ChatGPT

Introduction

NFL proposition bets (“props”) are wagers on specific player or team outcomes – from an individual’s receiving yards or longest catch to a team’s total points or win probability. A variety of modeling approaches are used to **generate prop betting lines** (i.e. predictions of these outcomes). These range from simple statistical models to sophisticated machine learning ensembles and simulation-based methods. Below we explore different methods and tools for creating NFL prop lines, including publicly available models, academic/statistical techniques, community/DFS approaches, coding libraries, and even no-code solutions. We also note what data each method requires and provide links to example projects where possible.

Publicly Available Models and Baseline Approaches

Elo Rating Models: One well-known public model for **game props** (like win probability or point spreads) is the Elo rating system popularized by FiveThirtyEight. Elo assigns each team a **power rating** that updates after every game based on wins/losses and the score margin. The difference in two teams’ Elo ratings (plus adjustments for factors like home field, quarterback changes, rest, etc.) can be converted into a win probability for a matchup [1] [2]. FiveThirtyEight openly publishes its Elo methodology and even provides code/data for their NFL Elo model, which produces win probabilities for each game [3]. This model mainly requires *historical game results* (scores, winners, etc.) as input. An independent analyst can replicate Elo ratings fairly easily or tweak the parameters to fit their own view. (In fact, FiveThirtyEight’s GitHub invites people to modify Elo and see how it affects predictions [4].) Elo is a **baseline** that captures team strength and can be used to set moneyline odds or even approximate point spreads (FiveThirtyEight notes that dividing Elo rating difference by 25 gives an implied point spread) [5].

Simple Averages and Expert Projections: For **player props**, a baseline method is using historical averages or publicly available projections. For example, one could take a player’s average receiving yards per game (perhaps weighted towards recent games) as a rough “line”. Many fantasy football sites publish weekly player projections; these can be repurposed as prop estimates. Some public models aggregate expert opinions – e.g. FantasyPros consensus projections – which independent bettors use as starting points. While simple averages ignore context, they require minimal data (just the player’s past game logs or expert estimates). They can be improved by adjusting for opponent strength (e.g. increasing a RB’s rushing yards projection if facing a weak run defense).

Open-Source Prop Models: There are also fully **open projects** focusing on props. For instance, *The Quant’s Playbook* (Quant Galore) published code for an NFL player **receptions prop model** on GitHub [6] [7]. Their approach predicts a player’s reception totals and identifies betting edges. Similarly, community repositories on GitHub or Kaggle often share models for game outcomes or player stats (e.g. a student project using ensemble models to predict NFL game winners [8]). These public models typically need rich datasets (player stats, team stats, etc.), but they provide templates that independent bettors can study or adapt. Even if one doesn’t run the code, the existence of these models helps validate that it’s possible to beat certain prop lines with data-driven predictions [9]. Public models highlight features or techniques that work – for example, incorporating sportsbooks’ own lines as inputs or using third-party ratings like

FiveThirtyEight's ELO as features [10] [11]. In summary, leveraging public models and data (when available) is a great way to bootstrap your own line-making process with proven baselines.

Academic and Statistical Methods

Regression Models: A classic statistical approach to prop lines is to build a regression model that predicts the quantity of interest from historical data. For example, one could use **multiple linear regression** to predict a player's receiving yards given inputs like their season average, the opposing defense's average yards allowed, weather conditions, etc. The model's output (a continuous numerical prediction) serves as the projected **over/under line**. A simple version might be:

- *Player receiving yards* = β_0 + β_1*(player's recent avg yards)* + β_2(opponent pass defense rank) + β_3*(team expected points) + ... + ϵ.

This requires **historical game-by-game stats** for players and teams as data. Regression models are relatively easy to implement (even in Excel or with statistical software) and their coefficients can offer interpretable insights (e.g. how much an elite defense reduces expected yards). However, linear models assume a roughly linear relationship and may not capture complex interactions (like a defense being uniquely good against tight ends). For *team props* like total points, analysts sometimes use **regression on team offense/defense metrics** – e.g. predict combined score from each team's average scoring and allowing tendencies (plus perhaps a term for pace or matchup effects). Regression models work best when you have enough data and the relationships are linear or at least monotonic. They typically need *season or multi-season statistics*, possibly normalized for league averages.

Poisson and Distribution Models: Many academic analyses treat scoring events as counts governed by statistical distributions. The **Poisson distribution** is often used to model counts of events (it's famously applied to soccer goals). In NFL betting, Poisson models can estimate probabilities for props like *"will a QB throw an interception?"* or *"over/under 2.5 touchdowns"*. The idea is to use a player or team's average rate as the Poisson mean (λ). For example, if a quarterback historically throws **0.8 interceptions per game**, you can model the probability of 0, 1, 2, ... interceptions in a given game as Poisson(λ=0.8). The chance he throws **at least 1 INT** is then 1 – P(0) = 1 – e^(-0.8) ≈ 55%. This provides an implied "yes/no" prop probability which you compare to the odds on offer. As one tutorial noted, given a QB's career stats, it's straightforward to decide if a bet like *over 1.5 TD passes* is good value using a Poisson model [12]. Poisson models assume events occur at a constant rate and independently, which is a simplification (NFL scoring has context and correlation), but they often yield a decent baseline. They require only *aggregate stats* (mean rates of the event for the player/team, and possibly for the opponent's defense to adjust the rate).

For **total points props**, a common approach is to treat each team's score as a random variable – sometimes modeled by Poisson or a related distribution – and derive the distribution of their sum. For instance, you might estimate Team A's expected points and Team B's expected points (using season averages adjusted for each other's defense), assume each follows a Poisson, and then convolve them to get a distribution for total points. Some models use a **bivariate Poisson** or correlated Poisson approach (since team scores aren't entirely independent). In practice, even if the distributional assumption isn't perfect, using Poisson estimates can help calculate the probability of the game going over or under a certain total. Academically, researchers have also applied **Poisson regression** (a GLM) to relate team scoring to predictors like offense/defense ratings [13]. The key data needed are teams' scoring stats (points for/against, possibly broken down by situation).

Logistic Models: For binary outcomes (yes/no props, e.g. *"Will Team X make the playoffs"* or *"Will Player Y score a TD"*), logistic regression is a popular statistical method. A logistic model can output the probability of the event as a function of various features. For example, to model the probability a QB throws an interception in a game, you might use logistic regression with features like the QB's average interception rate, the opposing defense's interception rate, game setting (home/away, favorite/underdog), etc. The model can be trained on historical game data labeled with whether an INT occurred. This approach needs a dataset of past games with the outcome of interest (e.g. a binary column for "threw INT") and relevant predictor variables for each game. Academic sports analytics often uses logistic models for win probability and fourth-down decisions; those same techniques translate to prop probabilities. Logistic models are interpretable (coefficients show how a factor shifts odds) and can account for non-linear effects through transformations or interaction terms.

Time-Series and Bayesian Methods: Another angle is treating prop lines as forecasts in a time-series sense. For example, a player's weekly yardage could be forecasted using an ARIMA model or exponential smoothing that accounts for recent trends (this requires a *time series of the player's game logs*). However, NFL player data can be sparse (limited games per season), so time-series methods are less common than cross-sectional models using many players/games. Bayesian methods, on the other hand, have gained traction in academic circles: one might build a **Bayesian hierarchical model** where each player's performance is modeled with a prior distribution (reflecting overall league averages) updated by that player's data. For instance, you could use a Bayesian model to estimate each kicker's field goal success probability, pooling information across the league (this was done academically for **field goal models**). Bayesian models can incorporate uncertainty in a principled way and output full distributions for a prop. The downside is they can be computationally intensive and require expertise (often implemented via MCMC in tools like PyMC or Stan). They need similar data as regression models but allow you to include prior knowledge (e.g. a highly drafted rookie WR might have a prior expected yards based on college performance).

In summary, academic/statistical approaches provide a toolkit of relatively **transparent models** – from linear and logistic regression to Poisson count models – that an independent analyst can implement with moderate effort. These methods typically demand a solid dataset of historical observations (games or seasons) for fitting, but once trained they can output an expected value or probability for the prop in question. They serve as a foundation that can be enhanced with more advanced techniques or domain-specific tweaks.

Machine Learning and Ensemble Models

More advanced approaches in recent years use **machine learning (ML)** to predict prop outcomes. ML models can capture nonlinear relationships and interactions in the data, often leading to more accurate predictions at the cost of interpretability. A prime example is the ensemble model from *The Quant's Playbook* for predicting player props. In their NFL receptions prop project, they built an **ensemble of tree-based models and a neural network**, averaging their predictions to get a robust expected value [14] [15] .

Tree-Based Models: Decision tree algorithms like **Random Forests** and **Gradient Boosting (XGBoost)** are popular for sports props. These models automatically find splits in the data – essentially learning which factors (and what thresholds) matter for predicting the outcome. For instance, a regression tree predicting *longest reception yards* might first split on something like "Was the opponent's pass defense top-5 in the league?" and then on "Does the QB average ≥250 passing yards?". Tree models handle nonlinear effects (e.g. a dome vs. snow game) and interactions (QB–receiver chemistry under certain conditions) without manual feature engineering. As the Quant's Playbook article notes, **XGBoost and Random Forests** are the "de-facto leaders" in tree-based regression and were used for predicting continuous prop metrics like yardage [16]. A random forest is essentially an ensemble of many decision trees averaged together, which reduces overfitting and usually improves accuracy. Gradient boosting (e.g. XGBoost or LightGBM) builds trees sequentially, each one correcting errors of the previous, to produce a strong predictive model. These models typically require a **rich feature set**: you feed in as many relevant variables as you can – player stats, team stats, matchup data, weather, etc. – and let the algorithm figure out the relationships. They also need a considerable amount of data for training; usually one would gather multiple seasons of play-by-play or game-level data for all players to train a general model (or at least several seasons of data for the specific player/prop type). The output is a prediction (e.g. expected yards or probability of event) which can directly serve as a projected line. Notably, tree models can also provide a measure of feature importance, telling you which inputs were most predictive (though not as cleanly as a linear model's coefficients).

Neural Networks: Neural networks are another ML approach used in sports modeling, though they are somewhat "black boxes." A neural network can model very complex patterns by learning weights for many interconnected layers of "neurons." In practice, one might use a neural network to predict something like a player's fantasy points or yards by feeding it all the same inputs (opponent, player stats, etc.) and letting it train through backpropagation. The Quant's ensemble included a neural network alongside tree models [15]. The network essentially tries to find an optimal combination of features through its hidden layers, potentially capturing interactions that trees might miss. However, neural nets require even more data to generalize well – typically tens of thousands of examples – and careful tuning (to avoid overfitting). In NFL prop contexts, neural nets are often used in combination with other methods (an ensemble) to hedge against any single model's weaknesses [17]. For an independent bettor, implementing a neural network is doable with libraries like TensorFlow or PyTorch, but interpreting the results can be challenging. It's often overkill unless you have a very large dataset (e.g. years of play-by-play data or a synthesized dataset of simulated games). Still, some public efforts exist (e.g. a GitHub project using a feed-forward network to predict NFL game outcomes).

Feature Engineering: Regardless of ML model type, a crucial step is constructing useful **features** from raw data. Sports data is messy and needs context – for example, a raw stat like "5 receptions last game" means different things if the QB threw 50 passes vs. 20 passes. In the receptions prop model, they explicitly engineered features capturing the **QB–receiver relationship** (how often a QB targets that player, and at what depth) and the conditions (weather, venue) [18] [19]. They even ask complex conditional questions – e.g. *"when wind > 10 mph and temp < 60°F, who does the QB favor and are passes shorter?"* [20]. These insights become features such as "target share in cold/windy games" or "average air yards in bad weather." Good feature engineering often requires *play-by-play data* (to know situational stats) or at least splitting stats by conditions (splits for indoor vs. outdoor, etc.). With comprehensive datasets like those from the **nflverse** (which provides play-by-play and aggregated stats in R/Python form [21] [22]), independent analysts can create dozens of features: e.g. offensive pace (plays per game), red zone targets, YAC (yards after catch) allowed by the defense, and so on. ML models will benefit from these rich inputs and can handle the complexity of many correlated features.

Ensembling: A hallmark of modern ML approaches is ensembling – combining multiple models to improve stability and accuracy. Each model might have different biases; by averaging or blending their predictions, we get a consensus that is often better than any individual model. In prop betting context, one might ensemble a linear model, a random forest, and a gradient booster, for example. The Quant's model averaged **three** algorithms (two tree-based and one neural net) to get a single predicted value [14] [15]. The idea is that if all models agree on a certain player's expected yards, we can be more confident in it; if one model is an outlier, the averaging mitigates its impact. Ensembling does require running/training multiple models, which is computationally heavier and demands more work to maintain. But for independent analysts, even something as simple as **"wisdom of the crowd" averaging** can be powerful – e.g. average the predictions from a regression model and a Poisson model, or average projections from several different sources (your model + a public projection). This is essentially a manual ensemble. Many DFS analysts do this by averaging projections from 3-4 fantasy sites to get a more robust estimate for player performance.

In summary, machine learning models (random forests, gradient boosting, neural nets, etc.) can capture complex relationships in NFL data to generate prop lines. They typically require **large datasets** (multiple seasons of data for all teams/players relevant to the prop) and computational tools, but they offer potentially higher accuracy. They work best when combined with thoughtful feature engineering and often with each other (ensembling). For an independent bettor, using ML means moving into coding (Python/R) and leveraging libraries, but the payoff is a model that can find subtle edges (e.g. a receiver's yardage being underestimated because the model learned a pattern about that defensive scheme). Some publicly shared ML models (like open GitHub repos) can jump-start this process, providing code for data prep and even pretrained models.

Simulation and Monte Carlo Techniques

A powerful complement to the above methods is **Monte Carlo simulation**. Simulation involves generating random outcomes based on a model or empirical distribution to assess probabilities of various prop results. Many bettors use simulation especially for props that depend on multiple uncertain components (e.g. yards = targets × yards per catch, etc.). The approach taken by Unabated's team in a recent tutorial is a great example: they built a basic player prop projection (for a tight end's receiving stats) and then **simulated it 10,000 times** using the Unabated NFL Prop Simulator [23]. The simulation produces a full **distribution** of outcomes, from which you can derive the probability of clearing an over/under line or the fair odds for a yes/no prop.

Bootstrapping from Data: One simulation method is to resample historical performance to create possible game outcomes. For instance, if you have 10 past games of a player, you might randomly pick one of those games (or one of those opponents, etc.) to simulate the next game. More sophisticated, you might bootstrap **play-by-play sequences** or drive outcomes to simulate a full game. Unabated's mention of *bootstrapping and weighting* suggests they take past game data, weight it (more weight to recent or relevant games), and sample from it to simulate a new game's stats. This requires detailed data (either game-level or play-level). The benefit is that it preserves the distributional shape – e.g. if a player is boom-or-bust, bootstrapping past games will reflect that variance in the simulation.

Distribution-Fitting: If you don't want to resample actual games, you can assume a statistical distribution for the performance. For example, one might model a receiver's yardage as a **log-normal** distribution (since yardage can't be negative and is skewed) characterized by an estimated mean and variance. After estimating those parameters (e.g. using a weighted average and standard deviation of past outcomes [24]), you can simulate many random draws from that distribution. Alternatively, some break the problem into parts: simulate the **number of catches** (which is a count, maybe Poisson-binomial) and the **yards per catch** (which could follow a gamma distribution), then multiply for total yards. By running thousands of simulations, you can count how often the total exceeds a line like 55.5 yards to get the win probability of the over. This approach requires an understanding of the underlying stats – e.g. knowing the variance is as important as the mean. As the Unabated guide emphasizes, incorporating the **standard deviation** (spread of outcomes) is crucial, not just the mean [24].

Correlated Simulations: For game props like total points or win probability, Monte Carlo simulation can incorporate correlation between teams. For example, you might simulate an entire game by simulating each team's score (from a distribution or via drive-by-drive simulation). If you simulate play-by-play, you naturally capture that one team's offense being hot might coincide with more scoring opportunities for the opponent (or conversely, if one team dominates time of possession, the total might stay low). Some advanced simulators (like those used by DFS optimizers and betting tools) simulate games 10,000+ times using **play-by-play models** or **drive outcome models**. These often leverage the NFL's play-by-play datasets (like those available in the nflfastR package) to estimate transition probabilities (chance of a drive resulting in a TD, FG, etc.). For an independent bettor, building a full game simulator is a big project, but it's not unheard of – open-source projects exist (e.g. an **nflseedR** package in R can simulate entire seasons using win probabilities [25]). More accessible are tools like **Unabated's simulators** which allow you to plug in your projections and run Monte Carlo in a user-friendly interface [23].

The result of simulation is typically a more **granular understanding** of the prop. Instead of just "I think the fair line is 60 yards," a simulation will tell you "there's a 52% chance he goes over 58.5 yards, and a 10% chance he goes for 100+ yards," etc. This helps in pricing alternate lines or understanding tail risks. For example, a yes/no prop ("will he throw an INT?") is essentially asking for the probability of at least one occurrence; if you simulate a QB's pass attempts and interception probability per attempt, you can estimate this probability more dynamically than a simple Poisson (e.g. accounting for the possibility of 40 pass attempts in a come-from-behind scenario vs 20 in a blowout).

In summary, Monte Carlo simulation is a versatile approach that builds on a predictive model (from statistical or ML methods) by adding the element of **random variability**. It requires either a model of variance or an empirical distribution of outcomes. With modern computing, even a laptop can simulate thousands of game scenarios in seconds. Independent analysts who are comfortable with a bit of coding can use libraries like NumPy (for random draws) or even specialist tools (Unabated's simulator, or custom scripts) to run these simulations. The result is an **actionable probability distribution** for any prop, which is the key to identifying bets with positive expected value. (After all, a sportsbook's line is essentially a statement about the probability distribution – by simulating, you're checking if your view of that distribution differs from the market's.)

Sports Analytics Techniques in DFS and Betting Communities

The daily fantasy sports (DFS) community and sports betting forums have developed their own pragmatic techniques for projecting player performance – techniques that can be applied to prop betting. These approaches are often **"bottom-up"**: projecting the components of a stat line, then combining them, rather than predicting the outcome in one go. They also lean on shared data and collective wisdom. Here are a few key techniques from the DFS/betting community:

Bottom-Up Player Projections: In DFS, players often project a stat like receiving yards by breaking it down: $\textit{targets} \rightarrow \textit{receptions} \rightarrow \textit{yards}$. For example, one might project that a receiver will get 8 targets, catch 5 (assuming a catch rate ~63%), and average 12 yards per catch – yielding ~60 yards. Each of those inputs can be derived from data: targets might be predicted from the team's pass attempts and the player's target share, receptions = targets × catch%, yards per catch based on the player's average depth of target and YAC.

This bottom-up approach is very **actionable** for independent analysts, since you can find many of these stats on sites like Pro Football Reference or via the nflverse data. It requires thinking in terms of **rates and shares**: e.g. team plays per game, pass play percentage, target distribution among receivers, etc. Bettors often adjust these components for the matchup – e.g. if facing a strong secondary, maybe reduce expected catch rate or yards per catch; if the game is projected to be high-scoring (per the Vegas total), maybe bump up the team's total plays and thus opportunities. By tweaking these factors, you effectively build a custom projection. Tools like Excel or Google Sheets are commonly used in the community to organize these calculations for each player. This method uses broadly available data (averages, percentages) and some subjective adjustments, but it's transparent and easy to update.

Incorporating Vegas Lines: A clever technique widely used is to leverage the information in the betting lines themselves as inputs. For instance, the Vegas **over/under and point spread** for a game tell a story about expected game script: a high total (e.g. 52 points) implies more yardage and scoring to go around; a big spread implies one team might be ahead (affecting rushing vs passing volume). DFS projections often start by allocating the sportsbook's projected points to players. Suppose a team's implied total is 28 points; one might distribute those across the skill players (e.g. 3 TDs to the QB's passing, 1 to the RB rushing) based on usage, and ensure the yardage totals roughly match 28 points worth of offense (around 400 yards, say). This anchors the projection in a baseline that is **market-consistent**. Independent bettors can do this without coding – simply read the Vegas lines and use them as a guide for team-level expectations, then drill down to players. Another example: If the market expects Team A to have 300 passing yards (perhaps inferred from a QB's prop line or from total yards props), and you're projecting a receiver's yards, make sure the sum of all receivers' projected yards is in that ballpark. Essentially, you **reverse-engineer** prop lines and spread totals to maintain consistency. This approach doesn't require extra data beyond what's publicly posted by sportsbooks, but it does require careful bookkeeping to align your projections with those implied totals (or deliberately differ if you have an edge case).

Community Projections and Consensus: The betting and DFS community often shares projections or at least their analysis through forums (like Reddit's r/sportsbook or r/sportsanalytics) and on Twitter/Discord. One common practice is to aggregate or compare multiple projection sources. For example, you might take the average of projections from FantasyLabs, NumberFire, and an independent model. If one doesn't want to build a model from scratch, aggregating several freely available projections is an **actionable shortcut**. Many bettors also use **consensus rankings** (from fantasy football) to derive prop expectations – e.g. if multiple analysts rank a player highly for the week, it's likely his yardage will be high. Some community-driven projects (like open Google Sheets) compile such information weekly. While not "models" in the strict sense, these are tools to generate a reasonable line: if five different sources all project around 65–70 yards for a player, that consensus can be treated as a makeshift expected value for his yardage prop. It's then up to the bettor to decide if the sportsbook line diverges from that.

Heuristic Angles and Adjustments: The community also uses a lot of heuristic or situational angles that can refine prop lines. For example: "Receiver X performs better at home" – one might bump his projection a bit at home games (if data supports it). Or matchup-based tweaks: "Team Y funnels passes to running backs, so the RB's receiving yards should be above his baseline this week." These are not full models, but rules of thumb derived from watching games or looking at split stats. DFS analysts are famous for uncovering such trends (sometimes using data mining, sometimes anecdotal) – e.g. a certain cornerback's coverage might limit a #1 WR, leading one to downgrade that WR's yards prop. Independent bettors can incorporate these community insights by following sports analytics Twitter accounts, listening to podcasts, or reading forums where such ideas are discussed. They then adjust their prop lines accordingly. The data needed here can vary – sometimes it's *specific split stats* (like vs certain coverage schemes, which sites like PFF or NextGenStats might provide), but often it's just a matter of accounting for a qualitative factor (e.g. weather: extreme wind -> reduce passing yards prop).

Comparing to the Market for Value: Finally, a technique used both in DFS and betting is to **compare one's projections to the market lines to identify value**. Communities often talk about "edges" in terms of how far off a projection is from the posted line. For instance, BetQL (a betting analytics platform) describes that their model projects a stat for every player and then **compares it to the sportsbook's line** – the bigger the difference, the higher they rate that bet [26]. Independent bettors do the same: once you have a projected line (via any method above), you see if it's higher or lower than the sportsbook's number. If you projected 70.5 yards and the book is offering over/under 62.5, you might see a significant **over** edge. It's important to incorporate the implied odds too (an over at -120 needs a higher true probability to be value than one at +100). Communities often share when their projections find large discrepancies, although one should be cautious – sometimes the market knows something your model doesn't (e.g. an injury, or a scheme change). Nonetheless, this comparison step is crucial and essentially turns your **line generation** into actionable betting decisions. It doesn't require new data, just the discipline to regularly fetch odds (manually or via an API) and cross-check them against your numbers. Some bettors even automate this, flagging any prop where the difference is beyond a certain threshold, akin to what BetQL's star ratings do [27].

In essence, DFS and betting community techniques emphasize **practical, replicable processes**: breaking stats into components, using market info as a guide, crowdsourcing projections, and constantly comparing to actual lines. They may not be as theoretically pure as academic models or as complex as ML, but they are effective and relatively easy to implement with publicly available information. Independent analysts can often achieve a lot by following these community methods – they're the kind of approaches where you can start with a spreadsheet and gradually refine as you gather more data or find better heuristics.

Tools and Libraries for Custom Modeling (Python & R)

If you choose to build custom models or simulations, there is a rich ecosystem of **tools in Python, R, and other languages** to support sports data analysis and modeling. Here are some notable ones:

- **Data Access Libraries (nflverse):** A fundamental need is historical data – fortunately, the **nflverse** collection in R (and its Python ports) makes this easy. The nflverse is a set of packages dedicated to NFL data, including `nflfastR` (for play-by-play data with advanced metrics like EPA/WPA), `nflreadr` (to easily download cleaned datasets), and `nflplotR` (for visualization) [28]. Most nflverse data (e.g. game logs, play-by-play, betting lines, etc.) is stored in accessible repositories, and there's even a Python library `nfl-data-py` for pulling this data into Python scripts [22]. This means an independent analyst can quickly gather the raw material needed for modeling – e.g. you can get a DataFrame of every rush attempt in 2022 with one function call, or a table of every game's closing spread and total. Having data at your fingertips is half the battle; these libraries remove the need to manually scrape websites.

- **General-Purpose ML Libraries:** For building models, the standard Python libraries apply: **pandas** for data manipulation, **scikit-learn** for a wide range of regression, classification, and clustering algorithms, **statsmodels** for more traditional statistical modeling (e.g. linear regression, ARIMA, logistic regression with detailed output), and **scipy** for distributions and statistical functions. In R, the **tidyverse** (dplyr, tidyr, etc.) helps wrangle data, and packages like **glmnet** (for regularized regression), **randomForest** or **xgboost** (interfaces to those ML models), and **brms** or **rstanarm** (for Bayesian regression) are widely used in sports analytics. These tools aren't sports-specific but are well-documented and have large communities. For example, using scikit-learn, you could set up a pipeline to train a random forest on historical props data in just a few lines of code. Many Kaggle notebooks and blog posts demonstrate how to use these libraries for sports data; if you have a basic programming background, adapting them to your prop problem is quite feasible.

- **Specialized Sports Modeling Packages:** There are emerging libraries aimed specifically at sports betting modeling. One example is the **sports-betting** Python package by Georgios Douzas [29]. It's a toolkit that provides functionalities to **fetch sports data, build predictive models, backtest betting strategies, and even a simple GUI** to interact with models. While it originally focused on soccer, it's expanding to other sports and offers a framework for things like value bet identification. Such packages often include utilities to compute implied probabilities, to handle odds formats, and to evaluate model performance in betting terms (like ROI or Kelly criterion). Another specialized library is **pyDBL** or **pySports** (hypothetical examples for demonstration), which might have modules for Elo ratings or Poisson models tailored to sports. While not all sports have dedicated libraries, the combination of general libraries with domain data packages (like nflverse) covers most needs.

- **Optimization and Simulation Tools:** If your modeling extends to simulating games or optimizing portfolios of bets, you might use libraries like **NumPy** and **simpy** (for custom simulations), or even **PuLP** or **ortools** if you're optimizing bankroll or DFS lineups. For example, once you generate probabilities for various props, you could use an optimizer to allocate your bankroll to the most +EV (expected value) bets subject to constraints – this veers into betting strategy, but it's something Python can handle with linear programming solvers. In R, packages like **ROI** or **ompr** could do similar optimization. These are beyond generating lines but are useful if you want to turn your prop lines into a full betting strategy tool.

- **APIs and Data Sources:** To keep your models up-to-date, you may want to integrate with external APIs for injuries, depth charts, or real-time odds. For instance, the **Prop-Odds API** (prop-odds.com) provides current prop lines from sportsbooks; the Quant's Playbook project used it to fetch market odds for comparison [7]. There's also the NFL's official API (undocumented but used by some developers) and community APIs like **sportsdata.io** or **The Odds API** that provide schedules, stats, and odds. Python's `requests` library or R's `httr` can be used to pull data from these APIs and feed your model. Many independent analysts automate daily data pulls so their models always have fresh inputs (like updated player stats and latest betting lines).

In short, a technically-inclined bettor has a wealth of libraries to build their own modeling pipeline. You might start in R for quick analysis with nflverse data and glm models, then switch to Python for deploying a more complex model with scikit-learn – whatever suits your workflow. The key is that **you don't have to build everything from scratch**: data collection is largely solved, and implementation of algorithms is one import away. The barrier to entry for creating a custom NFL prop model is lower than ever thanks to these tools. The main effort is in designing the model logic and interpreting the results – the libraries handle the heavy lifting of computation and data processing.

No-Code and Low-Code Solutions for Betting Line Prediction

Not everyone wants to code their own models from the ground up. Fortunately, there are **no-code or low-code tools** emerging in the sports analytics space that can help generate prop betting lines or at least assist in that process:

Rithmm – Custom Models via UI: *Rithmm* is a recent AI-driven sports betting platform that allows users to create personalized models with minimal technical effort. The idea is you can choose what factors you think are important – for example, emphasize a team's offensive strength vs. defensive strength, account for weather, etc. – and Rithmm's backend will produce predictions based on those choices [30]. In the context of NFL, Rithmm provides over/under predictions for games and even player props, powered by historical data and machine learning under the hood [31]. A key feature is the ability to **tweak the model without coding**: adjust sliders or inputs for offensive vs defensive emphasis, include or exclude certain player performance factors, and instantly see how the predicted totals or probabilities change [32]. Essentially, Rithmm is packaging a lot of the techniques we discussed (data, models, simulations) behind a user-friendly interface. For an independent bettor, using Rithmm means you can get model-driven insights without writing a line of code – you focus on strategy and intuition (which factors to stress) and let the platform handle data processing and algorithmic predictions. Rithmm's outputs include things like recommended over/under bets where their model finds a discrepancy from the market. It's a paid service, but it exemplifies the "no-code" model ethos: *custom analytics at the click of a button*.

Unabated Prop Simulator: We mentioned Unabated's prop simulation in the context of methodology, but as a tool it is very much a low-code solution. Unabated (founded by pro bettors) offers a suite of tools like an **NFL Props Simulator**, **Line Comparison**, **Arbitrage Calculator**, etc., accessible through their website [23] [33]. The Prop Simulator lets you input your own projection and standard deviation for a player, then run simulations to output the probability of various outcomes (like hitting the over). This means you don't need to program the Monte Carlo yourself – you just supply the assumptions. Unabated also provides default stats and sliders to adjust if you don't have your own model's number. It's *low-code* in that the heavy statistical lifting is pre-built; you operate through a form or dashboard. Similarly, their **NFL Season Simulator** takes team ratings and simulates entire seasons (useful for win total bets or playoff odds). These tools require a subscription for full features, but they're built exactly for bettors who want analytical rigor without having to be data scientists. Essentially, Unabated is providing the *calculators* so you can focus on betting decisions. The only "data" you need is whatever input values you believe (which you might get from simple research or gut feeling), and the tool handles the rest (running 10k sims, etc.).

Spreadsheet Models and Templates: A more old-school but still prevalent low-code tool is the **Excel model**. There are many Excel templates and Google Sheets available (often shared on betting forums or sold by handicappers) that allow you to plug in a few assumptions and see projected lines. For instance, someone might share a "NFL Prop Model in Excel" that has tabs where you enter each team's offensive stats, defensive stats, maybe some pace metrics, and it will output predicted scores and even player stat lines using built-in formulas. Excel has surprisingly powerful features – you can do regressions with the LINEST function or the Analysis ToolPak, solve for rates using Goal Seek, and of course use innumerable formulas to create custom metrics. A lot of independent bettors start by building a spreadsheet that pulls in data (perhaps via Power Query or pasted from websites) and then uses a series of calculations (maybe based on the regression they ran or a Poisson formula) to spit out an expected prop line. This is effectively **no-code** since Excel's interface is doing the computation. The drawbacks are that complex logic can get unwieldy in a spreadsheet, and scaling to lots of players/games is manual. However, it's worth noting that many professionals first prototyped models in Excel before porting to code. If you prefer point-and-click, Excel or Google Sheets is a viable path – and the community often shares tips on using these for sports (e.g. using Excel's built-in data types to fetch player stats, or using Power Query to get ESPN projections automatically).

Automated Machine Learning (AutoML): Outside of sports-specific tools, there are general AutoML platforms like Google's AutoML Tables or DataRobot that can take a dataset and automatically try a bunch of models to make predictions. An independent bettor with a compiled dataset (say, a CSV of past games with features and outcomes) could theoretically feed it into such a platform and get a predictive model without manually coding algorithms. This is more "low-code" as you still have to prepare the data and interpret the results, but the actual model training/tuning is handled by the platform's AI. While these aren't tailored to NFL, they might discover patterns a simple model would miss. The limitation is these services are often enterprise-focused (they may be costly or have usage limits) and you need enough data to make them effective. For a hobbyist, something like the **sports-betting GUI** mentioned earlier [29] (which provides a friendly interface to upload data, select model types, and see backtest results) might be more accessible – it's like a mini AutoML specifically for betting.

Community Platforms: There are also community-driven tools such as betting **Discord bots** or apps that surface projections. For example, some Discord servers have bots where you can type a command like "!proj Mahomes yards" and get a projection pulled from a consensus or a specific model. These are based on underlying code, of course, but from the user perspective it's no-code. Even some Twitter accounts automatically post prop lines vs. projections using scripts; following them can be an easy way to get model insights without running a model. And sites like BetQL (as cited) or Action Network provide prop projections and calculators as part of their product – effectively outsourcing the modeling to them. No-code bettors can leverage these by simply using the information: if BetQL says a certain prop is five stars (meaning their model shows a big edge), you can decide to follow that rather than compute your own probability. Caution is advised to trust but verify such sources, but they are available.

In summary, no-code/low-code tools lower the barrier to entry for analytics-driven betting. Whether it's a specialized platform like Rithmm, an educational toolset like Unabated, or just a well-crafted Excel sheet, these options allow independent bettors to **replicate sophisticated methods without programming**. They are ideal for those who understand the concepts and strategies but aren't as comfortable with coding or data science. By using these tools, one can still apply the principles of modeling, simulation, and data-driven decision-making – which ultimately leads to more informed prop betting – without needing to build everything from scratch. As the sports betting tech ecosystem grows, we can expect even more user-friendly analytical tools to emerge, making it possible for anyone (with enough football knowledge and curiosity) to generate their own NFL prop lines and uncover value bets.

Conclusion

Generating NFL prop betting lines can be approached from multiple angles, each with its own data requirements and complexity. Public models (like FiveThirtyEight's Elo or open GitHub projects) provide baseline predictions using readily available data [3]. Academic and statistical methods (regressions, logistic/Poisson models, etc.) rely on historical stats and offer interpretable, repeatable frameworks grounded in math [12]. Advanced machine learning techniques (tree ensembles, neural nets) leverage large datasets and computational power to squeeze extra predictive accuracy, especially when combined in ensembles [16] [15]. Meanwhile, the DFS and betting community contributes practical know-how – from breaking down projections piece by piece to using market lines as guideposts – that independent bettors can apply with little more than spreadsheets and diligence. Modern tools in Python/R make it easier than ever to implement custom models or run simulations, and if coding isn't your forte, new no-code platforms and traditional Excel-based models have you covered.

In practice, many successful analysts blend these approaches. For example, one might use a quick regression model to get a baseline, refine it with a bit of machine learning for nonlinear effects, simulate the outcomes to understand probabilities, and then adjust using a gut-check from community insights or matchup specifics – finally comparing against the sportsbook's line to identify a bet. The methods discussed are not mutually exclusive; they are pieces of a larger puzzle of predictive analytics in sports. The key is that these methods are **actionable** for independent bettors: with public data (e.g. via nflverse [21]), accessible algorithms, and ever-improving tools, individuals can replicate many of the techniques once reserved for sportsbooks or pros. By understanding how each method works and what data it needs, a bettor can choose the level of sophistication that suits them – whether that's a simple Excel model based on averages or a full-fledged machine learning pipeline. Armed with these models and tools, bettors are better equipped to find **inefficiencies in prop markets** and make informed wagers based on data-driven lines rather than hunches. And as the NFL and sports analytics evolve, so too will the models – but the combination of statistical reasoning, computational power, and domain knowledge will remain the core of beating the props.

Sources: Publicly available data, models, and discussions were referenced in compiling these insights, including FiveThirtyEight's NFL Elo methodology [1] [2], an NFL prop modeling case study by Quant Galore [14] [34], the Unabated props simulator guide [23], a Stackademic article on Poisson for NFL props [12], BetQL's description of their prop model [26], Rithmm's documentation of their AI-driven predictions [35] [32], and the nflverse project for NFL data access [28] [22], among others. These illustrate the breadth of approaches and tools available to bettors and analysts today.

---

[1] [2] [5] How Our NFL Predictions Work | FiveThirtyEight
https://fivethirtyeight.com/methodology/how-our-nfl-predictions-work/

[3] [4] GitHub - fivethirtyeight/nfl-elo-game: Data and code for FiveThirtyEight's NFL game
https://github.com/fivethirtyeight/nfl-elo-game

[6] [7] [9] GitHub - quantgalore/nfl-props: NFL Reception Player Props - The Quant's Playbook @ Substack | Quant Galore
https://github.com/quantgalore/nfl-props

[8] [10] [11] GitHub - peanutshawny/nfl-sports-betting: Model created to predict outcomes of nfl games with the goal of making bets to win money!
https://github.com/peanutshawny/nfl-sports-betting

[12] Prop Bets in the NFL and the Poisson Distribution with Python | by Michael Wayne Smith | Stackademic
https://blog.stackademic.com/prop-bets-in-nfl-and-the-poisson-distribution-with-python-a715516b741d?gi=722a8a1f06c4

[13] Predicting Football Match Results Using a Poisson Regression Model
https://www.mdpi.com/2076-3417/14/16/7230

[14] [15] [16] [17] [18] [19] [20] [34] Sport Markets Aren't So Efficient After All... [Code Included]
https://www.quant-galore.com/p/sport-markets-arent-so-efficient

21 22 25 28 nflverse · GitHub
https://github.com/nflverse

23 24 33 How to Create NFL Player Projections For Player Props
https://unabated.com/articles/creating-player-projections-for-nfl-player-props

26 27 Player Prop Model Ratings – BetQL
https://support.betql.co/hc/en-us/articles/15337997591187-Player-Prop-Model-Ratings

29 sports-betting · PyPI
https://pypi.org/project/sports-betting/

30 How AI And Predictive Analytics Power Sports Betting Tool 'Rithmm'
https://www.gamingtoday.com/news/how-artificial-intelligence-predictive-analytics-power-rithmm-ai-sports-betting-tool/

31 32 35 What is an Over/Under in Football?
https://www.rithmm.com/post/what-is-an-over-under-in-football