance. All models were first evaluated at the season level. In addition, the best-performing models chosen for player-level aggregation were subjected to the same evaluation procedure, so that performance can be compared both within seasons and across aggregated player outcomes. This design provides a principled framework for judging whether the signal in college data is strong enough to support meaningful career forecasts.

Since the models are ultimately intended to support decision-making rather than to provide causal explanations, treating them as black boxes is acceptable. Nevertheless, it can be valuable to know which features drive predictions most strongly. For this purpose, permutation feature importance was computed for the winning models, providing an interpretable view of the factors that most influence classification outcomes. As aggregation methods only combine season-level predictions, feature importance was calculated on the pre-aggregation models, where the underlying relationships between player features and outcomes are directly accessible.

5 Results

5.1 Model Performance

As the first step in the modelling process, the data was processed through the pipeline at the season level to provide an initial benchmark of predictive power before aggregating to the player level.

VORP and WS were initially modelled as continuous regression targets, since they are standard measures of on-court performance. Both raw and log-transformed variants were evaluated, with logged values giving a slight uplift but never exceeding an R^2 of 0.1. Given this very limited explanatory power, the regression setup was abandoned. Instead, VORP and WS are treated in their classifier versions, which—alongside the `starter` and `rotation` labels—provide alternative ways of capturing player performance beyond simple survival.

The results show a clear difference between the models, especially between linear and tree-based approaches. Ridge and Elastic Net, which were run first as baseline models, consistently delivered the weakest performance, with ROC–AUC around the mid-0.60 range and PR–AUC only marginally better than random. This was particularly evident for labels that were more imbalanced, such as `starter`, where the limited number of positive cases proved difficult for linear models to capture. While these models offer a good degree of interpretability, they provide only limited predictive power and were therefore disregarded going forward.

Moving on to Random Forests, there were significant improvements across all tasks. For example, on the `rotation` label, ROC–AUC reached around 0.69, representing a meaningful gain over linear models. PR–AUC values were similarly stronger, reflecting better identification of true positives among the many candidates. The performance boost stems from the model's ability to account for non-linear interactions between box-score features.

The last model family, Gradient Boosted Trees, achieved the strongest per-season results overall. Several variants were tested, including scikit-learn's Gradient Boosting, XGBoost, LightGBM, and CatBoost. Among them, scikit-learn's implementation performed strongest, mostly outperforming Random Forests, with ROC–AUC values in the low- to mid-0.70s. For `rotation`, the model achieved approximately 0.74 ROC–AUC, paired with higher PR–AUC than competing models. Boosting appeared particularly strong for moderately imbalanced outcomes.

Taken together, these findings establish a consistent ranking between the models: tree-based methods outperform linear baselines, and boosted trees still performed somewhat better than Random Forests, providing the best pre-aggregation performance. Absolute performance levels were still modest but clearly above random, demonstrating that college box-score data contains predictive information. It is, however, not sufficient for high-confidence forecasts on its own. To assist decision makers, player-level aggregation is necessary to make a single judgment for each player.

5.2 Aggregation

Having established model performance at the season level, the next step was to move to the player level. Draft-day decisions concern players rather than individual college seasons, so season-level outputs had to be aggregated into a single prediction per player. Several straightforward rules, along with one more complex learner, were tested for this purpose.

The primary experiments used gradient boosting, which had shown the strongest season-level results. As a runner-up, Random Forests were also carried forward for aggregation. This proved important, as their aggregated performance on some labels turned out to be surprisingly strong.

Across the aggregation methods, uniform averaging emerged as a robust baseline. Weighting predictions by games or minutes played provided only small gains compared to uniform averaging. This pattern held across boosting variants as well, suggesting that simple averaging already captures the main signal, with playing time implicitly reflected in the features. A stacked meta-learner provided a boost in some weaker boosted settings, most notably for `impact4_vorp`, but did not generalise improvements across labels. A further heuristic, the "`prob_or`" rule (interpreting success as the probability of succeeding in any season), consistently underperformed and was subsequently dropped.

Comparing boosting runs across the two experimental passes ("pre" without anthropometric features, and "post" with the addition of height and weight) showed that the inclusion of physical measurements had mixed effects. For `rotation` and `survival`, later runs with these features improved results, though gains were modest. For `impact4_vorp`, `impact4_ws`, and `starter`, performance declined compared to earlier boosting runs without anthropometrics. This suggests that while physical measurements add predictive value for career durability, they do not consistently improve the identification of high-impact players.

Taken together, the results establish a clear pattern: Random Forests achieved the strongest results on the durability-oriented labels `rotation` and `survival`, while boosted models remained ahead on the ceiling-oriented outcomes `impact4_vorp` and `starter`. For `impact4_ws`, Random Forests held a slight edge on PR–AUC despite weaker ROC–AUC, highlighting a trade-off between ranking ability and top-end precision. In all cases, PR–AUC served as the decisive criterion for model comparison, both because of the imbalanced label distributions and because, on draft day, decision makers require high confidence in their positive predictions.

The comparison highlights three Random Forest winners (`rotation`, `survival`, `impact4_ws`) and two boosted winners (`starter`, `impact4_vorp`). This balance confirms that both model families capture different aspects of the prediction task, with Random Forests excelling on high-prevalence outcomes and boosted trees on sparser, ceiling-type outcomes.

Table 3: *Boosted vs. Random Forest on all labels (test set).*

Label	Task	Model	ROC AUC	PR AUC	Pass	Aggregation
`survival`	Binary	Boosted **RF**	0.692 **0.719**	0.840 **0.844**	post **pre**	minutes **uniform**
`rotation`	Binary	Boosted **RF**	0.742 **0.780**	0.777 **0.820**	post **pre**	uniform **stack_gb**
`starter`	Binary	**Boosted** RF	**0.744** 0.710	**0.365** 0.373	**pre** pre	**minutes** uniform
`impact4_ws`	Binary	Boosted **RF**	0.675 **0.655**	0.578 **0.596**	pre **post**	minutes **uniform**
`impact4_vorp`	Binary	**Boosted** RF	**0.715** 0.676	**0.678** 0.617	**pre** pre	**stack_gb** games

5.3 Confusion Matrices

To validate robustness across thresholds, PR–AUC curves were generated for all winning models. They confirmed that area under the curve values match the reported metrics. As a representative example, one PR–AUC curve is shown for the strongest label, `rotation`.

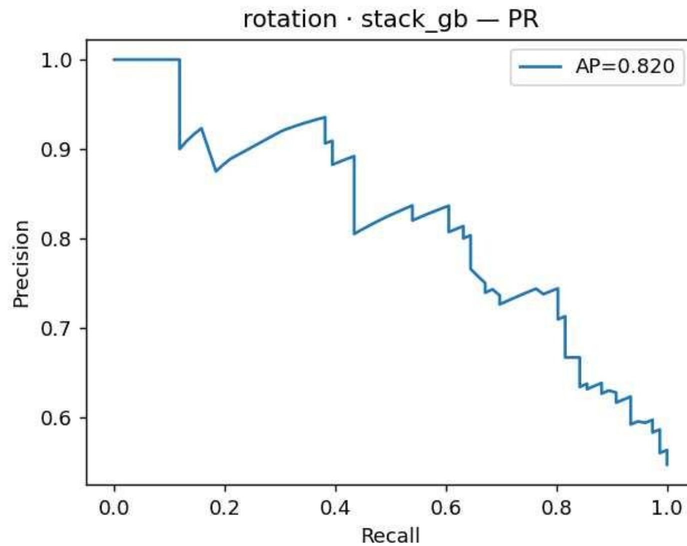


Figure 9: *Precision–Recall curve for Rotation (test set).*

While PR–AUC summarizes performance across thresholds, draft-day use requires operating at a fixed decision threshold. To illustrate this, confusion matrices were computed for each winning model at its selected threshold τ. These reveal that some labels with promising PR–AUC values collapse at the threshold level, such as `starter` and `impact4_ws`, which showed very low precision despite acceptable ranking ability. In contrast, `rotation` maintained a bal-

anced trade-off between precision and recall, underscoring its practical utility. For `survival`, the selected threshold effectively predicted all players as positive, yielding perfect recall but precision only at the base rate. This indicates that while the label provides a strong ranking signal as reflected in its PR–AUC, its thresholded predictions are less informative for direct draft-day use.

TN=0	FP=44
FN=0	TP=95

`survival` (RF, pre, uniform)

TN=25	FP=38
FN=9	TP=67

`rotation` (RF, pre, stack_gb)

TN=110	FP=3
FN=25	TP=1

`starter` (Boosted, pre, minutes)

TN=30	FP=60
FN=7	TP=42

`impact4_ws` (RF, post, uniform)

TN=36	FP=45
FN=14	TP=44

`impact4_vorp` (Boosted, pre, stack_gb)

Figure 10: *Confusion matrices of winning models (test set).*

5.4 Feature Importance

Finally, to look into what drives Predicions, PFI (Permutation Feature Importance) was computed for the winning model on `rotation`, based on the underlying season-level data sincefeature importance cannot be derived from the aggregated player-level predictions. The results show that importance values were relatively small in absolute terms, with the leading features reaching around 0.03. The top-ranked predictors were class year and games played, both clearly ahead of the rest. Assist-related statistics and effective field goal percentage followed, with importance values in the 0.015–0.025 range. The remaining features, including free-throw rate, rebounding, and free-throw percentage, contributed smaller effects below 0.01. Standard deviations indicate moderate variation across runs, suggesting that while the overall ranking of features is stable, the precise magnitudes are less certain.

Table 4: *Permutation feature importance for Rotation (test set).*

Feature	Importance (mean)	Std. dev.
class_FR	0.0316	0.0122
g	0.0258	0.0157
ast_per40	0.0241	0.0137
efg_pct	0.0162	0.0063
ast_tov	0.0153	0.0089
ft_rate	0.0091	0.0044
fta	0.0058	0.0068
fg_pct_college	0.0058	0.0086
drb_per40	0.0057	0.0048
ft_pct_college	0.0056	0.0056

5.5 Discussion of Key Results

This study asked to what extent college basketball statistics can predict player success during the first four years of an NBA career. The results suggest that while such data provides useful predictive signal, its strength varies across outcomes and its applicability is not uniform. At the season level, linear baselines such as Ridge and Elastic Net performed close to random, especially on imbalanced outcomes like `starter`. Tree-based methods were consistently stronger, with gradient boosting ahead of Random Forests before aggregation. Yet once predictions were aggregated to the player level—where draft decisions are made—the balance between models shifted.

Across the five labels, Random Forests and boosted models each emerged as winners. Random Forests proved strongest on durability-oriented outcomes, winning `rotation`, `survival`, and `impact4_ws`. Boosted models instead excelled on the ceiling-oriented outcomes `starter` and `impact4_vorp`. This split suggests that Random Forests provide stability where positive cases are common, while boosting is better at capturing rarer, higher-ceiling cases. From a methodological perspective, both model families add complementary value.

Aggregation was central to these results. Uniform averaging gave a strong baseline across labels, while more elaborate schemes such as stacking only helped selectively (notably for `impact4_vorp`). Weighting by minutes or games played added little beyond uniform aggregation, indicating that playing time is already encoded in the features. In practice, this means relatively simple aggregation strategies are sufficient for reliable player-level predictions.

The inclusion of height and weight for a second pass of all models had asymmetric effects. For `rotation` and `survival`, performance improved slightly, reinforcing the link between physical profile and career durability. For the ceiling-oriented labels, however, predictive accuracy declined, consistent with the idea that size and strength are necessary for survival but not decisive for excelling beyond replacement level. These findings mirror basketball intuition: physical attributes may help sustain careers, but skill and efficiency drive star-level impact.

Threshold-level evaluation revealed an additional layer of complexity. Precision–recall curves confirmed the reported PR–AUC values, yet confusion matrices showed that some la-

bels collapsed when reduced to binary predictions. Most notably, `survival` achieved a strong PR–AUC but at its selected threshold classified all players as survivors, offering little practical information. Similarly, *Starter* and `impact4_ws` showed respectable ranking ability but collapsed into low precision at threshold. In contrast, `rotation` maintained a workable balance between precision and recall, making it the most directly usable label. `impact4_vorp` provided some separation on upside potential but produced many false positives, limiting its standalone value.

Several points stand out from these findings. First, `rotation` is the most reliable and actionable label, capturing immediate contribution with balanced threshold performance. Second, `survival` offers strong ranking insight into career viability, but less guidance once thresholded. Third, ceiling-oriented labels such as `starter` and `impact4_vorp` contain signal but remain fragile, while `impact4_ws` provides only limited complementary value. These differences underline the trade-off between predictive strength and practical draft-day usability.

Some outcomes were unexpected. Random Forests outperformed boosting on durability outcomes, despite boosting's season-level superiority. Equally surprising, `impact4_ws` emerged as an RF "winner" in PR–AUC, even though it closely parallels VORP in concept. These results illustrate how methodological choices and label definitions interact in non-obvious ways. Most importantly, the decline in utility at threshold highlights how easily promising signals can erode when translated into real decision settings.

For decision makers, the findings carry several implications. The most immediate value lies in identifying prospects who are likely to secure rotation roles early in their careers. This helps reduce draft risk by clarifying which players are positioned to contribute at least at a baseline level, even if their eventual ceiling remains uncertain. Labels such as `rotation` and, to a lesser extent, `survival`, can therefore serve as practical guides for avoiding costly misses.

For decision makers, the findings highlight both opportunity and caution. The clearest value lies in distinguishing prospects who are likely to establish themselves as rotation players. This type of forecast can meaningfully reduce draft risk, ensuring that teams invest in players with a strong chance of early contribution, which is especailly importnat for rebuilding teams. `survival` adds a complementary perspective on long-term viability, even if its binary predictions are less actionable.

Where the models are less decisive is in forecasting ceiling outcomes such as starters or high-impact contributors. Here the signal is present but fragile, reminding decision makers that box-score models can help in ranking candidates but should not be treated as standalone forecasts. At the same time, the fact that any signal emerges from simple college statistics inputs is encouraging. It suggests that when combined with richer sources of information already available to teams such as biometrics, scouting, or tracking data—the same modelling framework could support a more complete picture of both floor and ceiling potential.

For interpretability, feature importance was examined on the winning model for `rotation`. The strongest signals came from the freshman class-year indicator and from games played, sug-

gesting that both early entry into the draft and consistent season-long availability are reliable markers of reaching NBA rotation level. Assist-related measures and scoring efficiency followed, highlighting the value of steady decision-making and efficient offense in translating to professional roles. While absolute importance values were modest at the season level, this reflects the limits of non-aggregated prediction; at the player level—where model performance was stronger—the underlying signals are likely clearer, even if they cannot be directly measured. For decision makers, this indicates that the models are capturing patterns consistent with established basketball logic, lending credibility to their outputs.

In summary, college statistics already offer draft-day value by highlighting rotation-level viability and filtering out clear risks. While they cannot yet pinpoint future stars with confidence, their integration into broader evaluation processes holds real promise. The results therefore underscore that even modest models can strengthen decision-making today, while pointing to the potential for sharper insights as data richness increases.

6 Conclusion

6.1 Summary

The goal of this work was to examine whether publicly available college basketball statistics can be used to usefully predict the early professional success of NBA players, since draft outcomes remain notoriously uncertain. And because rookie contracts tie franchises to multi-year commitments, teams face considerable risk when players fail to establish themselves. This work addressed that challenge by focusing on the first four years of NBA careers, which are both highly formative for players and extremely important financially for franchises, yet have received little systematic research.

To examine whether draft decisions can be informed by pre-draft performance, a dataset was built from scratch that covers all college players drafted between 2002 and 2019. Season-level college statistics were scraped, cleaned, and harmonized with subsequent NBA career outcomes, producing a consistent record of nearly two decades of draft cohorts. From this base, outcome labels were constructed to capture increasing levels of early-career success, from simply surviving a rookie contract to establishing a rotation role or even providing starter-level impact.

On top of this dataset, an end-to-end modelling pipeline was implemented that balanced interpretability with predictive power. Season averages were transformed into comparable features and validated through grouped cross-validation to respect the repeated-season structure of college careers. A range of model families was explored: regularized linear baselines were tested for transparency but found to provide little predictive signal, while tree-based methods proved markedly stronger. Random Forests and boosted ensembles were benchmarked side by side, allowing the analysis to assess not only whether predictive signal exists but also how different modelling approaches capture it. Predictions were generated at the season level and then aggregated to the player level, since this models the draft decision without loosing underlying data. Evaluation relied on metrics suited to imbalanced outcomes, and permutation feature importance was used to interpret the drivers behind the most reliable model.

The analysis shows that college box-score data does contain predictive signal, though of limited strength. Tree-based models in particular managed to identify players likely to claim rotation minutes and survive their initial contract window, providing a practical way of reducing risk. Attempts to forecast higher-ceiling outcomes such as starters or major impact players yielded weaker and less stable results, underlining the limits of what can be achieved from box scores alone. Still, the study demonstrates that even modest models, built on publicly available data, can sharpen draft-day evaluation and provide a reproducible baseline on which richer information from other data sources can be layered.

6.2 Limitations

This study is of course subject to the limitations imposed on it, largely through the constraints of data availability and the inability to perfectly capture real-world player development and career outcomes by means of theoretical models.

The analysis relies on publicly available college basketball data that can be collected and merged at scale within realistic technical limits. As a consequence, the sample covers only drafted college players in the 2002–2019 window, while, undrafted and international players as well as alternative development pathways remain outside the scope. Feature availability is restricted to box-score statistics aggregated at the season level together with basic height and weight information. Richer biometric measures such as wingspan or agility drills, or player-tracking statistics were not obtainable in a structured and consistent way that would have allowed usage for prediction. Similarly, coaching style, team scheme and lineup context were deliberately excluded in order to focus on pre-draft player information, but these omissions also remove part of the variance that shapes realized NBA outcomes. In addition, technical constraints on data-source scraping (enforced delays) prevented the collection of full game logs at scale, which would otherwise have allowed the addition of much more depth in terms of raw data, but also time-series effects that could have been taken into account. These restrictions define the scope of this work, as the models can only reflect what can be consistently observed and reproduced from public sources, while unobserved or finer-grained features could in principle improve predictive power.

Another limitation arises from the way outcomes are defined. The variables used here capture realized NBA roles and contributions rather than intrinsic talent. Metrics such as Win Shares and VORP also carry team and teammate effects, while role-based labels depend strongly on minutes allocation, health, and coaching decisions. Threshold choices and the restriction to the first few career seasons sharpen these context effects further. Strictly speaking, the models are estimating the likelihood of achieving certain roles or impacts given observed deployment, rather than isolating underlying player quality.

A further consideration is that the mapping from college performance to early NBA outcomes is not stationary. The period 2002–2019 spans rule changes and stylistic shifts in both college and professional basketball, the three-point revolution, the transfer portal, and alternative development leagues have since altered the player pipeline. We can try mitigating but cannot fully eliminate these differences, which means that external validity beyond the studied era and pathway mix must be viewed with caution. All results of the modelling process must therefore be viewed in light of these limitations and judged carefully for their real-world application.

6.3 Future Work

To improve on the answers to the question *"To what extent can college basketball statistics predict player success during the first four years of an NBA career?"*, future research can proceed

along two main directions.

For one, the data foundation can be extended. As covered in the previous chapter, access to more data is possible in principle; the structured collection of this data is the obstacle. However, if time and resources permit, full college game records can be obtained by an enduring approach to scraping, or possibly from the owners directly. Access to detailed game-level logs and play-by-play data would allow features that capture tempo and other situational dynamics that season averages obscure, as well as development curves over games, capturing trends as well as consistency. Further enrichment of the data foundation can be done by incorporating international and alternative development pathways—such as EuroLeague, FIBA juniors, the G League, or high school circuits—allowing for a fuller view of how drafted players made their way to the NBA. More detailed biometric and combine information, including wingspan, standing reach, vertical jump, agility drills, and body composition, would strengthen predictive performance compared to the coarse proxy of height and weight. These data are being recorded for the public draft, but are not available in an obtainable and structured way. In addition, tracking and video-based statistics produced by systems such as SportVU could quantify shot quality, contest distance, movement patterns, and spacing in ways that traditional box-score data cannot. These systems are slowly being installed in most large-scale college franchises, but access is still privileged. Textual information, such as scouting reports, could be processed with natural language methods, while injury and availability data would provide crucial context on player durability. These too are available via news outlets.

The second direction concerns outcomes and modelling strategies. Richer labels could be employed, such as continuous minutes played, contract values, or advanced impact measures like EPM (Estimated Plus-Minus), RAPM (Regularized Adjusted Plus-Minus), or WAR (Wins Above Replacement), moving beyond binary thresholds. I showed, however, that with the current data and pipeline setup, regression outcomes only carry very limited signal. Survival models could be used instead of binary labels to estimate the expected duration until a player reaches milestones such as establishing a rotation role or remaining in the league for four years. Sequence models, including convolutional or transformer-based architectures, would allow per-game trajectories to be modelled directly, while multi-instance learning approaches could treat a player as a collection of games or seasons with attention-based pooling. Such methods may capture developmental dynamics more faithfully than simple season-level averages.

Taken together, these avenues outline a wide research agenda, largely dependent on privileged access to data. By expanding the data inputs and then refining both the labels and the modelling approaches, future work can move closer to the complex reality of how pre-draft signals translate into professional basketball outcomes—and in doing so, continue to improve on the methods of draft-day decision making.

References

de Araújo Costa, D., Fechine, J. M., da Silva Brito, J. R., Ferro, J. V. R., de Barros Costa, E., & Lopes, R. V. V. (2024). A machine learning approach using interpretable models for predicting success of ncaa basketball players to reach nba. *Proceedings of the 16th International Conference on Agents and Artificial Intelligence (ICAART 2024)*, 758–765. https://doi.org/10.5220/0012390000003636

Farrell, S., Laity, E., Laughlin, D., Oliver, D., & Pennebaker, J. W. (2025). Beyond the box score: Using psychological metrics to forecast nba success [Conference presentation]. *MIT Sloan Sports Analytics Conference*.

Foutzopoulos, G., Pandis, N., & Tsagris, M. (2025). Predicting full retirement attainment of NBA players. *International Journal of Data Science and Analytics*. https://doi.org/10.1007/s41060-025-00821-z

Ichniowski, C., & Preston, A. (2021). Anchoring bias in the evaluation of basketball players: A closer look at nba draft decision-making. *Journal of Sports Analytics*.

Levin, A., Rangan, V., Spungin, C., & Tully, J. (2024, March). *Beyond the arc: Unveiling the pivotal impact of nba three-pointers*. Retrieved August 8, 2025, from https://sites.northwestern.edu/nusportsanalytics/2024/03/28/beyond-the-arc-unveiling-the-pivotal-impact-of-nba-three-pointers/

Miguel, C. G., Milan, F. J., de Almeida Soares, A. L., Quinaud, R., et al. (2019). Modelling the relationship between nba draft and the career longevity of players using generalized additive models. *Revista de Psicologia del Deporte*. https://www.researchgate.net/publication/331134640

NBA Communications. (2016, January). *Stats llc and nba to make stats sportvu player tracking data available to more fans than ever before*. Retrieved August 3, 2025, from https://pr.nba.com/stats-llc-nba-sportvu-player-tracking-data/

Patton, A. N., Scott, M., Walker, N., Ottenwess, A., Power, P., Cherukumudi, A., & Lucey, P. (2020). Predicting nba talent from enormous amounts of college basketball tracking data. *MIT Sloan Sports Analytics Conference*.

Rincon, M. (2024, May). *How sports analytics is changing the game*. Retrieved August 8, 2025, from https://news.asu.edu/20240507-business-and-entrepreneurship-how-sports-analytics-changing-game

Sailofsky, D. (2018, April). *Drafting errors and decision making theory in the nba draft* [Master of Arts thesis]. Brock University.

Sports Reference LLC. (2025a). *Basketball reference*. Retrieved September 10, 2025, from https://www.basketball-reference.com/

Sports Reference LLC. (2025b). *College basketball at sports reference*. Retrieved September 10, 2025, from https://www.sports-reference.com/cbb/

A Additional Figures

Sports Reference® | Baseball | Football (college) | Basketball (college) | Hockey | Soccer | Blog | Stathead® | Immaculate Grid® | Create Account | Ad-Free Login | Questions or Comments?

SRCBB

Enter Person, Team, Section, etc

Search

Players | Schools | Seasons | Leaders | Scores 1 | NCAA Tournaments | Stathead | Newsletter | Full Site Menu Below ▾

Consensus AA | 2001-02 WCC POY | 2x All-WCC | 2x All-WCC Tourney | 2x WCC Tourney MVP

21

Become a Stathead & surf this site ad-free.

Dan Dickau

Position: Guard

6-0, 190lb (183cm, 86kg)

Hometown: Vancouver, WA

Schools: Washington (Men) and Gonzaga (Men)

Draft: Sacramento Kings, 1st round (28th pick, 28th overall), 2002 NBA draft

SUMMARY	G	PTS	TRB	AST	FG%	FG3%	FT%	eFG%	WS
Career	97	13.3	2.4	3.7	45.1	46.2	85.4	57.8	12.4

Dickau Overview | Game Logs ▾ | More Dickau Pages ▾

On this page:

Per Game | Totals | Per 40 Minutes | Per 100 Poss | Advanced | Leaderboards, Awards, & Honors | Full Site Menu

Per Game ✨Upgraded✨ Share & Export ▾ Glossary Hide Partial Rows

Season

Season	Team	Conf	Class	Pos	G	GS	MP	FG	FGA	FG%	3P	3PA	3P%	2P	2PA	2P%	eFG%	FT	FTA	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV	PF	PTS	Awards
1997-98	Washington	Pac-10	FR	G	28	0	9.4	1.0	2.5	.420	0.6	1.1	.533	0.5	1.4	.333	.536	1.1	1.4	.795			0.9	1.0	0.3	0.0	1.5	1.3	3.8	
1998-99	Washington	Pac-10	SO	G	13	11	22.8	1.7	4.3	.393	0.8	2.4	.355	0.8	1.9	.440	.491	0.4	0.5	.714			2.9	2.6	0.8	0.0	2.2	2.1	4.6	
1999-00	Did not play - transfer																													
2000-01	Gonzaga	WCC	JR	G	24	24	33.7	5.5	11.4	.485	3.0	6.2	.480	2.6	5.3	.492	.615	4.8	5.6	.866	0.5	2.8	3.3	6.3	0.8	0.1	3.7	2.2	18.9	
2001-02	Gonzaga	WCC	SR	G	32	32	**34.7**	6.1	13.8	.441	3.7	8.0	**.457**	2.4	5.8	.419	**.574**	5.2	6.0	.864	0.5	2.5	3.0	4.7	0.8	0.1	2.9	1.9	**21.0**	AA-1
Career					**97**	**67**	**25.5**	**3.9**	**8.7**	**.451**	**2.2**	**4.8**	**.462**	**1.7**	**3.9**	**.436**	**.578**	**3.3**	**3.8**	**.854**	**0.5**	**2.6**	**2.4**	**3.7**	**0.7**	**0.1**	**2.6**	**1.8**	**13.3**	
Gonzaga (2 Yrs)					**56**	**56**	**34.3**	**5.9**	**12.8**	**.458**	**3.4**	**7.2**	**.465**	**2.5**	**5.6**	**.449**	**.589**	**5.0**	**5.8**	**.865**	**0.5**	**2.6**	**3.1**	**5.3**	**0.8**	**0.1**	**3.2**	**2.0**	**20.1**	
Washington (2 Yrs)					**41**	**11**	**13.6**	**1.2**	**3.0**	**.408**	**0.7**	**1.5**	**.443**	**0.6**	**1.6**	**.375**	**.516**	**0.9**	**1.1**	**.783**			**1.5**	**1.5**	**0.5**	**0.0**	**1.7**	**1.5**	**4.0**	

Bold season totals indicate player led men's conference.
Black Ink appears for stats with a leaderboard.

Figure A.1: *Sports-Reference player page.*

B Code Excerpts

A total of 17 final notebooks were produced. Due to constraints on appendix size and practicability, a choice was made to show the important notebooks, showing a scraping task, the label engineering and a modelling task. Notebooks were marginally visually edited for display.

Notebook 02 — College Scraper

```
# Imports and configuration

from pathlib import Path
import pandas as pd

# Project paths
DATA_DIR = Path("../1_data")
INTERIM_DIR = DATA_DIR / "1_2_interim"

# Input with draft rows and college URLs
URL_CSV = INTERIM_DIR / "nba_draft_with_college_urls.csv"

# Display option
pd.set_option("display.max_colwidth", 300)


# Load college URLs (Sports-Reference CBB)

df_urls_raw = pd.read_csv(URL_CSV)

# Keep only valid CBB player URLs
df_urls = (
    df_urls_raw[["Player", "DraftYear", "CollegeURL"]]
    .rename(columns={"CollegeURL": "college_url"})
    .dropna(subset=["college_url"])
    .drop_duplicates(subset=["college_url"])
    .reset_index(drop=True)
)

print(f"Loaded {len(df_urls_raw):,} rows from {URL_CSV.name}")
print(f"Usable unique CBB player URLs: {len(df_urls):,}")
df_urls.head(10)


# Install dependencies

!pip install --quiet cloudscraper bs4 lxml html5lib


# Test fetch with cloudscraper

import time, random
import cloudscraper

# Request delay with jitter
def human_delay():
    time.sleep(3.1 + random.uniform(0.4, 1.6))

# Scraper session (Chrome-like)
scraper = cloudscraper.create_scraper(
    browser={
        "browser": "chrome",
        "platform": "windows",
        "mobile": False,
    }
)

HEADERS = {
    "User-Agent": ("Mozilla/5.0 (Windows NT 10.0; Win64; x64) "
```

```
60                     "AppleWebKit/537.36 (KHTML, like Gecko) "
61                     "Chrome/126.0.0.0 Safari/537.36"),
62     "Accept-Language": "en-US,en;q=0.9",
63     "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8",
64     "Connection": "keep-alive",
65     "Referer": "https://www.sports-reference.com/cbb/players/",
66     "Upgrade-Insecure-Requests": "1",
67 }
68
69 test_url = df_urls.loc[0, "college_url"]
70
71 human_delay()
72 resp = scraper.get(test_url, headers=HEADERS, timeout=30)
73 print("Status:", resp.status_code)
74 print("Title contains 'Just a moment':", "Just a moment" in resp.text[:400])
75 print("HTML length:", len(resp.text))
76
77 html = resp.text if resp.status_code == 200 else None
78
79
80 # Extract the CBB Per-Game table (first pass)
81
82 import pandas as pd
83 from bs4 import BeautifulSoup, Comment
84
85 def extract_per_game_df(html: str) -> pd.DataFrame:
86     soup = BeautifulSoup(html, "lxml")
87
88     t = soup.select_one("table#players_per_game")
89     if t:
90         return pd.read_html(str(t))[0]
91
92     for c in soup.find_all(string=lambda x: isinstance(x, Comment)):
93         csoup = BeautifulSoup(c, "lxml")
94         t = csoup.select_one("table#players_per_game")
95         if t:
96             return pd.read_html(str(t))[0]
97
98     for t in soup.select("table"):
99         if t.select_one('[data-stat="pts_per_g"]'):
100            return pd.read_html(str(t))[0]
101    for c in soup.find_all(string=lambda x: isinstance(x, Comment)):
102        csoup = BeautifulSoup(c, "lxml")
103        for t in csoup.select("table"):
104            if t.select_one('[data-stat="pts_per_g"]'):
105                return pd.read_html(str(t))[0]
106
107    raise ValueError("Per-Game table not found.")
108
109 if html:
110     try:
111         per_game_df = extract_per_game_df(html)
112         print(f"Rows: {len(per_game_df)}, Cols: {len(per_game_df.columns)}")
113         per_game_df.head(10)
114     except Exception as e:
115         print("Parse error:", e)
116 else:
117     print("No HTML to parse.")
118
119
120 # Loop over URLs with checkpoints (test limit)
121
122 from pathlib import Path
123 import io
124 import pandas as pd
125 from bs4 import BeautifulSoup, Comment
126
127 # Parser using StringIO
128 def extract_per_game_df(html: str) -> pd.DataFrame:
129     soup = BeautifulSoup(html, "lxml")
130
131     t = soup.select_one("table#players_per_game")
```

```
132     if t:
133         return pd.read_html(io.StringIO(str(t)))[0]
134
135     for c in soup.find_all(string=lambda x: isinstance(x, Comment)):
136         csoup = BeautifulSoup(c, "lxml")
137         t = csoup.select_one("table#players_per_game")
138         if t:
139             return pd.read_html(io.StringIO(str(t)))[0]
140
141     for t in soup.select("table"):
142         if t.select_one('[data-stat="pts_per_g"]'):
143             return pd.read_html(io.StringIO(str(t)))[0]
144     for c in soup.find_all(string=lambda x: isinstance(x, Comment)):
145         csoup = BeautifulSoup(c, "lxml")
146         for t in csoup.select("table"):
147             if t.select_one('[data-stat="pts_per_g"]'):
148                 return pd.read_html(io.StringIO(str(t)))[0]
149
150     raise ValueError("Per-Game table not found.")
151
152 # Output configuration
153 OUT_DIR = Path("../1_data/1_3_raw")
154 OUT_DIR.mkdir(parents=True, exist_ok=True)
155 OUT_CSV = OUT_DIR / "college_season_stats_per_game.csv"
156
157 TEST_LIMIT = 10
158 MAX_RETRIES = 3
159 CHECKPOINT_EVERY = 25
160
161 # Fetcher (uses session from test cell)
162 def fetch_html(url: str) -> str | None:
163     for attempt in range(1, MAX_RETRIES + 1):
164         try:
165             human_delay()
166             r = scraper.get(url, headers=HEADERS, timeout=30)
167             interstitial = "Just a moment" in r.text[:600] or "Attention Required" in r.text[:600]
168             if r.status_code == 200 and not interstitial:
169                 return r.text
170             print(f"[{attempt}/{MAX_RETRIES}] Status {r.status_code} or interstitial → {url}")
171         except Exception as e:
172             print(f"[{attempt}/{MAX_RETRIES}] Error: {e}")
173     return None
174
175 # Main loop (test run)
176 rows = []
177 n = len(df_urls) if TEST_LIMIT is None else min(TEST_LIMIT, len(df_urls))
178
179 for i in range(n):
180     url = df_urls.loc[i, "college_url"]
181     player = df_urls.loc[i, "Player"] if "Player" in df_urls.columns else None
182     draft_year = df_urls.loc[i, "DraftYear"] if "DraftYear" in df_urls.columns else None
183
184     html = fetch_html(url)
185     if not html:
186         print(f" Skipping (no HTML): {url}")
187         continue
188
189     try:
190         df = extract_per_game_df(html)
191         if player is not None: df["player_name"] = player
192         if draft_year is not None: df["draft_year"] = draft_year
193         df["source_url"] = url
194
195         rows.append(df)
196         print(f" Parsed {player or url} ({i+1}/{n}) - {len(df)} rows")
197     except Exception as e:
198         print(f" Parse failed ({i+1}/{n}): {url} - {e}")
199
200     if (i + 1) % CHECKPOINT_EVERY == 0 and rows:
201         pd.concat(rows, ignore_index=True).to_csv(OUT_CSV, index=False)
202         print(f" Checkpoint written → {OUT_CSV}")
203
```

```
204 # Final write (test run)
205 if rows:
206     full = pd.concat(rows, ignore_index=True)
207     full.to_csv(OUT_CSV, index=False)
208     print(f " Saved {len(full)} rows to {OUT_CSV}")
209 else:
210     print("No rows collected.")
211
212
213 # Full scrape with resume and checkpoints
214
215 from pathlib import Path
216 import io
217 import pandas as pd
218 from bs4 import BeautifulSoup, Comment
219
220 # Parser using StringIO
221 def extract_per_game_df(html: str) -> pd.DataFrame:
222     soup = BeautifulSoup(html, "lxml")
223
224     t = soup.select_one("table#players_per_game")
225     if t:
226         return pd.read_html(io.StringIO(str(t)))[0]
227
228     for c in soup.find_all(string=lambda x: isinstance(x, Comment)):
229         csoup = BeautifulSoup(c, "lxml")
230         t = csoup.select_one("table#players_per_game")
231         if t:
232             return pd.read_html(io.StringIO(str(t)))[0]
233
234     for t in soup.select("table"):
235         if t.select_one('[data-stat="pts_per_g"]'):
236             return pd.read_html(io.StringIO(str(t)))[0]
237     for c in soup.find_all(string=lambda x: isinstance(x, Comment)):
238         csoup = BeautifulSoup(c, "lxml")
239         for t in csoup.select("table"):
240             if t.select_one('[data-stat="pts_per_g"]'):
241                 return pd.read_html(io.StringIO(str(t)))[0]
242
243     raise ValueError("Per-Game table not found.")
244
245 # Output configuration
246 OUT_DIR = Path("../1_data/1_3_raw")
247 OUT_DIR.mkdir(parents=True, exist_ok=True)
248 OUT_CSV = OUT_DIR / "college_season_stats_per_game.csv"
249
250 TEST_LIMIT = None
251 MAX_RETRIES = 3
252 CHECKPOINT_EVERY = 25
253
254 # Fetcher (uses session from test cell)
255 def fetch_html(url: str) -> str | None:
256     for attempt in range(1, MAX_RETRIES + 1):
257         try:
258             human_delay()
259             r = scraper.get(url, headers=HEADERS, timeout=30)
260             interstitial = "Just a moment" in r.text[:600] or "Attention Required" in r.text[:600]
261             if r.status_code == 200 and not interstitial:
262                 return r.text
263             print(f"[{attempt}/{MAX_RETRIES}] Status {r.status_code} or interstitial → {url}")
264         except Exception as e:
265             print(f"[{attempt}/{MAX_RETRIES}] Error: {e}")
266     return None
267
268 # Resume from existing CSV if present
269 processed = set()
270 if OUT_CSV.exists() and OUT_CSV.stat().st_size > 0:
271     try:
272         prev = pd.read_csv(OUT_CSV, usecols=["source_url"])
273         processed = set(prev["source_url"].dropna().astype(str))
274         print(f"Found existing CSV with {len(prev):,} rows from {len(processed):,} URLs. Will skip those.")
275     except Exception as e:
```

```
276          print("Could not read existing CSV for resume. Will start fresh.", e)
277
278  rows_buffer = []
279  buffer_target = CHECKPOINT_EVERY
280  header_written = OUT_CSV.exists() and OUT_CSV.stat().st_size > 0
281
282  def write_buffer(buf: list[pd.DataFrame], header_written: bool):
283      if not buf:
284          return header_written
285      chunk = pd.concat(buf, ignore_index=True)
286      mode = "a" if header_written else "w"
287      chunk.to_csv(OUT_CSV, index=False, mode=mode, header=not header_written)
288      print(f" Wrote {len(chunk):,} rows to {OUT_CSV}")
289      buf.clear()
290      return True
291
292  # Main loop (full run)
293  n = len(df_urls) if TEST_LIMIT is None else min(TEST_LIMIT, len(df_urls))
294
295  for i in range(n):
296      url = str(df_urls.loc[i, "college_url"])
297      if url in processed:
298          continue
299
300      player = df_urls.loc[i, "Player"] if "Player" in df_urls.columns else None
301      draft_year = df_urls.loc[i, "DraftYear"] if "DraftYear" in df_urls.columns else None
302
303      html = fetch_html(url)
304      if not html:
305          print(f" Skipping (no HTML): {url}")
306          continue
307
308      try:
309          df = extract_per_game_df(html)
310          if player is not None: df["player_name"] = player
311          if draft_year is not None: df["draft_year"] = draft_year
312          df["source_url"] = url
313
314          rows_buffer.append(df)
315          processed.add(url)
316          print(f" Parsed {player or url} - {len(df)} rows ({len(processed)}/{n})")
317      except Exception as e:
318          print(f" Parse failed ({i+1}/{n}): {url} - {e}")
319
320      if len(rows_buffer) >= buffer_target:
321          header_written = write_buffer(rows_buffer, header_written)
322
323  # Final flush
324  write_buffer(rows_buffer, header_written)
325  print (" Done.")
```

Notebook — Label Engineering

```
1   # Setup and configuration
2   from pathlib import Path
3   import datetime as dt
4
5   # Versioning
6   VERSION_TAG = "v3"
7   STAMP = dt.datetime.now().strftime("%Y%m%d")
8
9   # Paths
10  DATA_DIR = Path("../1_data")
11  JOINT_PATH = DATA_DIR / "1_2_interim" / "college_nba_joint_w_flags.csv"
12  MODEL_INPUTS_DIR = DATA_DIR / "1_5_modelinputs"
13  MODEL_INPUTS_DIR.mkdir(parents=True, exist_ok=True)
14
15  LABELS_OUT = MODEL_INPUTS_DIR / f"labels_season_{VERSION_TAG}.csv"
16  MODEL_INPUTS_OUT = MODEL_INPUTS_DIR / f"model_inputs_{VERSION_TAG}.csv"
```

```

# Label parameters
LABEL_PARAMS = {
    "survived_years_threshold": 4,
    "rotation_mpg_threshold": 15.0,
    "starter_mpg_threshold": 25.0,
    "ws_impact_threshold": 15.0,
    "vorp_impact_threshold": 1.0,
    "enforce_nesting": True,
}

print("JOINT_PATH:", JOINT_PATH)
print("LABELS_OUT:", LABELS_OUT)
print("MODEL_INPUTS_OUT:", MODEL_INPUTS_OUT)
print("Label params:", LABEL_PARAMS)


# Load joint dataset
import pandas as pd

joint = pd.read_csv(JOINT_PATH)

print("Joint dataset shape:", joint.shape)
print("Available columns (first 15):", joint.columns[:15].tolist())

nba_cols = ["yrs", "g_total", "mp_total", "mp_per_game", "ws", "vorp"]
print("\nNBA outcome columns present:", [c for c in joint.columns if c in nba_cols])

print("\nSummary of NBA outcome columns:")
print(joint[nba_cols].describe().T)


# Binary labels
import pandas as pd

labels_bin = pd.DataFrame({
    "player_name_college": joint["player_name_college"],
    "draft_year": joint["draft_year"],
    "season": joint.get("season", None),
})

# Survival
labels_bin["label_survived_4y"] = (
    joint["yrs"] >= LABEL_PARAMS["survived_years_threshold"]
).astype(int)

# Rotation
rotation_raw = (joint["mp_per_game"] >= LABEL_PARAMS["rotation_mpg_threshold"]).astype(int)
labels_bin["label_rotation"] = (
    (labels_bin["label_survived_4y"] == 1) & (rotation_raw == 1)
).astype(int) if LABEL_PARAMS["enforce_nesting"] else rotation_raw

# Starter
starter_raw = (joint["mp_per_game"] >= LABEL_PARAMS["starter_mpg_threshold"]).astype(int)
labels_bin["label_starter"] = (
    (labels_bin["label_survived_4y"] == 1) & (starter_raw == 1)
).astype(int) if LABEL_PARAMS["enforce_nesting"] else starter_raw

# Prevalence checks
print("Season-weighted means:")
print(labels_bin[["label_survived_4y","label_rotation","label_starter"]].mean().round(3))

player_level = (
    labels_bin
    .drop_duplicates(subset=["player_name_college","draft_year"])
    [["player_name_college","draft_year","label_survived_4y","label_rotation","label_starter"]]
)

print("\nPlayer-weighted means:")
print(player_level[["label_survived_4y","label_rotation","label_starter"]].mean().round(3))

n_season_rows = len(labels_bin)
```

```
89  n_players = len(player_level)
90  print(f"\nCounts -> season rows: {n_season_rows} | players: {n_players}")
91
92
93  # Continuous and composite labels
94  import numpy as np
95  import pandas as pd
96
97  def signed_log(x):
98      if pd.isna(x):
99          return np.nan
100     return np.sign(x) * np.log1p(abs(x))
101
102 labels_cont = pd.DataFrame({
103     "player_name_college": joint["player_name_college"],
104     "draft_year": joint["draft_year"],
105 })
106
107 # Continuous targets
108 labels_cont["label_ws"]   = joint["ws"]
109 labels_cont["label_vorp"] = joint["vorp"]
110
111 # Transformed targets
112 labels_cont["label_ws_log1p"]  = np.log1p(labels_cont["label_ws"].clip(lower=0))
113 labels_cont["label_vorp_slog"] = labels_cont["label_vorp"].apply(signed_log)
114
115 # Composite binaries
116 _surv = (
117     labels_bin
118     .drop_duplicates(subset=["player_name_college","draft_year"])
119     [["player_name_college","draft_year","label_survived_4y"]]
120     .rename(columns={"label_survived_4y":"_survived_4y"})
121 )
122
123 comb = labels_cont.merge(_surv, on=["player_name_college","draft_year"], how="left", validate="many_to_one")
124
125 labels_cont["label_impact4_ws"] = (
126     (comb["_survived_4y"] == 1) &
127     (comb["label_ws"] >= LABEL_PARAMS["ws_impact_threshold"])
128 ).astype(int)
129
130 labels_cont["label_impact4_vorp"] = (
131     (comb["_survived_4y"] == 1) &
132     (comb["label_vorp"] >= LABEL_PARAMS["vorp_impact_threshold"])
133 ).astype(int)
134
135 # Prevalence checks
136 player_level_cont = labels_cont.drop_duplicates(subset=["player_name_college","draft_year"])
137 print("Composite binary prevalences:")
138 print(player_level_cont[["label_impact4_ws","label_impact4_vorp"]].mean().round(3))
139
140 print("\nContinuous targets summary:")
141 print(player_level_cont[["label_ws","label_ws_log1p","label_vorp","label_vorp_slog"]].describe().T.round(3))
142
143
144 # Merge binary and continuous labels
145 import pandas as pd
146
147 bin_player  = labels_bin.drop_duplicates(subset=["player_name_college","draft_year"]).copy()
148 cont_player = labels_cont.drop_duplicates(subset=["player_name_college","draft_year"]).copy()
149
150 labels_player = bin_player.merge(
151     cont_player,
152     on=["player_name_college","draft_year"],
153     how="left",
154     validate="one_to_one"
155 )
156
157 # Monotonic checks
158 viol_starter_not_rotation = labels_player[(labels_player["label_starter"]==1) & (labels_player["
        label_rotation"]==0)]
```

```
viol_rotation_not_surv    = labels_player[(labels_player["label_rotation"]==1) & (labels_player["
     label_survived_4y"]==0)]
print("Monotonic violations:")
print("  starter   rotation:", len(viol_starter_not_rotation))
print("  rotation   survived_4y:", len(viol_rotation_not_surv))

# Save player-level labels
LABELS_OUT_PLAYER = MODEL_INPUTS_DIR / f"labels_player_{VERSION_TAG}.csv"
labels_player.to_csv(LABELS_OUT_PLAYER, index=False)
print(f"Saved player-level labels -> {LABELS_OUT_PLAYER} | shape={labels_player.shape}")

label_cols = [c for c in labels_player.columns if c.startswith("label_")]
print("Label columns included:", label_cols)


# Season-level labels and model inputs
import pandas as pd
from pathlib import Path

FEATURES_PATH = Path("../1_data/1_5_modelinputs/college_features_ready_2.csv")
features = pd.read_csv(FEATURES_PATH)
print(f"Loaded features -> {FEATURES_PATH} | shape={features.shape}")

required_keys = ["player_name_college","draft_year","season"]
assert all(c in features.columns for c in required_keys), "features missing required keys"

label_cols = [c for c in labels_player.columns if c.startswith("label_")]
labels_season = (
    features[required_keys].drop_duplicates()
    .merge(
        labels_player[["player_name_college","draft_year"] + label_cols],
        on=["player_name_college","draft_year"],
        how="left",
        validate="many_to_one"
    )
)

labels_season.to_csv(LABELS_OUT, index=False)
print(f"Saved season-level labels -> {LABELS_OUT} | shape={labels_season.shape}")

model_inputs = features.merge(
    labels_season,
    on=required_keys,
    how="left",
    validate="many_to_one"
)
model_inputs.to_csv(MODEL_INPUTS_OUT, index=False)
print(f"Saved model inputs -> {MODEL_INPUTS_OUT} | shape={model_inputs.shape}")

assert "season" in labels_season.columns, "'season' missing in labels_season"
assert all(c in model_inputs.columns for c in [*required_keys]), "merge lost required keys"
print("Integrity check passed.")
```

Notebook — Random Forest

```
# Imports and configuration

import os
from pathlib import Path
import warnings

import numpy as np
import pandas as pd

from sklearn.model_selection import GroupKFold
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline

# Reproducibility
```

```
15 SEED = 42
16 np.random.seed(SEED)
17
18 # Paths
19 DATA_DIR   = Path(r"..\1_data\1_5_modelinputs")
20 INPUT_FILE = DATA_DIR / "model_inputs_v3.csv"
21
22 MODEL_TAG  = "11_random_forest_v1"
23 OUT_DIR    = Path(rf"..\2_models\{MODEL_TAG}")
24 OUT_DIR.mkdir(parents=True, exist_ok=True)
25
26 # Columns and identifiers
27 ID_COL        = "player_id"
28 YEAR_COL      = "draft_year"
29 META_COLS     = ["player_id", "season", "draft_year", "conference", "class", "pos", "awards"]
30
31 # Labels (season-level)
32 BINARY_LABELS = ["survival", "rotation", "starter", "impact4_ws", "impact4_vorp"]
33 REG_LABELS    = ["log1p_ws", "signedlog_vorp"]
34
35 # Cross-validation (grouped by player_id)
36 N_FOLDS = 5
37 gkf = GroupKFold(n_splits=N_FOLDS)
38
39 warnings.filterwarnings("ignore")
40
41
42 # Load data and define feature columns
43
44 # Load
45 df = pd.read_csv(INPUT_FILE)
46 print(f"Loaded features -> {INPUT_FILE} | shape={df.shape}")
47
48 # Identify ID and year columns
49 ID_CANDIDATES = ["player_id", "player_name_college"]
50 ID_COL = next((c for c in ID_CANDIDATES if c in df.columns), None)
51 assert ID_COL is not None, f"None of the ID candidates found: {ID_CANDIDATES}"
52 assert "draft_year" in df.columns, "Missing required column: draft_year"
53
54 # Resolve label columns to canonical names
55 LABEL_CANDIDATES = {
56     "survival":       ["survival", "label_survived_4y"],
57     "rotation":       ["rotation", "label_rotation"],
58     "starter":        ["starter", "label_starter"],
59     "impact4_ws":     ["impact4_ws", "label_impact4_ws"],
60     "impact4_vorp":   ["impact4_vorp", "label_impact4_vorp"],
61     "log1p_ws":       ["log1p_ws", "label_ws_log1p"],
62     "signedlog_vorp": ["signedlog_vorp", "label_vorp_slog"],
63 }
64 resolved_labels = {}
65 for canon, options in LABEL_CANDIDATES.items():
66     found = next((c for c in options if c in df.columns), None)
67     assert found is not None, f"Missing label for '{canon}'. Tried: {options}"
68     resolved_labels[canon] = found
69     if canon not in df.columns:
70         df[canon] = df[found]
71
72 BINARY_LABELS = ["survival", "rotation", "starter", "impact4_ws", "impact4_vorp"]
73 REG_LABELS    = ["log1p_ws", "signedlog_vorp"]
74 ALL_LABELS    = BINARY_LABELS + REG_LABELS
75
76 # Define feature columns (numeric only; exclude meta and labels)
77 META_BASE = ["season", "draft_year", "conference", "class", "pos", "awards"]
78 legacy_label_cols = [c for c in df.columns if c.startswith("label_")]
79 legacy_label_names = set(sum(LABEL_CANDIDATES.values(), []))
80
81 exclude_cols = set(META_BASE + [ID_COL]) | set(ALL_LABELS) | set(legacy_label_cols) | legacy_label_names
82 candidate_cols = [c for c in df.columns if c not in exclude_cols]
83 feature_cols = df[candidate_cols].select_dtypes(include=[np.number]).columns.tolist()
84 assert len(feature_cols) > 0, "No numeric feature columns found after exclusions."
85
86 # Info
```

```
87  n_nan = int(df[feature_cols].isna().sum().sum())
88  print(f"ID column: {ID_COL}")
89  print(f"Features selected: {len(feature_cols)} | NaNs across features: {n_nan}")
90  print(f"Example features: {feature_cols[:10]}")
91  
92  
93  # Holdout split (grouped by player)
94  
95  from sklearn.model_selection import GroupShuffleSplit
96  
97  # Shared split path for reuse
98  SHARED_SPLIT_DIR = Path(r"..\2_models\_shared")
99  SHARED_SPLIT_DIR.mkdir(parents=True, exist_ok=True)
100 SPLIT_FILE = SHARED_SPLIT_DIR / f"holdout_players_{ID_COL}_seed{SEED}.csv"
101 
102 if SPLIT_FILE.exists():
103     holdout_players = pd.read_csv(SPLIT_FILE)[ID_COL].astype(df[ID_COL].dtype).tolist()
104     print(f"Loaded holdout players from {SPLIT_FILE} | count={len(holdout_players)}")
105 else:
106     # Create a grouped holdout split (by player)
107     gss = GroupShuffleSplit(n_splits=1, test_size=0.20, random_state=SEED)
108     idx_all = np.arange(len(df))
109     (train_idx, test_idx), = gss.split(idx_all, groups=df[ID_COL])
110 
111     holdout_players = df.loc[test_idx, ID_COL].drop_duplicates().tolist()
112     pd.DataFrame({ID_COL: holdout_players}).to_csv(SPLIT_FILE, index=False)
113     print(f"Created holdout players -> {SPLIT_FILE} | count={len(holdout_players)}")
114 
115 # Row-level masks/indices
116 is_holdout = df[ID_COL].isin(holdout_players)
117 idx_holdout = np.flatnonzero(is_holdout.values)
118 idx_train   = np.flatnonzero(~is_holdout.values)
119 
120 # Report
121 n_players_total = df[ID_COL].nunique()
122 n_players_hold  = len(holdout_players)
123 print(f"Players: total={n_players_total}, holdout={n_players_hold}, train={n_players_total - n_players_hold
         }")
124 print(f"Rows:    total={len(df)}, holdout={len(idx_holdout)}, train={len(idx_train)}")
125 
126 
127 # Metric helpers
128 
129 from typing import Dict, Tuple
130 from sklearn.metrics import (
131     roc_auc_score, average_precision_score, f1_score, balanced_accuracy_score,
132     log_loss, brier_score_loss, confusion_matrix, precision_recall_curve,
133     mean_squared_error, mean_absolute_error, r2_score
134 )
135 
136 EPS = 1e-12
137 
138 def _safe_proba(p: np.ndarray) -> np.ndarray:
139     """Clamp probabilities to a safe range."""
140     p = np.asarray(p).reshape(-1)
141     return np.clip(p, EPS, 1 - EPS)
142 
143 def eval_classification(y_true: np.ndarray, y_proba: np.ndarray, threshold: float = 0.5) -> Dict[str, float
         ]:
144     """Evaluate binary classification with positive-class probabilities."""
145     p = _safe_proba(y_proba)
146     y_pred = (p >= threshold).astype(int)
147     out = {
148         "roc_auc": float(roc_auc_score(y_true, p)),
149         "pr_auc": float(average_precision_score(y_true, p)),
150         "f1": float(f1_score(y_true, y_pred, zero_division=0)),
151         "balanced_accuracy": float(balanced_accuracy_score(y_true, y_pred)),
152         "log_loss": float(log_loss(y_true, p, eps=EPS)),
153         "brier": float(brier_score_loss(y_true, p))
154     }
155     return out
156 
```

```
157 def eval_regression(y_true: np.ndarray, y_pred: np.ndarray) -> Dict[str, float]:
158     """Evaluate regression predictions."""
159     rmse = float(np.sqrt(mean_squared_error(y_true, y_pred)))
160     mae  = float(mean_absolute_error(y_true, y_pred))
161     r2   = float(r2_score(y_true, y_pred))
162     return {"rmse": rmse, "mae": mae, "r2": r2}
163
164 def pr_curve_points(y_true: np.ndarray, y_proba: np.ndarray) -> Tuple[np.ndarray, np.ndarray, np.ndarray]:
165     """Return precision, recall, thresholds."""
166     p = _safe_proba(y_proba)
167     return precision_recall_curve(y_true, p)
168
169 def cm_counts(y_true: np.ndarray, y_proba: np.ndarray, threshold: float = 0.5) -> Dict[str, int]:
170     """Return confusion matrix counts at a given threshold."""
171     p = _safe_proba(y_proba)
172     y_pred = (p >= threshold).astype(int)
173     tn, fp, fn, tp = confusion_matrix(y_true, y_pred, labels=[0,1]).ravel()
174     return {"tn": int(tn), "fp": int(fp), "fn": int(fn), "tp": int(tp)}
175
176
177 # Random Forest classifier: grouped hyperparameter search and CV metrics
178
179 from sklearn.ensemble import RandomForestClassifier
180 from sklearn.model_selection import RandomizedSearchCV, cross_val_predict
181 from sklearn.metrics import make_scorer, average_precision_score
182
183 # Parameter space
184 param_distributions = {
185     "rf__n_estimators":       [400, 600, 800, 1000, 1200],
186     "rf__max_depth":          [None, 12, 16, 24, 32],
187     "rf__min_samples_leaf":   [1, 2, 4, 8],
188     "rf__min_samples_split":  [2, 4, 8, 16],
189     "rf__max_features":       ["sqrt", "log2", 0.3, 0.5],
190     "rf__class_weight":       ["balanced", "balanced_subsample"],
191 }
192
193 scoring = {"roc_auc": "roc_auc", "pr_auc": "average_precision"}
194
195 cv_results_cls = {}
196 best_params_map = {}
197
198 for label in BINARY_LABELS:
199     print(f"\n=== Random Forest (Classifier) | Label: {label} ===")
200
201     X = df.loc[idx_train, feature_cols].copy()
202     y = df.loc[idx_train, label].astype(int)
203     groups = df.loc[idx_train, ID_COL]
204
205     # Pipeline
206     pipe = Pipeline([
207         ("imputer", SimpleImputer(strategy="median")),
208         ("rf", RandomForestClassifier(random_state=SEED, n_jobs=-1))
209     ])
210
211     # Group-aware randomized search
212     search = RandomizedSearchCV(
213         estimator=pipe,
214         param_distributions=param_distributions,
215         n_iter=40,
216         scoring=scoring,
217         refit="roc_auc",
218         cv=gkf,
219         random_state=SEED,
220         n_jobs=-1,
221         verbose=0,
222         return_train_score=False
223     )
224     search.fit(X, y, groups=groups)
225
226     # Store best params and quick scores from the search
227     best_params = search.best_params_
228     best_index  = search.best_index_
```

```
229     mean_cv_roc = float(search.cv_results_["mean_test_roc_auc"][best_index])
230     mean_cv_pr  = float(search.cv_results_["mean_test_pr_auc"][best_index])
231     best_params_map[label] = {
232         "best_params": best_params,
233         "cv_mean_roc_auc": mean_cv_roc,
234         "cv_mean_pr_auc":  mean_cv_pr
235     }
236     print(f"Best ROC-AUC (CV): {mean_cv_roc:.3f} | Best PR-AUC (CV): {mean_cv_pr:.3f}")
237     print(f"Best params: {best_params}")
238 
239     # Out-of-fold probabilities using the selected best params
240     pipe_best = Pipeline([
241         ("imputer", SimpleImputer(strategy="median")),
242         ("rf", RandomForestClassifier(random_state=SEED, n_jobs=-1))
243     ])
244     pipe_best.set_params(**best_params)
245 
246     y_proba_oof = cross_val_predict(
247         pipe_best, X, y,
248         groups=groups,
249         cv=gkf,
250         method="predict_proba",
251         n_jobs=-1,
252         verbose=0
253     )[:, 1]
254 
255     # Metrics on OOF
256     metrics = eval_classification(y, y_proba_oof)
257     counts  = cm_counts(y, y_proba_oof)
258 
259     cv_results_cls[label] = {
260         **metrics,
261         **counts,
262         "cv_mean_roc_auc_search": mean_cv_roc,
263         "cv_mean_pr_auc_search":  mean_cv_pr
264     }
265     print(metrics)
266     print(counts)
267 
268 # Save summarized CV metrics and best params
269 pd.DataFrame(cv_results_cls).T.to_csv(OUT_DIR / "cv_metrics_classifier.csv")
270 pd.Series({k: v for k, v in best_params_map.items()}).to_json(OUT_DIR / "cv_rf_best_params.json", indent=2)
271 
272 print(f"\nSaved classifier CV metrics -> {OUT_DIR/'cv_metrics_classifier.csv'}")
273 print(f"Saved best params map -> {OUT_DIR/'cv_rf_best_params.json'}")
274 
275 
276 # Fit best RFs, select thresholds on CV, evaluate on holdout
277 
278 from time import time
279 from sklearn.base import clone
280 from sklearn.ensemble import RandomForestClassifier
281 from sklearn.model_selection import cross_val_predict
282 
283 try:
284     from tqdm.auto import tqdm
285     _use_tqdm = True
286 except Exception:
287     _use_tqdm = False
288 
289 # Load best params (from memory or disk)
290 if 'best_params_map' not in globals() or not best_params_map:
291     import json
292     with open(OUT_DIR / "cv_rf_best_params.json", "r") as f:
293         best_params_map = json.load(f)
294 
295 def find_best_thresholds(y_true: np.ndarray, y_proba: np.ndarray):
296     """Return thresholds that maximize F1 and balanced accuracy on CV predictions."""
297     p = _safe_proba(y_proba)
298     taus = np.linspace(0.0, 1.0, 201)
299     best_f1, best_tau_f1 = -1.0, 0.5
300     best_bal, best_tau_bal = -1.0, 0.5
```

```
301     for tau in taus:
302         y_pred = (p >= tau).astype(int)
303         f1   = f1_score(y_true, y_pred, zero_division=0)
304         bal  = balanced_accuracy_score(y_true, y_pred)
305         if f1 > best_f1:
306             best_f1, best_tau_f1 = f1, float(tau)
307         if bal > best_bal:
308             best_bal, best_tau_bal = bal, float(tau)
309     return {"tau_f1": best_tau_f1, "f1_at_tau": float(best_f1),
310             "tau_bal": best_tau_bal, "bal_at_tau": float(best_bal)}
311
312 holdout_metrics = {}
313 thresholds_map  = {}
314 pred_rows = []
315
316 for label in BINARY_LABELS:
317     print(f"\n=== Holdout evaluation | Label: {label} ===")
318     t0 = time()
319
320     # Data splits
321     X_tr = df.loc[idx_train, feature_cols].copy()
322     y_tr = df.loc[idx_train, label].astype(int).to_numpy()
323     grp  = df.loc[idx_train, ID_COL]
324     X_ho = df.loc[idx_holdout, feature_cols].copy()
325     y_ho = df.loc[idx_holdout, label].astype(int).to_numpy()
326
327     # Model with best params
328     pipe = Pipeline([
329         ("imputer", SimpleImputer(strategy="median")),
330         ("rf", RandomForestClassifier(random_state=SEED, n_jobs=-1))
331     ])
332     pipe.set_params(**best_params_map[label]["best_params"] if isinstance(best_params_map[label], dict)
333                     else best_params_map[label]["best_params"])
334
335     # CV probabilities for threshold selection
336     print•(" CV probability pass for threshold selection")
337     fold_indices = list(gkf.split(X_tr, y_tr, groups=grp))
338     y_proba_oof = np.zeros(len(X_tr), dtype=float)
339     iterator = tqdm(enumerate(fold_indices, 1), total=len(fold_indices), leave=False) if _use_tqdm \
340                else enumerate(fold_indices, 1)
341     for i, (tri, vai) in iterator:
342         if not _use_tqdm:
343             print(f"  - fold {i}/{len(fold_indices)}")
344         pipe.fit(X_tr.iloc[tri], y_tr[tri])
345         y_proba_oof[vai] = pipe.predict_proba(X_tr.iloc[vai])[:, 1]
346
347     # Threshold selection on CV
348     th = find_best_thresholds(y_tr, y_proba_oof)
349     thresholds_map[label] = th
350     print(f•" Selected thresholds: tau_f1={th['tau_f1']:.3f} (F1={th['f1_at_tau']:.3f}), "
351           f"tau_bal={th['tau_bal']:.3f} (BalAcc={th['bal_at_tau']:.3f})")
352
353     # Fit on full train and predict holdout
354     print•(" Fit on train and predict holdout")
355     pipe.fit(X_tr, y_tr)
356     y_proba_ho = pipe.predict_proba(X_ho)[:, 1]
357
358     # Threshold-free metrics
359     m_free = {
360         "roc_auc": float(roc_auc_score(y_ho, y_proba_ho)),
361         "pr_auc": float(average_precision_score(y_ho, y_proba_ho)),
362         "log_loss": float(log_loss(y_ho, _safe_proba(y_proba_ho), eps=EPS)),
363         "brier": float(brier_score_loss(y_ho, _safe_proba(y_proba_ho))),
364     }
365
366     # Thresholded metrics
367     def thresh_eval(tau):
368         return {
369             **eval_classification(y_ho, y_proba_ho, threshold=tau),
370             **{f"cm_{k}": v for k, v in cm_counts(y_ho, y_proba_ho, threshold=tau).items()}
371         }
372
```

```
373      m_05   = thresh_eval(0.5)
374      m_f1   = thresh_eval(th["tau_f1"])
375      m_bal  = thresh_eval(th["tau_bal"])
376
377      holdout_metrics[label] = {
378          "threshold_free": m_free,
379          "thr_0p5": m_05,
380          "thr_tau_f1": {"tau": th["tau_f1"], **m_f1},
381          "thr_tau_bal": {"tau": th["tau_bal"], **m_bal},
382          "best_params": best_params_map[label]["best_params"] if isinstance(best_params_map[label], dict)
383                         else best_params_map[label]["best_params"]
384      }
385
386      # Save per-row predictions
387      df_pred = pd.DataFrame({
388          ID_COL: df.loc[idx_holdout, ID_COL].values,
389          "label": label,
390          "y_true": y_ho,
391          "proba": y_proba_ho,
392          "pred_0p5": (y_proba_ho >= 0.5).astype(int),
393          "pred_tau_f1": (y_proba_ho >= th["tau_f1"]).astype(int),
394          "pred_tau_bal": (y_proba_ho >= th["tau_bal"]).astype(int),
395      })
396      pred_rows.append(df_pred)
397
398      print(f•" Done in {time()-t0:.1f}s")
399
400  # Save artifacts
401  preds_holdout = pd.concat(pred_rows, ignore_index=True)
402  preds_holdout.to_csv(OUT_DIR / "preds_holdout_classifier.csv", index=False)
403
404  import json
405  with open(OUT_DIR / "holdout_metrics_classifier.json", "w") as f:
406      json.dump(holdout_metrics, f, indent=2)
407  with open(OUT_DIR / "thresholds_classifier.json", "w") as f:
408      json.dump(thresholds_map, f, indent=2)
409
410  print(f"\nSaved holdout predictions -> {OUT_DIR/'preds_holdout_classifier.csv'}")
411  print(f"Saved holdout metrics -> {OUT_DIR/'holdout_metrics_classifier.json'}")
412  print(f"Saved thresholds -> {OUT_DIR/'thresholds_classifier.json'}")
413
414
415  # Inspect holdout metrics
416
417  import json
418
419  with open(OUT_DIR / "holdout_metrics_classifier.json", "r") as f:
420      holdout_metrics = json.load(f)
421
422  rows = []
423  for label, content in holdout_metrics.items():
424      roc_auc = content["threshold_free"]["roc_auc"]
425      pr_auc  = content["threshold_free"]["pr_auc"]
426      cm = {k: v for k, v in content["thr_0p5"].items() if k.startswith("cm_")}
427      rows.append({
428          "label": label,
429          "roc_auc": round(roc_auc, 3),
430          "pr_auc": round(pr_auc, 3),
431          "tn": cm["cm_tn"], "fp": cm["cm_fp"],
432          "fn": cm["cm_fn"], "tp": cm["cm_tp"]
433      })
434
435  summary_df = pd.DataFrame(rows)
436  print(summary_df.to_string(index=False))
437
438  # Summarize holdout metrics at F1-optimized thresholds
439
440  import json
441
442  with open(OUT_DIR / "holdout_metrics_classifier.json", "r") as f:
443      holdout_metrics = json.load(f)
444  with open(OUT_DIR / "thresholds_classifier.json", "r") as f:
```

```
445         thresholds_map = json.load(f)
446 
447     rows = []
448     for label, content in holdout_metrics.items():
449         roc_auc = content["threshold_free"]["roc_auc"]
450         pr_auc  = content["threshold_free"]["pr_auc"]
451 
452         thr = content["thr_tau_f1"]
453         cm  = {k: v for k, v in thr.items() if k.startswith("cm_")}
454         f1  = thr["f1"]
455 
456         rows.append({
457             "label": label,
458             "roc_auc": round(roc_auc, 3),
459             "pr_auc": round(pr_auc, 3),
460             "tau_f1": round(thr["tau"], 3),
461             "f1": round(f1, 3),
462             "tn": cm["cm_tn"], "fp": cm["cm_fp"],
463             "fn": cm["cm_fn"], "tp": cm["cm_tp"]
464         })
465 
466     summary_df = pd.DataFrame(rows)
467     print(summary_df.to_string(index=False))
468 
469 
470     # Summarize holdout metrics at balanced--accuracyoptimized thresholds
471 
472     import json
473 
474     with open(OUT_DIR / "holdout_metrics_classifier.json", "r") as f:
475         holdout_metrics = json.load(f)
476 
477     rows = []
478     for label, content in holdout_metrics.items():
479         roc_auc = content["threshold_free"]["roc_auc"]
480         pr_auc  = content["threshold_free"]["pr_auc"]
481 
482         thr = content["thr_tau_bal"]
483         cm  = {k: v for k, v in thr.items() if k.startswith("cm_")}
484         bal = thr["balanced_accuracy"]
485 
486         rows.append({
487             "label": label,
488             "roc_auc": round(roc_auc, 3),
489             "pr_auc": round(pr_auc, 3),
490             "tau_bal": round(thr["tau"], 3),
491             "bal_acc": round(bal, 3),
492             "tn": cm["cm_tn"], "fp": cm["cm_fp"],
493             "fn": cm["cm_fn"], "tp": cm["cm_tp"]
494         })
495 
496     summary_df = pd.DataFrame(rows)
497     print(summary_df.to_string(index=False))
498 
499 
500     # Random Forest regressor: grouped hyperparameter search and CV metrics
501 
502     from time import time
503     from sklearn.ensemble import RandomForestRegressor
504     from sklearn.model_selection import RandomizedSearchCV
505     from sklearn.base import clone
506 
507     try:
508         from tqdm.auto import tqdm
509         _use_tqdm_reg = True
510     except Exception:
511         _use_tqdm_reg = False
512 
513     param_distributions_reg = {
514         "rf__n_estimators":       [400, 600, 800, 1000, 1200],
515         "rf__max_depth":          [None, 12, 16, 24, 32],
516         "rf__min_samples_leaf":   [1, 2, 4, 8],
```

```
517         "rf__min_samples_split":  [2, 4, 8, 16],
518         "rf__max_features":       ["sqrt", "log2", 0.3, 0.5],
519     }
520 
521     scoring_reg = {
522         "rmse": "neg_root_mean_squared_error",
523         "mae":  "neg_mean_absolute_error",
524         "r2":   "r2",
525     }
526 
527     cv_results_reg = {}
528     best_params_reg_map = {}
529 
530     for label in REG_LABELS:
531         print(f"\n=== Random Forest (Regressor) | Target: {label} ===")
532         t0 = time()
533 
534         X_all = df.loc[idx_train, feature_cols]
535         y_all = df.loc[idx_train, label].astype(float)
536         g_all = df.loc[idx_train, ID_COL]
537 
538         mask = np.isfinite(y_all.values)
539         X = X_all.loc[mask].copy()
540         y = y_all.loc[mask].to_numpy()
541         groups = g_all.loc[mask]
542 
543         print(f•" Rows: train total={len(X_all)}, used={len(X)}, dropped_invalid_y={len(X_all)-len(X)}")
544 
545         pipe = Pipeline([
546             ("imputer", SimpleImputer(strategy="median")),
547             ("rf", RandomForestRegressor(random_state=SEED, n_jobs=-1))
548         ])
549 
550         print•(" Hyperparameter search: starting (n_iter=40)")
551         search = RandomizedSearchCV(
552             estimator=pipe,
553             param_distributions=param_distributions_reg,
554             n_iter=40,
555             scoring=scoring_reg,
556             refit="rmse",
557             cv=gkf,
558             random_state=SEED,
559             n_jobs=-1,
560             verbose=2,
561             return_train_score=False
562         )
563         search.fit(X, y, groups=groups)
564 
565         best_params = search.best_params_
566         best_index  = search.best_index_
567         mean_cv_rmse = float(-search.cv_results_["mean_test_rmse"][best_index])
568         mean_cv_mae  = float(-search.cv_results_["mean_test_mae"][best_index])
569         mean_cv_r2   = float(search.cv_results_["mean_test_r2"][best_index])
570 
571         best_params_reg_map[label] = {
572             "best_params": best_params,
573             "cv_mean_rmse": mean_cv_rmse,
574             "cv_mean_mae":  mean_cv_mae,
575             "cv_mean_r2":   mean_cv_r2
576         }
577         print(f•" Search done in {time()-t0:.1f}s | Best RMSE={mean_cv_rmse:.3f}, MAE={mean_cv_mae:.3f}, R²={
                 mean_cv_r2:.3f}")
578         print(f•" Best params: {best_params}")
579 
580         print•(" OOF prediction: starting grouped CV passes")
581         pipe_best = clone(pipe).set_params(**best_params)
582 
583         y_pred_oof = np.zeros(len(X), dtype=float)
584         fold_indices = list(gkf.split(X, y, groups=groups))
585         iterator = tqdm(enumerate(fold_indices, 1), total=len(fold_indices), leave=False) if _use_tqdm_reg \
586                    else enumerate(fold_indices, 1)
587 
```

```
588     for i, (tr_idx, va_idx) in iterator:
589         if not _use_tqdm_reg:
590             print(f"  - fold {i}/{len(fold_indices)}")
591         X_tr, y_tr = X.iloc[tr_idx], y[tr_idx]
592         X_va       = X.iloc[va_idx]
593         pipe_best.fit(X_tr, y_tr)
594         y_pred_oof[va_idx] = pipe_best.predict(X_va)
595
596     metrics = eval_regression(y, y_pred_oof)
597     cv_results_reg[label] = {
598         **metrics,
599         "cv_mean_rmse_search": mean_cv_rmse,
600         "cv_mean_mae_search":  mean_cv_mae,
601         "cv_mean_r2_search":   mean_cv_r2
602     }
603     print(f•" OOF metrics: {metrics}")
604     print(f•" Total elapsed: {time()-t0:.1f}s")
605
606 pd.DataFrame(cv_results_reg).T.to_csv(OUT_DIR / "cv_metrics_regressor.csv")
607 pd.Series({k: v for k, v in best_params_reg_map.items()}).to_json(OUT_DIR / "cv_rf_reg_best_params.json",
      indent=2)
608
609 print(f"\nSaved regressor CV metrics -> {OUT_DIR/'cv_metrics_regressor.csv'}")
610 print(f"Saved best params map (reg) -> {OUT_DIR/'cv_rf_reg_best_params.json'}")
611
612 # %%
613 # Fit best RF regressors and evaluate on holdout
614
615 from time import time
616 from sklearn.ensemble import RandomForestRegressor
617
618 # Load best params from memory or disk
619 if 'best_params_reg_map' not in globals() or not best_params_reg_map:
620     import json
621     with open(OUT_DIR / "cv_rf_reg_best_params.json", "r") as f:
622         best_params_reg_map = json.load(f)
623
624 holdout_metrics_reg = {}
625 pred_rows_reg = []
626
627 for label in REG_LABELS:
628     print(f"\n=== Holdout evaluation (Regressor) | Target: {label} ===")
629     t0 = time()
630
631     X_tr_all = df.loc[idx_train, feature_cols]
632     y_tr_all = df.loc[idx_train, label].astype(float)
633     X_ho_all = df.loc[idx_holdout, feature_cols]
634     y_ho_all = df.loc[idx_holdout, label].astype(float)
635
636     m_tr = np.isfinite(y_tr_all.values)
637     m_ho = np.isfinite(y_ho_all.values)
638
639     X_tr = X_tr_all.loc[m_tr].copy()
640     y_tr = y_tr_all.loc[m_tr].to_numpy()
641     X_ho = X_ho_all.loc[m_ho].copy()
642     y_ho = y_ho_all.loc[m_ho].to_numpy()
643
644     print(f•" Rows: train used={len(X_tr)} (dropped={len(X_tr_all)-len(X_tr)}), "
645           f"holdout used={len(X_ho)} (dropped={len(X_ho_all)-len(X_ho)})")
646
647     pipe = Pipeline([
648         ("imputer", SimpleImputer(strategy="median")),
649         ("rf", RandomForestRegressor(random_state=SEED, n_jobs=-1))
650     ])
651     params = best_params_reg_map[label]["best_params"] if isinstance(best_params_reg_map[label], dict) \
652              else best_params_reg_map[label]["best_params"]
653     pipe.set_params(**params)
654
655     print•(" Fit on train and predict holdout")
656     pipe.fit(X_tr, y_tr)
657     y_pred_ho = pipe.predict(X_ho)
658
```

```
659        m = eval_regression(y_ho, y_pred_ho)
660        holdout_metrics_reg[label] = {**m, "best_params": params}
661        print(f•" Holdout metrics: {m}")
662        print(f•" Done in {time()-t0:.1f}s")
663
664        df_pred = pd.DataFrame({
665            ID_COL: df.loc[idx_holdout, ID_COL].loc[m_ho].values,
666            "target": label,
667            "y_true": y_ho,
668            "y_pred": y_pred_ho,
669        })
670        pred_rows_reg.append(df_pred)
671
672    preds_holdout_reg = pd.concat(pred_rows_reg, ignore_index=True)
673    preds_holdout_reg.to_csv(OUT_DIR / "preds_holdout_regressor.csv", index=False)
674
675    import json
676    with open(OUT_DIR / "holdout_metrics_regressor.json", "w") as f:
677        json.dump(holdout_metrics_reg, f, indent=2)
678
679    print(f"\nSaved holdout predictions (reg) -> {OUT_DIR/'preds_holdout_regressor.csv'}")
680    print(f"Saved holdout metrics (reg) -> {OUT_DIR/'holdout_metrics_regressor.json'}")
681
682
683    # %%
684    # Configuration and helpers for classifier aggregation from holdout predictions
685
686    from pathlib import Path
687    import numpy as np
688    import pandas as pd
689
690    from sklearn.metrics import (
691        roc_auc_score, average_precision_score, brier_score_loss, log_loss,
692        precision_recall_fscore_support, confusion_matrix, roc_curve, precision_recall_curve
693    )
694    from sklearn.calibration import calibration_curve
695    import matplotlib.pyplot as plt
696
697    # Paths
698    RF_DIR   = Path(r"..\2_models\11_random_forest_v1")
699    CSV_PATH = RF_DIR / "preds_holdout_classifier.csv"
700
701    # Optional: season-level exposures for games/minutes weighting. If unavailable, uniform weighting is used.
702    FEAT_PATH = Path(r"..\1_data\1_5_modelinputs\college_features_ready.csv")  # optional
703
704    PLAYER_KEY_NAME = "player_name_college"
705    LABEL_COL       = "label"
706    YTRUE_COL       = "y_true"
707    PROBA_COL       = "proba"
708
709    # RF label names in the file (no 'label_' prefix)
710    LABELS_CLF_RF = ["survival", "rotation", "starter", "impact4_ws", "impact4_vorp"]
711
712    # Helpers
713    def _compute_total_minutes(df: pd.DataFrame) -> pd.Series:
714        g  = df["g"]  if "g"  in df.columns else (df["games"] if "games" in df.columns else None)
715        mp = df["mp"] if "mp" in df.columns else (df["mp_per_g"] if "mp_per_g" in df.columns else None)
716        if g is not None and mp is not None:
717            return pd.to_numeric(g, errors="coerce") * pd.to_numeric(mp, errors="coerce")
718        return pd.Series(np.nan, index=df.index, dtype="float64")
719
720    def _weights(sub: pd.DataFrame, mode: str):
721        """Return normalized weights per row for the given aggregation mode."""
722        n = len(sub)
723        if n == 0:
724            return np.array([])
725        if mode == "uniform":
726            return np.full(n, 1.0 / n)
727        if mode == "games" and "g" in sub.columns:
728            w = pd.to_numeric(sub["g"], errors="coerce").to_numpy(dtype=float)
729        elif mode == "minutes" and "total_minutes" in sub.columns:
730            w = pd.to_numeric(sub["total_minutes"], errors="coerce").to_numpy(dtype=float)
```

```
731         else:
732             return np.full(n, 1.0 / n)
733         w = np.where(np.isfinite(w) & (w > 0), w, 0.0)
734         s = w.sum()
735         return (w / s) if s > 0 else np.full(n, 1.0 / n)
736 
737     def _agg_probs(df: pd.DataFrame, mode: str, binary_col: str):
738         """Aggregate season-level probabilities to player-level using a weighting mode."""
739         rows = []
740         for pid, sub in df.groupby(PLAYER_KEY_NAME):
741             w = _weights(sub, mode)
742             p = float(np.dot(w, sub[PROBA_COL].to_numpy(dtype=float)))
743             y = int(sub[binary_col].iloc[0])
744             rows.append((pid, p, y))
745         return pd.DataFrame(rows, columns=[PLAYER_KEY_NAME, "p_player", binary_col])
746 
747     def _metrics(y, p):
748         p_clip = np.clip(p, 1e-6, 1 - 1e-6)
749         return dict(
750             auc   = roc_auc_score(y, p) if len(np.unique(y)) == 2 else np.nan,
751             pr    = average_precision_score(y, p),
752             brier = brier_score_loss(y, p_clip),
753             logloss = log_loss(y, p_clip, labels=[0, 1])
754         )
755 
756     def _best_tau(y, p):
757         taus = np.linspace(0, 1, 501)
758         best_tau, best_f1 = 0.5, -1.0
759         for t in taus:
760             yhat = (p >= t).astype(int)
761             _, _, f1, _ = precision_recall_fscore_support(y, yhat, average="binary", zero_division=0)
762             if f1 > best_f1:
763                 best_tau, best_f1 = t, f1
764         return best_tau
765 
766 
767     # %%
768     # Aggregate season holdout predictions to player level (uniform/games/minutes)
769 
770     from pathlib import Path
771     import numpy as np, pandas as pd
772     from sklearn.metrics import roc_auc_score, average_precision_score, brier_score_loss, log_loss
773     from sklearn.metrics import precision_recall_fscore_support
774 
775     # Configuration
776     CSV_PATH = r"C:\Users\fwiet\OneDrive\Desktop\Catolica\_Thesis\_Code\2_models\11_random_forest_v1\
            preds_holdout_classifier.csv"
777     PLAYER_KEY_NAME = "player_name_college"
778 
779     # RF file uses short label names; map them to the boosted/analysis convention
780     LABEL_MAP = {
781         "survival":      "label_survived_4y",
782         "rotation":      "label_rotation",
783         "starter":       "label_starter",
784         "impact4_ws":    "label_impact4_ws",
785         "impact4_vorp":  "label_impact4_vorp",
786     }
787 
788     # Iterate in this order
789     LABELS_CLF = list(LABEL_MAP.values())
790 
791     # Column names in RF CSV
792     LABEL_COL = "label"
793     YTRUE_COL = "y_true"
794     PROBA_COL = "proba"
795 
796     # Helpers (minimal)
797     def _compute_total_minutes(df: pd.DataFrame) -> pd.Series:
798         g  = df["g"]  if "g"  in df.columns else (df["games"] if "games" in df.columns else None)
799         mp = df["mp"] if "mp" in df.columns else (df["mp_per_g"] if "mp_per_g" in df.columns else None)
800         if g is not None and mp is not None:
801             g  = pd.to_numeric(g, errors="coerce")
```

```
802         mp = pd.to_numeric(mp, errors="coerce")
803         return g * mp
804     return pd.Series(np.nan, index=df.index, dtype="float64")
805 
806 def _weights(sub: pd.DataFrame, mode: str):
807     if mode == "uniform":
808         return np.full(len(sub), 1/len(sub)) if len(sub) else np.array([])
809     elif mode == "games":
810         w = sub["g"].to_numpy().astype(float)
811     else:  # "minutes"
812         w = sub["total_minutes"].to_numpy().astype(float)
813     w = np.where(np.isfinite(w) & (w > 0), w, 0.0)
814     s = w.sum()
815     return (w / s) if s > 0 else (np.full(len(sub), 1/len(sub)) if len(sub) else np.array([]))
816 
817 def _agg_probs(df: pd.DataFrame, mode: str, label_col: str):
818     rows = []
819     for pid, sub in df.groupby(PLAYER_KEY_NAME):
820         w = _weights(sub, mode)
821         p = float(np.dot(w, sub["proba"].to_numpy()))
822         y = int(sub[label_col].iloc[0])
823         rows.append((pid, p, y))
824     return pd.DataFrame(rows, columns=[PLAYER_KEY_NAME, "p_player", label_col])
825 
826 def _metrics(y, p):
827     p_clip = np.clip(p, 1e-6, 1-1e-6)
828     out = dict(
829         auc   = roc_auc_score(y, p) if len(np.unique(y)) == 2 else np.nan,
830         pr    = average_precision_score(y, p),
831         brier = brier_score_loss(y, p_clip),
832         logloss = log_loss(y, p_clip, labels=[0,1]),
833     )
834     return out
835 
836 # Load RF holdout CSV
837 df_all = pd.read_csv(CSV_PATH)
838 need_cols = {PLAYER_KEY_NAME, LABEL_COL, YTRUE_COL, PROBA_COL}
839 missing = [c for c in need_cols if c not in df_all.columns]
840 if missing:
841     raise ValueError(f"Missing columns in RF CSV: {missing}\nFound: {df_all.columns.tolist()}")
842 
843 # Normalize label names to boosted-style (new column 'label_std')
844 if set(df_all[LABEL_COL].unique()) - set(LABEL_MAP.keys()):
845     print (" Some labels in RF CSV are not in LABEL_MAP; they will be ignored:",
846           sorted(set(df_all[LABEL_COL].unique()) - set(LABEL_MAP.keys())))
847 df_all["label_std"] = df_all[LABEL_COL].map(LABEL_MAP)
848 
849 # Loop per label and aggregate
850 test_store_rf   = {}
851 test_players_rf = {}
852 summary_rows = []
853 labels_in_file_std = sorted([l for l in df_all["label_std"].dropna().unique().tolist()])
854 
855 for LABEL in LABELS_CLF:
856     if LABEL not in labels_in_file_std:
857         print(f "  Skipping LABEL='{LABEL}': not present in RF CSV (std labels found: {labels_in_file_std})
            ")
858         continue
859 
860     base_cols  = [PLAYER_KEY_NAME, "label_std", YTRUE_COL, PROBA_COL]
861     extra_cols = [c for c in ["g","games","mp","mp_per_g","total_minutes"] if c in df_all.columns]
862     use_cols   = list(dict.fromkeys(base_cols + extra_cols))
863     sub = df_all.loc[df_all["label_std"] == LABEL, use_cols].copy()
864 
865     # Normalize exposure column names
866     if "games" in sub.columns and "g" not in sub.columns:
867         sub.rename(columns={"games": "g"}, inplace=True)
868     if "mp_per_g" in sub.columns and "mp" not in sub.columns:
869         sub.rename(columns={"mp_per_g": "mp"}, inplace=True)
870 
871     # Ensure total_minutes
872     if "total_minutes" not in sub.columns:
```

```
873            sub["total_minutes"] = np.nan
874        if sub["total_minutes"].isna().all() and {"g","mp"}.issubset(sub.columns):
875            sub["total_minutes"] = _compute_total_minutes(sub)
876
877        # Rename target to match helper expectations (use LABEL as the target column name)
878        sub.rename(columns={YTRUE_COL: LABEL}, inplace=True)
879
880        # Decide which aggregation modes are feasible
881        modes = ["uniform"]
882        if "g" in sub.columns and sub["g"].notna().any():
883            modes.append("games")
884        if np.isfinite(sub["total_minutes"]).any():
885            modes.append("minutes")
886
887        # Build a season-level frame with only necessary columns
888        keep_cols = [PLAYER_KEY_NAME, "proba", LABEL] + [c for c in ["g","total_minutes"] if c in sub.columns]
889        sub = sub[keep_cols].copy()
890
891        # Aggregate and score
892        test_results = {}
893        player_preds = {}
894        for mode in modes:
895            dfp = _agg_probs(sub, mode, LABEL)
896            y, p = dfp[LABEL].to_numpy(), dfp["p_player"].to_numpy()
897            m = _metrics(y, p)
898
899            # Report f1/precision/recall at 0.5
900            yhat = (p >= 0.5).astype(int)
901            prec, rec, f1, _ = precision_recall_fscore_support(y, yhat, average="binary", zero_division=0)
902
903            test_results[mode] = dict(
904                auc=m["auc"], pr=m["pr"], brier=m["brier"], logloss=m["logloss"],
905                tau_used=0.5, precision=prec, recall=rec, f1=f1, base_rate=float(y.mean())
906            )
907            player_preds[mode] = dfp
908
909            summary_rows.append({
910                "label": LABEL, "aggregation": mode,
911                "AUC_Test": m["auc"], "PR_AUC_Test": m["pr"],
912                "Brier_Test": m["brier"], "LogLoss_Test": m["logloss"],
913                "Precision@0.5": prec, "Recall@0.5": rec, "F1@0.5": f1,
914                "base_rate_Test": float(y.mean())
915            })
916
917        test_store_rf[LABEL]   = test_results
918        test_players_rf[LABEL] = player_preds
919        print(f"{LABEL}: done. modes={modes}")
920
921    # Summary
922    summary_rf_df = pd.DataFrame(summary_rows).sort_values(
923        by=["label","AUC_Test","PR_AUC_Test"], ascending=[True, False, False]
924    ).reset_index(drop=True)
925
926    summary_rf_df
927
928
929    # %%
930    from pathlib import Path
931
932    list(Path(r"2_models/11_random_forest_v1").glob("**/*oof*.csv"))
933
934
935    # %%
936    # F1: season-level OOF predictions with exposures for Random Forest
937
938    from pathlib import Path
939    import numpy as np, pandas as pd
940    from sklearn.pipeline import Pipeline
941    from sklearn.impute import SimpleImputer
942    from sklearn.ensemble import RandomForestClassifier
943
944    OOF_DIR = OUT_DIR / "oof"
```

```
945 OOF_DIR.mkdir(parents=True, exist_ok=True)
946
947 oof_store_rf = {}
948
949 def _series_or_nan(col_primary, col_alt, idx):
950     if col_primary in df.columns:
951         return df.loc[idx, col_primary].reset_index(drop=True)
952     if col_alt in df.columns:
953         return df.loc[idx, col_alt].reset_index(drop=True)
954     return pd.Series(np.nan, index=range(len(idx)), dtype="float64")
955
956 for label in BINARY_LABELS:
957     print(f"\n=== Building OOF season preds (RF) | Label: {label} ===")
958
959     X = df.loc[idx_train, feature_cols].copy()
960     y = df.loc[idx_train, label].astype(int).to_numpy()
961     groups = df.loc[idx_train, ID_COL]
962
963     # Pipeline (params will be set on the pipeline, not inside RF constructor)
964     pipe = Pipeline([
965         ("imputer", SimpleImputer(strategy="median")),
966         ("rf", RandomForestClassifier(random_state=SEED, n_jobs=-1)),
967     ])
968
969     # Set tuned params (keys like 'rf__n_estimators', etc.)
970     bp = best_params_map[label]["best_params"] if isinstance(best_params_map[label], dict) \
971          else best_params_map[label]["best_params"]
972     pipe.set_params(**bp)
973
974     oof_proba = np.full(len(y), np.nan, dtype=float)
975     oof_fold  = np.full(len(y), -1, dtype=int)
976
977     for fold_id, (tr_idx, va_idx) in enumerate(gkf.split(X, y, groups=groups)):
978         pipe.fit(X.iloc[tr_idx], y[tr_idx])
979         oof = pipe.predict_proba(X.iloc[va_idx])[:, 1]
980         oof_proba[va_idx] = oof
981         oof_fold[va_idx]  = fold_id
982
983     # Exposures and player key (robust to alternative column names)
984     sub = pd.DataFrame({
985         PLAYER_KEY_NAME: df.loc[idx_train, PLAYER_KEY_NAME].astype(str).reset_index(drop=True)
986                          if PLAYER_KEY_NAME in df.columns else df.loc[idx_train, ID_COL].astype(str).
                                 reset_index(drop=True),
987         label: y,
988     })
989     g_series  = _series_or_nan("g", "games", idx_train)
990     mp_series = _series_or_nan("mp", "mp_per_g", idx_train)
991     sub["g"] = pd.to_numeric(g_series, errors="coerce")
992     sub["mp"] = pd.to_numeric(mp_series, errors="coerce")
993     sub["total_minutes"] = sub["g"] * sub["mp"]
994
995     df_oof = pd.DataFrame({
996         "row": np.arange(len(y)),
997         "fold_id": oof_fold,
998         "proba": oof_proba,
999         label: y,
1000        PLAYER_KEY_NAME: sub[PLAYER_KEY_NAME].values,
1001        "g": sub["g"].values,
1002        "mp": sub["mp"].values,
1003        "total_minutes": sub["total_minutes"].values,
1004    })
1005
1006    oof_store_rf[label] = df_oof
1007
1008    out_path = OOF_DIR / f"class_{label}" / "oof_season.csv"
1009    out_path.parent.mkdir(parents=True, exist_ok=True)
1010    df_oof.to_csv(out_path, index=False)
1011    print(f"Saved OOF season preds -> {out_path} | shape={df_oof.shape}")
1012
1013
1014 # %%
1015 # F2: CV player-level aggregation from OOF (uniform/games/minutes)
```

```
1016
1017 import numpy as np, pandas as pd
1018 from sklearn.metrics import roc_auc_score, average_precision_score, brier_score_loss, log_loss
1019 from sklearn.metrics import precision_recall_fscore_support
1020
1021 # Helpers (local, minimal)
1022 def _weights(sub: pd.DataFrame, mode: str):
1023     n = len(sub)
1024     if n == 0: return np.array([])
1025     if mode == "uniform": return np.full(n, 1.0 / n)
1026     if mode == "games" and "g" in sub.columns:
1027         w = pd.to_numeric(sub["g"], errors="coerce").to_numpy(float)
1028     elif mode == "minutes" and "total_minutes" in sub.columns:
1029         w = pd.to_numeric(sub["total_minutes"], errors="coerce").to_numpy(float)
1030     else:
1031         return np.full(n, 1.0 / n)
1032     w = np.where(np.isfinite(w) & (w > 0), w, 0.0)
1033     s = w.sum()
1034     return (w / s) if s > 0 else np.full(n, 1.0 / n)
1035
1036 def _agg_probs(df: pd.DataFrame, mode: str, label_col: str, player_col: str):
1037     rows = []
1038     for pid, sub in df.groupby(player_col):
1039         w = _weights(sub, mode)
1040         p = float(np.dot(w, sub["proba"].to_numpy(float)))
1041         y = int(sub[label_col].iloc[0])
1042         rows.append((pid, p, y))
1043     return pd.DataFrame(rows, columns=[player_col, "p_player", label_col])
1044
1045 def _metrics(y, p):
1046     p_clip = np.clip(p, 1e-6, 1-1e-6)
1047     return dict(
1048         auc = roc_auc_score(y, p) if len(np.unique(y)) == 2 else np.nan,
1049         pr  = average_precision_score(y, p),
1050         brier = brier_score_loss(y, p_clip),
1051         logloss = log_loss(y, p_clip, labels=[0,1])
1052     )
1053
1054 def _best_tau(y, p):
1055     taus = np.linspace(0, 1, 501)
1056     best_tau, best_f1 = 0.5, -1.0
1057     for t in taus:
1058         yhat = (p >= t).astype(int)
1059         _, _, f1, _ = precision_recall_fscore_support(y, yhat, average="binary", zero_division=0)
1060         if f1 > best_f1:
1061             best_tau, best_f1 = float(t), float(f1)
1062     return best_tau
1063
1064 # Aggregate within-fold OOF for each label
1065 cv_store_rf = {}
1066 oof_players_rf = {}
1067 summary_rows = []
1068
1069 for label in BINARY_LABELS:
1070     if label not in oof_store_rf:
1071         print(f" Skipping '{label}': no OOF store found.")
1072         continue
1073
1074     df_oof = oof_store_rf[label].copy()
1075     assert {"fold_id","proba",label}.issubset(df_oof.columns), "OOF DF missing required columns"
1076
1077     # Determine feasible modes
1078     modes = ["uniform"]
1079     if "g" in df_oof.columns and df_oof["g"].notna().any(): modes.append("games")
1080     if "total_minutes" in df_oof.columns and np.isfinite(df_oof["total_minutes"]).any(): modes.append("
          minutes")
1081
1082     cv_results = {}
1083     player_frames = {}
1084
1085     for mode in modes:
1086         parts = []
```

```
1087         for f in sorted(df_oof["fold_id"].unique()):
1088             part = df_oof.loc[df_oof["fold_id"] == f, [PLAYER_KEY_NAME, "proba", label] +
1089                               [c for c in ["g","total_minutes"] if c in df_oof.columns]].copy()
1090             parts.append(_agg_probs(part, mode, label, PLAYER_KEY_NAME))
1091         dfp = pd.concat(parts, ignore_index=True)
1092         y, p = dfp[label].to_numpy(), dfp["p_player"].to_numpy()
1093 
1094         m = _metrics(y, p)
1095         tau = _best_tau(y, p)
1096         cv_results[mode] = {**m, "tau": tau, "base_rate": float(y.mean())}
1097         player_frames[mode] = dfp
1098 
1099         summary_rows.append({
1100             "label": label, "aggregation": mode,
1101             "AUC_CV": m["auc"], "PR_AUC_CV": m["pr"],
1102             "Brier_CV": m["brier"], "LogLoss_CV": m["logloss"],
1103             "tau_OOF": tau, "base_rate_CV": float(y.mean())
1104         })
1105 
1106     cv_store_rf[label] = cv_results
1107     oof_players_rf[label] = player_frames
1108     print(f"{label}: CV aggregation done. modes={modes}")
1109 
1110 # Save CV summary
1111 summary_cv_df = pd.DataFrame(summary_rows).sort_values(
1112     by=["label","AUC_CV","PR_AUC_CV"], ascending=[True, False, False]
1113 ).reset_index(drop=True)
1114 summary_cv_df.to_csv(OUT_DIR / "cv_player_agg_rf.csv", index=False)
1115 print(f"Saved CV player-level summary -> {OUT_DIR/'cv_player_agg_rf.csv'}")
1116 summary_cv_df.head(10)
1117 
1118 
1119 # %%
1120 # F3: Holdout season-to-player aggregation (uniform/games/minutes)
1121 
1122 import numpy as np, pandas as pd
1123 from sklearn.metrics import roc_auc_score, average_precision_score, brier_score_loss, log_loss
1124 from sklearn.metrics import precision_recall_fscore_support
1125 
1126 PRED_CSV = OUT_DIR / "preds_holdout_classifier.csv"
1127 assert PRED_CSV.exists(), f"Missing holdout preds at {PRED_CSV}"
1128 
1129 # Helpers (minimal)
1130 def _weights(sub: pd.DataFrame, mode: str):
1131     n = len(sub)
1132     if n == 0: return np.array([])
1133     if mode == "uniform": return np.full(n, 1.0 / n)
1134     if mode == "games" and "g" in sub.columns:
1135         w = pd.to_numeric(sub["g"], errors="coerce").to_numpy(float)
1136     elif mode == "minutes" and "total_minutes" in sub.columns:
1137         w = pd.to_numeric(sub["total_minutes"], errors="coerce").to_numpy(float)
1138     else:
1139         return np.full(n, 1.0 / n)
1140     w = np.where(np.isfinite(w) & (w > 0), w, 0.0)
1141     s = w.sum()
1142     return (w / s) if s > 0 else np.full(n, 1.0 / n)
1143 
1144 def _agg_probs(df: pd.DataFrame, mode: str, label_col: str, player_col: str):
1145     rows = []
1146     for pid, sub in df.groupby(player_col):
1147         w = _weights(sub, mode)
1148         p = float(np.dot(w, sub["proba"].to_numpy(float)))
1149         y = int(sub[label_col].iloc[0])
1150         rows.append((pid, p, y))
1151     return pd.DataFrame(rows, columns=[player_col, "p_player", label_col])
1152 
1153 def _metrics(y, p):
1154     p_clip = np.clip(p, 1e-6, 1-1e-6)
1155     return dict(
1156         auc = roc_auc_score(y, p) if len(np.unique(y)) == 2 else np.nan,
1157         pr  = average_precision_score(y, p),
1158         brier = brier_score_loss(y, p_clip),
```

```
1159         logloss = log_loss(y, p_clip, labels=[0,1])
1160     )
1161
1162 # Load season holdout predictions and attach exposures
1163 ph = pd.read_csv(PRED_CSV)
1164 need_cols = {"label","y_true","proba"}
1165 missing = [c for c in need_cols if c not in ph.columns]
1166 assert not missing, f"preds_holdout missing columns: {missing}"
1167
1168 # Normalize exposures from df[idx_holdout]
1169 def _series_or_nan(col_primary, col_alt, idx):
1170     if col_primary in df.columns:
1171         return df.loc[idx, col_primary].reset_index(drop=True)
1172     if col_alt in df.columns:
1173         return df.loc[idx, col_alt].reset_index(drop=True)
1174     return pd.Series(np.nan, index=range(len(idx)), dtype="float64")
1175
1176 player_series = (df.loc[idx_holdout, PLAYER_KEY_NAME].astype(str).reset_index(drop=True)
1177                  if PLAYER_KEY_NAME in df.columns else
1178                  df.loc[idx_holdout, ID_COL].astype(str).reset_index(drop=True))
1179 g_series  = pd.to_numeric(_series_or_nan("g","games", idx_holdout), errors="coerce")
1180 mp_series = pd.to_numeric(_series_or_nan("mp","mp_per_g", idx_holdout), errors="coerce")
1181 tm_series = g_series * mp_series
1182
1183 # Build a holdout-season frame per label and aggregate
1184 test_store_rf   = {}
1185 test_players_rf = {}
1186 summary_rows = []
1187
1188 labels_in_file = sorted(ph["label"].unique().tolist())
1189 for label in BINARY_LABELS:
1190     if label not in labels_in_file:
1191         print(f" Skipping '{label}': not in preds_holdout file.")
1192         continue
1193
1194     sub = ph.loc[ph["label"] == label, ["y_true","proba"]].copy().reset_index(drop=True)
1195     sub.rename(columns={"y_true": label}, inplace=True)
1196     sub[PLAYER_KEY_NAME] = player_series.values
1197     sub["g"] = g_series.values
1198     sub["mp"] = mp_series.values
1199     sub["total_minutes"] = tm_series.values
1200
1201     modes = ["uniform"]
1202     if sub["g"].notna().any(): modes.append("games")
1203     if np.isfinite(sub["total_minutes"]).any(): modes.append("minutes")
1204
1205     # Retrieve OOF tau (informational)
1206     tau_info = {}
1207     if "cv_store_rf" in globals() and label in cv_store_rf:
1208         for m in modes:
1209             tau_info[m] = cv_store_rf[label].get(m, {}).get("tau", 0.5)
1210
1211     test_results = {}
1212     player_preds = {}
1213     for mode in modes:
1214         keep = [PLAYER_KEY_NAME, "proba", label] + [c for c in ["g","total_minutes"] if c in sub.columns]
1215         dfp = _agg_probs(sub[keep], mode, label, PLAYER_KEY_NAME)
1216         y, p = dfp[label].to_numpy(), dfp["p_player"].to_numpy()
1217
1218         m = _metrics(y, p)
1219
1220         # Reference metrics at threshold 0.5
1221         yhat = (p >= 0.5).astype(int)
1222         prec, rec, f1, _ = precision_recall_fscore_support(y, yhat, average="binary", zero_division=0)
1223
1224         test_results[mode] = dict(
1225             auc=m["auc"], pr=m["pr"], brier=m["brier"], logloss=m["logloss"],
1226             tau_oof=tau_info.get(mode, 0.5),
1227             precision_at_0p5=prec, recall_at_0p5=rec, f1_at_0p5=f1,
1228             base_rate=float(y.mean())
1229         )
1230         player_preds[mode] = dfp
```

```
        summary_rows.append({
            "label": label, "aggregation": mode,
            "AUC_Test": m["auc"], "PR_AUC_Test": m["pr"],
            "Brier_Test": m["brier"], "LogLoss_Test": m["logloss"],
            "tau_OOF": tau_info.get(mode, 0.5),
            "Precision@0.5": prec, "Recall@0.5": rec, "F1@0.5": f1,
            "base_rate_Test": float(y.mean())
        })

    test_store_rf[label]   = test_results
    test_players_rf[label] = player_preds
    print(f"{label}: holdout aggregation done. modes={modes}")

# Save holdout summary
summary_test_df = pd.DataFrame(summary_rows).sort_values(
    by=["label","AUC_Test","PR_AUC_Test"], ascending=[True, False, False]
).reset_index(drop=True)
summary_test_df.to_csv(OUT_DIR / "test_player_agg_rf.csv", index=False)
print(f"Saved holdout player-level summary -> {OUT_DIR/'test_player_agg_rf.csv'}")
summary_test_df.head(20)


# %%
# F4: Stacked GBM on RF OOF player features and holdout evaluation (no confusion matrices)

import numpy as np, pandas as pd
from pathlib import Path
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import roc_auc_score, average_precision_score, brier_score_loss, log_loss
from sklearn.metrics import precision_recall_fscore_support

PRED_CSV = OUT_DIR / "preds_holdout_classifier.csv"
assert PRED_CSV.exists(), f"Missing holdout preds at {PRED_CSV}"

def _metrics(y, p):
    p_clip = np.clip(p, 1e-6, 1-1e-6)
    return dict(
        auc = roc_auc_score(y, p) if len(np.unique(y)) == 2 else np.nan,
        pr  = average_precision_score(y, p),
        brier = brier_score_loss(y, p_clip),
        logloss = log_loss(y, p_clip, labels=[0,1])
    )

def _best_tau(y, p):
    taus = np.linspace(0, 1, 501)
    best_tau, best_f1 = 0.5, -1.0
    for t in taus:
        yhat = (p >= t).astype(int)
        _, _, f1, _ = precision_recall_fscore_support(y, yhat, average="binary", zero_division=0)
        if f1 > best_f1:
            best_tau, best_f1 = float(t), float(f1)
    return best_tau

def _player_stack_features(df_season: pd.DataFrame, label: str, player_col: str) -> pd.DataFrame:
    feats = []
    for pid, sub in df_season.groupby(player_col):
        p = sub["proba"].to_numpy(dtype=float)

        # Optional weights
        w_games = None
        if "g" in sub.columns and np.isfinite(sub["g"]).any():
            w = pd.to_numeric(sub["g"], errors="coerce").to_numpy(dtype=float)
            w = np.where(np.isfinite(w) & (w > 0), w, 0.0)
            w_games = (w / w.sum()) if w.sum() > 0 else None

        w_min = None
        if "total_minutes" in sub.columns and np.isfinite(sub["total_minutes"]).any():
            w = pd.to_numeric(sub["total_minutes"], errors="coerce").to_numpy(dtype=float)
            w = np.where(np.isfinite(w) & (w > 0), w, 0.0)
            w_min = (w / w.sum()) if w.sum() > 0 else None

```

```
1303            mean_p = float(np.mean(p))
1304            feats.append({
1305                player_col: pid,
1306                label: int(sub[label].iloc[0]),
1307                "mean_p": mean_p,
1308                "max_p": float(np.max(p)),
1309                "min_p": float(np.min(p)),
1310                "std_p": float(np.std(p)) if len(p) > 1 else 0.0,
1311                "count_seasons": int(len(p)),
1312                "mean_games": float(np.average(p, weights=w_games)) if w_games is not None else mean_p,
1313                "mean_minutes": float(np.average(p, weights=w_min))   if w_min   is not None else mean_p,
1314                "top2_mean": float(np.mean(np.sort(p)[-2:])) if len(p) >= 2 else float(p[0]),
1315            })
1316        return pd.DataFrame(feats)
1317
1318    # Ensure stores exist
1319    if "cv_store_rf" not in globals(): cv_store_rf = {}
1320    if "test_store_rf" not in globals(): test_store_rf = {}
1321    if "test_players_rf" not in globals(): test_players_rf = {}
1322
1323    summary_rows = []
1324
1325    # Build holdout season frame base (shared across labels)
1326    ph = pd.read_csv(PRED_CSV)
1327
1328    def _series_or_nan(col_primary, col_alt, idx):
1329        if col_primary in df.columns:
1330            return df.loc[idx, col_primary].reset_index(drop=True)
1331        if col_alt in df.columns:
1332            return df.loc[idx, col_alt].reset_index(drop=True)
1333        return pd.Series(np.nan, index=range(len(idx)), dtype="float64")
1334
1335    player_holdout = (df.loc[idx_holdout, PLAYER_KEY_NAME].astype(str).reset_index(drop=True)
1336                      if PLAYER_KEY_NAME in df.columns else
1337                      df.loc[idx_holdout, ID_COL].astype(str).reset_index(drop=True))
1338    g_holdout  = pd.to_numeric(_series_or_nan("g","games", idx_holdout), errors="coerce")
1339    mp_holdout = pd.to_numeric(_series_or_nan("mp","mp_per_g", idx_holdout), errors="coerce")
1340    tm_holdout = g_holdout * mp_holdout
1341
1342    for label in BINARY_LABELS:
1343        if label not in oof_store_rf:
1344            print(f" Skipping '{label}': no OOF season store.")
1345            continue
1346        if label not in ph["label"].unique():
1347            print(f" Skipping '{label}': not present in holdout preds file.")
1348            continue
1349
1350        # OOF to player meta set (train meta-learner)
1351        df_oof = oof_store_rf[label].copy()
1352        df_oof_players = _player_stack_features(df_oof, label, PLAYER_KEY_NAME)
1353        X_meta = df_oof_players[["mean_p","max_p","min_p","std_p","count_seasons","mean_games","mean_minutes","
                 top2_mean"]].to_numpy()
1354        y_meta = df_oof_players[label].to_numpy()
1355
1356        meta = GradientBoostingClassifier(
1357            random_state=SEED, n_estimators=100, max_depth=2,
1358            learning_rate=0.05, subsample=1.0, min_samples_leaf=10
1359        )
1360        meta.fit(X_meta, y_meta)
1361
1362        p_meta_cv = meta.predict_proba(X_meta)[:, 1]
1363        m_cv = _metrics(y_meta, p_meta_cv)
1364        tau_cv = _best_tau(y_meta, p_meta_cv)
1365
1366        if label not in cv_store_rf: cv_store_rf[label] = {}
1367        cv_store_rf[label]["stack_gb"] = {**m_cv, "tau": tau_cv, "base_rate": float(y_meta.mean())}
1368
1369        # Holdout to player meta set (evaluate)
1370        sub = ph.loc[ph["label"] == label, ["y_true","proba"]].copy().reset_index(drop=True)
1371        sub.rename(columns={"y_true": label}, inplace=True)
1372        sub[PLAYER_KEY_NAME] = player_holdout.values
1373        sub["g"] = g_holdout.values
```

```
    sub["total_minutes"] = tm_holdout.values

    df_ho_players = _player_stack_features(sub, label, PLAYER_KEY_NAME)
    X_meta_te = df_ho_players[["mean_p","max_p","min_p","std_p","count_seasons","mean_games","mean_minutes","top2_mean"]].to_numpy()
    y_meta_te = df_ho_players[label].to_numpy()
    p_meta_te = meta.predict_proba(X_meta_te)[:, 1]
    m_te = _metrics(y_meta_te, p_meta_te)

    if label not in test_store_rf:   test_store_rf[label] = {}
    if label not in test_players_rf: test_players_rf[label] = {}

    test_store_rf[label]["stack_gb"] = dict(
        auc=m_te["auc"], pr=m_te["pr"], brier=m_te["brier"], logloss=m_te["logloss"],
        tau_oof=tau_cv, base_rate=float(y_meta_te.mean())
    )
    test_players_rf[label]["stack_gb"] = df_ho_players.assign(p_player=p_meta_te)

    summary_rows.append({
        "label": label, "aggregation": "stack_gb",
        "AUC_CV": m_cv["auc"], "PR_AUC_CV": m_cv["pr"], "Brier_CV": m_cv["brier"], "LogLoss_CV": m_cv["logloss"],
        "tau_OOF": tau_cv, "base_rate_CV": float(y_meta.mean()),
        "AUC_Test": m_te["auc"], "PR_AUC_Test": m_te["pr"], "Brier_Test": m_te["brier"], "LogLoss_Test": m_te["logloss"],
        "base_rate_Test": float(y_meta_te.mean()),
    })

# Save stacked summary
stack_sum_df = pd.DataFrame(summary_rows).sort_values(
    by=["label","AUC_Test","PR_AUC_Test"], ascending=[True, False, False]
).reset_index(drop=True)
stack_sum_df.to_csv(OUT_DIR / "stack_gb_summary_rf.csv", index=False)
print(f"Saved stacked GB summary -> {OUT_DIR/'stack_gb_summary_rf.csv'}")
stack_sum_df.head(25)


# %%
# F5: Holdout confusion matrices at OOF thresholds for selected models

import numpy as np, pandas as pd
from sklearn.metrics import confusion_matrix, precision_recall_fscore_support

def _confusion_at_tau(df_player, label_col, tau):
    y = df_player[label_col].to_numpy()
    p = df_player["p_player"].to_numpy()
    yhat = (p >= float(tau)).astype(int)
    tn, fp, fn, tp = confusion_matrix(y, yhat, labels=[0,1]).ravel()
    prec, rec, f1, _ = precision_recall_fscore_support(y, yhat, average="binary", zero_division=0)
    return dict(tn=int(tn), fp=int(fp), fn=int(fn), tp=int(tp),
                precision=float(prec), recall=float(rec), f1=float(f1),
                base_rate=float(y.mean()))

# Selection list
selections = [
    ("impact4_ws",   "stack_gb"), ("impact4_ws",   "uniform"),
    ("impact4_vorp", "stack_gb"), ("impact4_vorp", "uniform"),
    ("rotation",     "uniform"),
    ("survival",     "uniform"),
    ("starter",    "uniform"),  # optional; currently skipped
]

rows = []
missing = []

for label, mode in selections:
    # Presence checks
    if label not in cv_store_rf or mode not in cv_store_rf[label]:
        missing.append(f"cv_store_rf[{label!r}][{mode!r}]")
        continue
    if label not in test_players_rf or mode not in test_players_rf[label]:
        missing.append(f"test_players_rf[{label!r}][{mode!r}]")
```

```
1443            continue
1444
1445        tau = cv_store_rf[label][mode].get("tau", cv_store_rf[label][mode].get("tau_oof", 0.5))
1446        dfp = test_players_rf[label][mode]
1447        res = _confusion_at_tau(dfp, label, tau)
1448        rows.append({
1449            "label": label, "mode": mode, "tau_used": float(tau),
1450            **res
1451        })
1452
1453    confusions_df = pd.DataFrame(rows).sort_values(by=["label","mode"]).reset_index(drop=True)
1454    print(confusions_df.to_string(index=False))
1455
1456    # Optionally save
1457    confusions_df.to_csv(OUT_DIR / "holdout_confusions_selected_rf.csv", index=False)
1458
1459    if missing:
1460        print("\ n Missing artifacts for:", "; ".join(missing))
```