Advanced Models for Football Match Outcome Prediction

Bayesian Hierarchical Models

Bayesian hierarchical models use **multi-level structures** to capture team and player effects with uncertainty. Instead of fixed coefficients for each team, they treat parameters (like a team's attacking/defensive strength or home advantage) as random variables drawn from a league-wide distribution. This **partial pooling** improves estimates for teams with less data and naturally provides probability distributions for outcomes. Bayesian models can also incorporate **expert knowledge or historical data as priors**, such as using previous season performance or FIFA rankings to inform current team strengthfrontiersin.org. These models are applicable to predicting win/draw/loss or scores by estimating the scoring rates of each team.

- **Application to football:** Hierarchical Poisson models are common, where each team has attack and defense parameters that vary around a league average. For example, Baio and Blangiardo (2010) used a Bayesian hierarchical model to predict Italian Serie A match scores, assigning each team its own offense/defense ability while accounting for home advantagediscovery.ucl.ac.uk. Such models were shown to improve predictive fit but needed tweaks to avoid **over-shrinkage** of team strengths, leading to mixture models for better accuracydiscovery.ucl.ac.uk. Recent research on Brazil's Série A compared **hierarchical vs non-hierarchical Poisson regressions**, confirming that a two-level (team-level) random-effects model can enhance predictionsfrontiersin.org.
- **Notable use cases:** Bayesian approaches have been used in tournament forecasts – e.g. **Suzuki et al. (2006)** incorporated FIFA rankings and expert opinions as priors in a Bayesian model for World Cup matches, achieving good prediction ratesfrontiersin.org. Other studies applied dynamic Bayesian updating of team abilities over time (using state-space models or Kalman filters) to capture form changesfrontiersin.org. Overall, hierarchical Bayesian models offer a flexible framework to include various factors (team strength, venue, player injuries) with uncertainty, often yielding well-calibrated probabilities for match outcomes.

Survival Analysis (Time-to-Event Models)

Survival analysis focuses on **time until an event occurs** – in football, this typically means modeling the timing of goals (or other events like red cards). Instead of directly predicting final scores, a survival model treats goals as events in a timeline, using hazard functions or survival curves to estimate the probability of a goal as time progresses. By modeling the **distribution of goal times**, one can derive the likelihood of different scorelines or which team scores first. This approach captures the dynamic nature of a match – e.g. the longer a match stays 0–0, the more likely a late goal might occur from pressing, or how a goal hazard changes after a red card.

- **Application to football:** One can use a **Cox proportional hazards model** or parametric survival model to examine scoring. For example, Glasson and Bedford (2009) applied survival analysis to **time of first goal**, finding how the hazard of a goal relates to teams' relative strength (rank) and other factorsjournals.sagepub.com. Similarly, researchers have studied the effect of scoring the first goal on the timing of the second using Cox models (treating the first goal as a time-dependent variable).
- **Advanced survival models:** A recent approach by Boshnakov et al. used a **Weibull inter-arrival time model** for goalsfrontiersin.org. In their framework, each team's goal occurrences follow a Weibull distribution (a flexible survival distribution) for the time gaps between goals. They then coupled the two teams' processes with a **copula** to form a joint distribution of home and away goalsfrontiersin.org. This effectively produces a full match score prediction model: the Weibull hazard captures how goal likelihood changes over time, and the copula introduces a correlation between teams (e.g. accounting for the fact that a very defensive match yields low goals for both sides). By treating goal-scoring as a time-to-event problem, these models can simulate or estimate probabilities for win/draw/loss or exact scores given how long it typically takes teams to score. Survival analysis thus adds a temporal dimension to match outcome modeling, beyond what static regression models provide.

Poisson, Negative Binomial, and Zero-Inflated Count Models

Predicting exact scores or total counts (goals, corners, cards, etc.) often involves **count regression models**. The classic choice is the Poisson model, which assumes events occur at a constant rate and the probability of scoring k goals follows a Poisson distribution. In football analytics, it's common to use a **"double Poisson"** setup – one Poisson model for home goals and one for away goals – possibly with different means to reflect team strengths and home advantagemdpi.com. However, real match data can violate Poisson assumptions (equal mean and variance, independent team scores). Advanced count models address these issues:

- **Negative Binomial Regression:** If the variance of goals or other counts is higher than the mean (a frequent case in sports due to occasional blowout scores or high variability in cards/fouls), a negative binomial model can be used. It adds an extra parameter to model **over-dispersion**. In fact, early studies found that while Poisson is a good first model for goals, a Negative Binomial may fit better for datasets where some teams or matches produce unpredictably high scoresmdpi.com. Bäcklund and Nils (2021), for instance, compared negative binomial vs Poisson for football scores and found the former could better accommodate the variance in goal countsfrontiersin.org.

- **Zero-Inflated Models:** Football data often have **excess zeros**, particularly in goals (e.g. 0–0 draws, or a team failing to score). A standard Poisson might underestimate the frequency of zero-goal outcomesmdpi.com. *Zero-inflated Poisson (ZIP)* or *zero-inflated negative binomial* models address this by blending two processes: one that generates structural zeros (e.g. with some probability a team simply doesn't score at all), and another that generates counts according to a Poisson/NB for cases when a team *does* scorefrontiersin.org. This effectively boosts the probability of zero goals beyond the Poisson expectation. In a recent Premier League analysis, a simple Poisson model severely under-predicted 0–0 matches (assigning only 0.9% chance to a Man City vs Liverpool match ending 0–0, when goalless draws actually occurred in 23 of 380 matches)mdpi.com. The authors suggest adding a zero-inflation term to correct this biasmdpi.com. By estimating a zero-inflation parameter, the model explicitly accounts for the likelihood of a goal drought scenariomdpi.com.
- **Use in football:** Poisson regression has been a cornerstone of football score prediction since Maher's seminal 1982 paper. It forms the basis of many betting models and was later extended by Dixon & Coles (1997) with a slight covariance adjustment for low-scoring games. Negative binomial models have been tested as alternatives when data show greater variabilitymdpi.com. Zero-inflated models are newer in this domain, but studies have begun to apply them. For example, a team-specific zero-inflated Poisson was proposed in 2022 to improve goal predictions between certain teamsarxiv.org. Overall, these count models are well-suited for predicting **exact scores** (via two coupled distributions for home/away goals) or **totals** (goals, corners, cards, etc.), with more advanced versions capturing the idiosyncrasies of football data (lots of 0–0, occasional high counts, etc.)frontiersin.org.

Generalized Additive Models (GAMs)

Generalized Additive Models extend traditional linear regression by allowing **non-linear effects** of predictors while retaining interpretability. A GAM predicts an outcome as a sum of smooth functions of the inputs: $g(E[Y]) = \beta_0 + f_1(X_1) + f_2(X_2) + \dots$, where f_i are spline or loess functions fitted from data. This flexibility is useful for football data, where relationships may not be purely linear (e.g., the effect of possession percentage on win probability might increase steeply up to a point then level off). GAMs can handle both classification (via logistic link for win/draw/loss probability) and regression (via identity/log link for counts) by choosing an appropriate link function in the GLM family.

- **Application to football:** GAMs have been successfully used to model **expected goals (xG)** and other performance metrics in an interpretable way. For instance, Van Haaren et al. (2021) introduced an **explainable GAM for predicting the value of shots**, i.e. the chance a shot will result in a goalarxiv.org. By using smooth functions of shot features (like distance, angle, assist type), the GAM could match the accuracy of complex black-box models while being **easier to interpret for coaches**arxiv.org. The model broke down the contribution of each feature (e.g. shot distance effect curve) so practitioners could trust the numbersarxiv.org.

- **Why GAMs are useful:** They capture **non-linear patterns** (which are common in sports data) without manual feature engineering. For example, a GAM can learn that very low or very high possession values both lead to lower win probability (an inverted U-shape), or that the effect of a player's age on performance is non-linear. Indeed, studies on player performance trends have used GAMs to reveal subtle non-linear age curves and match intensity effectsnature.com. In football outcome prediction, one could use a GAM to allow, say, team Elo rating to have a diminishing returns effect, or different smooth terms for a team's form over the last N games. Decroos and Davis (2019) applied a GAM to predict the probability of a goal being scored in a sequence of play, as a more interpretable alternative to tree-based models.
- **Notable example:** Decroos & Davis replaced a complicated ensemble (used in a **VAEP** player value model) with a **simpler GAM using 10 features** to predict goal events, making the model's reasoning transparentsemanticscholar.org. This shows how GAMs can serve in **sports analytics** to balance accuracy and interpretability. Overall, GAMs are well-suited for football data, which often has nonlinear relationships and domain-specific thresholds (like fatigue effects after 70 minutes, or exponential drop in win probability as goal deficit increases). By visualizing each feature's effect, analysts can **explain the model's predictions**, an important factor for adoption by coaches and clubs.

Bivariate Outcome Models (Joint Score Distributions)

While one can model goals for each team separately, treating them jointly can improve exact score forecasts. **Bivariate distribution models** capture the correlation between two teams' scores. Two main approaches have emerged: (1) direct multivariate distributions (like bivariate Poisson or bivariate Negative Binomial), and (2) modeling the **goal difference** and total (or one team's goals and the goal difference). These methods acknowledge that the scores are not independent – for instance, very high scoring games typically involve both teams scoring more (or one scoring a lot and the other less), and low scoring games often mean low for both (a defensive battle).

- **Bivariate Poisson models:** A bivariate Poisson allows the two goal counts to share a component. In the Karlis & Ntzoufras model, each match has three Poisson components: one for home goals, one for away goals, and one shared component that adds to both (creating correlation). Their 2003 paper showed that a **bivariate Poisson fit** could better match observed score distributions in soccermdpi.com. Notably, they found it useful to include a **common home advantage parameter** and that one could often treat attack and defense as constant across home/away for simplicitymdpi.commdpi.com. Bivariate Poisson models have been widely cited in the literature as an improvement over independent Poissons for exact score predictions.

- **Skellam (goal-difference) models:** The Skellam distribution (difference of two independent Poissons) gives the distribution of goal difference. Modeling goal difference directly is another route: fit a Skellam for the score margin and maybe a separate model for total goals or one team's goals. Karlis & Ntzoufras also proposed a **Bayesian Skellam model** that included **zero-inflation for draws** and a correlation between teams' attack/defense strengthsfrontiersin.org. In effect, they inflated the probability of zero goal difference (draws) beyond the Skellam baseline, acknowledging that draws can be more frequent than naive models expectfrontiersin.org. This approach focuses on getting win/draw/loss probabilities right, since those relate directly to goal difference.
- **Copula and other multivariate methods:** Instead of a specific parametric form, one can model each team's goals with a marginal distribution (Poisson or other) and then join them with a **copula** to induce a correlation. Boshnakov et al. (2017) did this by using a **Weibull count process** for each team's goals (as discussed earlier) and coupling them with a copula to produce a full joint distribution for (home_goals, away_goals)frontiersin.org. The copula allows flexible correlation structures (for example, ensuring that if one team scores a lot, the other's likelihood of also scoring might increase in open-play scenarios or decrease if one dominates completely, depending on the chosen copula). Copula models are less common but offer flexibility to fit empirical joint distributions of scores.
- **Use cases:** Bivariate and goal-difference models are particularly useful for **exact score forecasting**. For example, a betting analyst might use a Dixon-Coles model (a slight tweak on independent Poisson with a covariance adjustment) to predict the probabilities of each possible scoreline for setting odds. Academic studies have found that incorporating a correlation term (like Dixon-Coles' adjustment for low scores or a full bivariate model) improves predictive log-likelihood of match scoresfrontiersin.org. Additionally, these models can be extended: **dynamic bivariate models** allow team strengths to evolve over time (Koopman & Lit (2015) used a state-space bivariate Poisson to update team attack/defense each weekfrontiersin.org), and **multivariate extensions** can include extra outcomes (like modeling goals and red cards together in a multivariate framework to see how a red card influences goals). In summary, jointly

modeling scores yields more accurate and realistic predictions for exact scores, since it captures the dependency between teams' performances in a match.

Elo Ratings and Bradley-Terry Models

Not all effective football prediction models are based on traditional regression or ML – **rating systems** like Elo and Bradley-Terry are powerful for win/draw/loss classification. These approaches convert past match results into team strength ratings, which in turn predict future outcomes.

- **Elo Ratings:** Originally developed for chess, Elo ratings have each team start with a rating (e.g., 1500) and then **update after every match** based on the result and the expected result. Beating a strong opponent yields a big rating gain, while losing to a weaker team causes a big loss; draws cause small adjustments depending on expectations. Elo ratings are **dynamic** and self-correcting – they reflect current form and strength. To predict a match, the difference in Elo ratings between two teams is mapped to an expected win probability (usually via a logistic curve). Elo models are **simple but effective**: they've been shown to rival more complex models in accuracy. For example, during World Cups, Elo-based predictions often outperform FIFA's official rankings in predicting match outcomesdubstat.com. Elo ratings provide a natural probability estimate for each outcome; by comparing two teams' ratings, one can calculate the chance of win/draw/loss (draws are handled by assuming a certain draw probability or by using an Elo variant that includes draws). Sports analysts and bettors favor Elo because of its **predictive accuracy and real-time adaptability**dubstat.com. It's common to see websites like ClubElo or FiveThirtyEight use Elo systems for league and international match forecasts.
- **Bradley-Terry and extensions:** The Bradley-Terry model is a classic statistical model for pairwise comparisons (matches). It assigns each team a strength parameter p_i; the probability team A beats team B is $P(A\text{ wins}) = \frac{p_A}{p_A + p_B}$. This can be reparameterized in logistic form $\logit(P_A) = \beta_A - \beta_B$. Fitting a Bradley-Terry model to past results yields a rating for each team (similar to Elo but derived from a maximum likelihood estimation rather than sequential updating). **Extensions** like the Davidson model incorporate draws by adding a draw parameter. In football analytics, Bradley-Terry models have been used to rate teams or even players (e.g., who wins more duels). They are essentially equivalent to **logistic regression on match outcomes** with team indicators. One advantage is that they naturally account for strength of schedule – beating strong teams raises p_i more significantly. These models are less frequently mentioned by name in soccer literature (often overshadowed by Elo), but the concept underlies many ranking systems.

- **Use in football:** Rating-based predictions are well-suited for structured data and **time-series forecasting of matchups** because they carry information from past matches into future predictions. A system like Elo implicitly captures momentum and form. For instance, if a mid-table team suddenly improves and wins several games, Elo will quickly raise its rating, and thus the model will predict higher chances for that team in upcoming fixtures – effectively a form of time-series adaptation. Ratings can also integrate player stats (some advanced versions adjust team ratings if key players are injured, etc.). Empirically, Elo ratings have demonstrated strong predictive power: one article notes that Elo's mathematically-updated rankings yield more accurate forecasts of match results than subjective or static systemsdubstat.comdubstat.com. Many bookmakers use Elo-like systems as part of their models. Bradley-Terry models, being equivalent to a logistic regression, can incorporate additional features (home advantage can be added as a constant term β_{home}; one can extend it to account for team form by making p_i time-dependent, etc.). These rating approaches are not in the user's PDF but are **commonly used in sports analytics** for classification tasks (will team A win, draw, or lose). They are particularly appealing when one has **structured historical data** (season-by-season results) and wants a quick, interpretable prediction model.

Transformer-Based Time-Series Models

Modern deep learning has introduced **transformer architectures** that excel at sequence modeling. Unlike RNNs (e.g. LSTM in the PDF) which process time steps sequentially, transformers use **self-attention** to learn long-range dependencies in parallel. In time-series forecasting, transformers can capture complex patterns, seasonality, and interactions across multiple series. For football, this could mean modeling sequences of past match features or player statistics over time to predict future outcomes or metrics. For example, one could feed a transformer the timeline of team performances (with features like goals, shots, possession in recent matches, lineup changes, etc.) to forecast the probability of winning the next game or to predict next match's goal counts.

- **Why transformers:** Transformers have shown **state-of-the-art performance** on many forecasting benchmarks, often outperforming traditional models and RNN-based modelsnature.com. They handle long sequences well and can incorporate covariates (e.g., known schedule, opponent features, weather) through appropriate architecture design. In a sports context, a transformer could learn patterns like "Team A tends to struggle in away games after a Champions League midweek match" or capture interactions like "if key player X is absent, and the team faces a high-pressing opponent, their goal rate drops significantly" – patterns that might be hard to encode manually.

- **Application potential:** While not yet prevalent in published football literature, these models are beginning to appear. Researchers are exploring transformers for sports time-series, such as injury prediction or trajectory forecasting. In match outcome prediction, one could use a **Temporal Fusion Transformer (TFT)** or similar model that takes as input: team form (as a sequence of past match stats), opponent strength series, and static info (team budget, coach, etc.), and outputs win/draw/loss probabilities or expected goals for the next match. The **attention mechanism** would allow the model to weigh the importance of specific past games (e.g., giving more weight to recent matches or to games against similar-caliber opponents). Because transformers are data-hungry, this approach works best with rich data (multiple seasons, player-level stats, etc.). Some contemporary studies in other domains (economics, meteorology) show transformers achieving **top accuracy in multivariate forecasting**nature.com, which suggests that as football clubs accumulate more data (tracking data, detailed event data over years), transformer models could become very powerful for predictions like total shots, possession, or even *in-game win probability* that evolves over time.
- **Example and performance:** A 2025 study proposed *MultiPatchFormer*, a transformer variant for multivariate time series, and it outperformed many baseline models on economic and energy datasetsnature.com. This indicates the model's ability to capture multi-scale temporal patterns and cross-series relationships – analogous to capturing multi-season cycles or team interactions in sports. While specific football-focused transformer models are still emerging, the technique is promising. **In summary**, transformer-based models represent the cutting edge for structured time-series prediction, offering the ability to model complex, non-linear temporal dynamics in football data. As data availability grows, we expect to see them perform strongly in problems like **match outcome forecasting, player performance over a season, or injury risk prediction**, complementing the more classical statistical models above.

Sources: The above approaches have been synthesized from sports analytics literature and recent machine learning research. Key references include traditional studies on Poisson and Bayesian modelsmdpi.comfrontiersin.orgfrontiersin.org, modern developments like zero-inflated and dynamic modelsmdpi.comfrontiersin.org, explainable models for expected goalsarxiv.org, and general insights from time-series forecasting advancesnature.com. These models, though not covered in the user's PDF of 17 basic ML models, are highly relevant for football (soccer) outcome prediction and have demonstrated their value in both research and practice. Each can be selected or combined depending on the task – classification (win/draw/loss), exact score prediction, or regression for match statistics – to improve predictive performance while accounting for the unique distributional characteristics of football data.

Sources

You said:

other statistical models or predictions models that are highly relevant for football (soccer) outcome prediction and have demonstrated their value in both research and practice

ChatGPT said:

To help you further, could you confirm if you're looking for:

1. **Only statistical models** (like those from econometrics or applied statistics), or are **machine learning models** also okay if they are used in practice for football?
2. Should the focus be only on **match outcome prediction** (win/draw/loss), or also include **in-play events** (e.g., real-time predictions, expected goals, player performance, etc.)?
3. Are you interested in **real-world implementations** (e.g., models used by betting companies or clubs) or just **peer-reviewed research models**?

This will help me prioritize the most applicable and evidence-backed models.

You said:

1. Only statistical models and predictions models only 2. both but specify with model is good for in play eents and which one is good for outcome match predictions 3. real world implementation in a pyton app

ChatGPT said:

Thanks for clarifying. I'll compile a list of statistical and predictive models—excluding pure ML models—that are highly relevant for football match outcome prediction or in-play event forecasting, and that have been implemented in real-world Python applications.

I'll indicate for each model whether it's suited for match outcome prediction (e.g., win/draw/loss, scoreline) or for in-play events (e.g., expected goals, cards, shots), and include examples of real-world usage where available. I'll let you know when the research is ready!

Research completed in 17m · 13 sources · 138 searches

Match Outcome Prediction Models

Poisson Goal Models (and Variants)

Use case: Best for predicting full-time scores and match outcomes (win/draw/loss probabilities).
How it works: Each team's goals are modeled as a Poisson-distributed random variable. In the basic model, home and away goals are assumed independent, with each team having an **expected goals** value based on its attack strength, opponent's defense, and home-field advantage. For example, if Team A averages 1.5 goals and Team B 1.0 goals, a Poisson model can estimate the probability of any scoreline (e.g. 2–1, 0–0, etc.). The **Skellam distribution** (difference of two Poissons) gives a direct handle on win/draw/loss probabilities. This approach is effective because real soccer scores roughly follow a Poisson pattern, making it straightforward to calculate outcome oddspinnacle.bet. However, the basic independent Poisson tends to **under-predict draws** (especially 0–0 or 1–1 outcomes) since it doesn't account for correlation in scoring.
Extensions and why effective: *Dixon-Coles model* is a famous improvement that adds a correlation adjustment (parameter ρ) to boost the probability of low-scoring drawspypi.org. In practice this means the model explicitly increases the likelihood of outcomes like 0–0, 1–0 or 0–1, aligning better with empirical datapypi.org. Dixon-Coles also proposed down-weighting older matches so that recent form counts more. Other variants include *bivariate Poisson* models, which introduce a shared factor so that one team's high scoring can slightly reduce the other's scoring rate (capturing defensive interplay), and using a Skellam distribution directly for goal **difference** modeling (useful for handicap betting). These statistical tweaks improve predictive accuracy by fitting soccer's tendency for extra draws and the defensive-offensive balance between teams.
Real-world use and Python implementation: Bookmakers and analysts commonly use Poisson-based models for odds-making – for instance, Pinnacle's betting guides demonstrate Poisson to forecast soccer scorespinnacle.bet. Data scientists often fit Poisson regression (a GLM) to estimate each team's attack/defense parameters. In Python, one can implement this with statsmodels or optimize a likelihood function manually (as shown in many blog tutorials). There are also ready libraries – e.g. the **penaltyblog** package – which offers high-performance Poisson, Dixon-Coles, and bivariate Poisson models for match outcome predictiongithub.com. These models have been validated in academic research and are used in practice for setting odds and powering prediction apps.

Rating Systems (Elo and Elo-based Models)

Use case: Primarily for predicting match results (win/draw/loss) and updating team strength over time.
How it works: Elo ratings assign each team a numerical strength rating. After each game, ratings are updated based on the result (teams gain points for wins, lose points for losses, with the exchange amount depending on the expected result). The **rating difference** between two teams can be converted into an expected win probability via a logistic formula. For example, a 100-point Elo advantage corresponds to roughly a 64% win chance for the higher-rated sideen.wikipedia.org. Draws are handled by smaller rating exchanges or an explicit draw probability formula. Elo's strength is its iterative nature – a team's rating evolves as results come in, reflecting form and quality changes. It has a solid statistical basis and is **widely used in soccer** – FIFA's women's world ranking and many public models use Elo or variantsen.wikipedia.org.
Why it's effective: Elo is a **proven predictor** because it encapsulates a lot of historical performance information in one number per team. It's especially good for head-to-head outcome probabilities and inherently accounts for regression to the mean (a string of upset wins boosts a team's rating, raising expectations going forward, and vice versa). Unlike Poisson models, Elo doesn't directly yield scorelines, but it focuses on win/lose likelihood. Extensions like **Glicko** (which adds rating uncertainty) or **TrueSkill** (a Bayesian Elo for games with draws) similarly see use in sports analytics to maintain up-to-date team strength estimates.
Real-world use and Python implementation: Many betting and prediction systems incorporate Elo-type ratings. FiveThirtyEight's well-known club soccer forecasts, for example, include an Elo component in their Soccer Power Indexfivethirtyeight.comfivethirtyeight.com. Clubs and analysts might use Elo to track team performance or as features in larger models. Implementing Elo in Python is straightforward – it's essentially a few lines of code to update ratings after each match. Libraries like elo-python or the penaltyblog toolkit provide convenience functions as well (the latter even includes Elo and other rating systems built-ingithub.com). Overall, Elo provides a simple yet effective baseline model for match outcome prediction and is often combined with other approaches for more nuanced forecasting.

Bayesian Hierarchical Models

Use case: Best suited for match outcome and score predictions, especially when we want to estimate team abilities with limited data or incorporate uncertainty.
How it works: These models extend the Poisson goal framework by treating team **attack and defense strengths as random effects** (latent variables) rather than fixed values. In a typical hierarchical model, each team *i* has an attack strength α_i and defense strength β_i drawn from a population distribution (often centered around league-average). Goals scored by Team A against Team B are then modeled as Poisson($\exp(\alpha\_A – \beta\_B + H)$) for home team (H is home

advantage) and Poisson(exp(α_B – β_A)) for awaypypi.orgpypi.org. The Bayesian approach puts priors on these parameters and uses the data (matches played) to **update the distributions (posterior)** of team strengths. This yields a **partial pooling** effect: extreme values get "shrunk" toward the average unless strongly supported by data.
Why it's effective: Hierarchical Bayesian models can **simultaneously estimate team qualities and predict match outcomes**statmodeling.stat.columbia.edu. They borrow strength across teams and seasons – for example, a newly promoted team's attack rating will be informed by the overall prior (league average) until enough matches are observed. This prevents overfitting on small sample sizes and naturally quantifies uncertainty in predictions with credible intervals. Such models were shown to improve predictive accuracy in research. Baio and Blangiardo (2010) famously built a Bayesian model for Italian Serie A that could estimate the factors leading to wins/losses and predict match scoresstatmodeling.stat.columbia.edu. Similarly, R. Bååth (2015) demonstrated a Bayesian Poisson model on Spanish league data, producing not only match predictions but also **probabilistic team rankings** (with uncertainty bounds for each team's strength)lucs.lu.se.
Real-world use and Python implementation: While computationally heavier, these models are used in practice by advanced analysts and academics – for instance, to power club decision-making or betting syndicate models that continually update team ratings. In Python, one can implement hierarchical soccer models using libraries like **PyMC3/PyMC** or **Stan (via CmdStanPy/Pystan)**. There are examples in the community (e.g. blog posts and notebooks porting the Baio & Blangiardo model to PyMC3) that show how to use Markov Chain Monte Carlo to fit such modelsmedium.com. The open-source penaltyblog package also supports Bayesian formulations and even MCMC fitting for some modelspypi.org. These Bayesian models, validated in peer-reviewed studies, strike a good balance between **interpretability** (attack/defense ratings with uncertainty) and predictive performance, making them highly relevant for serious football outcome forecasting.

In-Play Event Forecasting Models

Expected Goals (xG) Models

Use case: Used for **in-play event prediction**, especially for evaluating chances of scoring from shots and forecasting total goals. (While xG is often a post-game analytic, it's fundamentally a predictive model for each shot's outcome, and summing xG gives an expectation of goals.)
How it works: An xG model typically uses a logistic regression (or another binary outcome model) to estimate the probability that a given shot results in a goal. Features like shot distance, angle to goal, type of assist, body part, etc., are inputs. The model outputs a probability between 0 and 1 (this is the "expected goal" value of the shot). For example, a shot from the center of the six-yard box might have xG ≈ 0.3 (30% chance of goal), whereas a long-range

attempt might be xG ≈ 0.02. These probabilities are learned from historical data of thousands of shots. **Why effective:** xG models quantify chance quality far better than raw shot counts – a team that had 10 shots worth 0.1 xG each (total 1.0 expected goals) likely posed more threat than a team with 10 speculative 0.01 xG shots (0.1 expected goals). Over many games, cumulative xG correlates strongly with goals scored, so it's predictive of future performance and scoring. In live forecasting, summing the xG of chances a team creates can update our expectation of how many goals they "should" score.

Real-world use: Expected goals have become **standard in clubs and broadcasting** to assess performance. Many betting models incorporate xG to adjust live odds (if a team is creating high-xG chances, their win probability in-play should increase even if they haven't scored yet). FiveThirtyEight's soccer SPI model, for instance, uses shot-based expected goals as a key componentfivethirtyeight.com. On the implementation side, building an xG model in Python can be done with libraries like scikit-learn or statsmodels (logistic regression). There are public datasets and notebooks demonstrating this. Additionally, some Python libraries (e.g., sklearn pipelines in **Metrica Sports** or open-source codes on GitHub) allow you to plug in shot data and train an xG model. In practice, companies like Opta/StatsPerform provide pre-trained xG models as part of their analytics offerings, and open-source projects exist for learning xG from event data. Overall, xG is a **practical, validated tool** for in-play goal forecasting and is often the foundation for more complex live win probability models.

Time-to-Event (Survival) Models for In-Play Forecasting

Use case: These approaches are used for **live match prediction** – e.g. forecasting the timing of the next goal, the evolving win/draw/lose probabilities as a match progresses, or the likelihood of events like a red card in the remainder of a game.

How it works: Survival analysis and point process models treat goals (or other events) as events in continuous time. A simple version is to assume goal arrivals follow a Poisson process with some rate (goals per minute). Under a constant rate assumption, the time until the next goal is exponentially distributed. More sophisticated models make the rate **state-dependent** – i.e. the scoring intensity λ_home and λ_away can change depending on the current score, time remaining, or other factors. For example, if a team is trailing late, their scoring rate might increase while the leading team's rate drops (parking the bus). Dixon & Robinson (1998) introduced a two-team **"birth process"** model: essentially each team's goals arrive as a Poisson process, and after each goal the intensities are adjusted for the new scoreline. In survival terms, one can also use a **Cox proportional hazards model** to include covariates (like which team scored first, or if a red card has occurred) when modeling time to next goal. These models naturally account for *censoring* (e.g. "no goal happened before full-time" is a censored outcome) and allow updating probabilities as time ticks away.

Why it's effective: Time-to-event models directly tackle the question "Given the match state

now, what is the probability X will happen in the next interval (or by game end)?” This is crucial for in-play betting and coaching decisions. For instance, a survival model can tell us the probability that the trailing team will equalize before the match ends, based on how much time is left and historical goal rates in similar situations. Unlike static pre-match models, survival models are **dynamic** – as the clock runs and no goal occurs, the chances of a comeback naturally decline (the hazard of goal is integrated over less time). These methods have been validated in research (e.g. using survival analysis to show how the first goal’s timing affects the second goal) and align with how odds swing during matches.
Real-world use: Modern win probability calculators often use a form of this. A practical implementation is to **estimate the instantaneous scoring probabilities** for each team and then simulate the rest of the match many times. For example, *American Soccer Analysis* described a model that at any game minute takes inputs like time remaining, score, team strengths, etc., and produces the probability each team scores in each of the next minutes – then runs Monte Carlo simulations to get win/draw/lose oddsamericansocceranalysis.com. In that approach, if Team A has (say) a 0.7% chance of scoring in the next minute and Team B a 3.8% chance (perhaps Team B is pushing hard), by simulating minute-by-minute you can compute that Team B might have, for example, a ~49% chance to overturn a 1–2 deficit in the remaining 40 minutesamericansocceranalysis.comamericansocceranalysis.com. Betting markets also implicitly use such models (the live odds of a draw or comeback are essentially derived from estimated scoring intensities given the state).
Python implementation: One can implement a basic hazard model by fitting an exponential or Weibull distribution to goal times, or use lifelines library for Cox models to include covariates. For the simulation approach, you can use a loop or vectorized simulation (or even a custom Markov chain) to roll out many random scenarios of goals given estimated per-minute scoring probabilities. While specialized libraries for sports survival models are rare, the logic can be coded with standard Python scientific stacks (numpy random draws for Poisson events, etc.). Some open-source notebooks (e.g., by soccer analytics enthusiasts) demonstrate live win-probability simulation using expected goals and time decay as inputs. These time-to-event models, whether analytical or simulation-based, are grounded in statistics (Poisson processes and survival analysis) and have been **validated by both peer-reviewed studies and real betting outcomes**, making them a cornerstone of in-play forecasting.

Other Statistical Approaches

Beyond the major categories above, a few other noteworthy modeling approaches are used in football analytics:

- **Ordered Outcome Models:** Instead of predicting exact scores, one can model the match result (win/draw/loss) as an **ordinal outcome**. Techniques like ordered logistic regression or probit models treat a draw as the “middle” outcome between a home win and away win. These can incorporate team strengths similarly to Elo or Poisson models. They have a solid statistical basis, though in practice they often perform similarly to ratings or Poisson-based methods when features are similar.

- **Rating Systems Extensions:** The basic Elo can be extended with factors for goals difference (to reward big wins) or adjusted for tournament importance. **TrueSkill**, a Bayesian rating system, has been used in some research to rate teams or even players, accounting for draw probabilities explicitly. These are statistically grounded (assuming underlying normal skill distributions) and have seen practical use in ranking teams and simulating tournament outcomes.
- **Markov Chain Models for Game Flow:** Some research models the progression of a match as a Markov chain with states (possession, attacking move, etc.) culminating in shots or goals. For example, one can define states for “team A in possession” vs “team B in possession” and absorbing states for goals. Transition probabilities (estimated from data) then yield insights into scoring likelihood. While more common in evaluating **tactical sequences** than directly predicting final scores, this approach is statistical (based on transition matrices) and has been explored in football analytics literature. It’s particularly useful for computing metrics like **expected possession value (EPV)** or **expected threat (xT)**, which forecast the chance of a goal given the ball’s current state. Python implementations often use custom code with numpy for matrix exponentiation or simulation to realize these Markov models.
- **Hawkes Processes (self-exciting events):** An emerging statistical approach is to model event sequences (like shots, passes, or fouls) with Hawkes point processes, where one event can temporarily raise the probability of follow-up events. In soccer, a flurry of shots might be modeled this way (a rebound shot is more likely right after an initial shot on goal, for instance). This is a research frontier rather than a widespread practice, but it is grounded in statistical point process theory. A few academic papers have applied Hawkes processes to soccer for things like modeling the momentum of attacks. While not yet common in production models, open-source libraries (like hawkesspatial in R, or custom Python code using tick library) have been used in experiments to forecast sequences of events in matches.

Each of these approaches – from ordered models to Markov and point processes – adds a twist to how we statistically view a football match. They are **unified by statistical principles** (probability distributions, Markov property, etc.) and several have been validated in studies or pilot implementations. For instance, a **hierarchical model for yellow cards** has been explored with Bayesian methods (treating referees as a factor)dm13450.github.io, and zero-inflated

Poisson models have been used to handle the excess of matches with 0 cards or 0 goalsresearchgate.net. While general machine learning (like SVMs or neural networks) can also tackle these problems, the approaches listed here have the advantage of interpretability and often require less data, making them highly relevant for both academic research and practical deployment in football outcome prediction and in-play forecasting.

Sources: The above guide is informed by a wide range of soccer analytics research and applications, including Dixon & Coles's seminal statistical model for scorespypi.org, Bayesian team-strength models from academic literaturestatmodeling.stat.columbia.edu, bookmaker methodologies for Poisson predictions, Elo rating adaptations to socceren.wikipedia.orgen.wikipedia.org, and modern implementations like the penaltyblog Python library which provides efficient tools for Poisson, Dixon-Coles, bivariate models and Elo ratingsgithub.comgithub.com. In-play forecasting insights draw from survival analysis studies and industry examples of live win probability simulationsamericansocceranalysis.com, as well as widely used metrics like expected goalsfivethirtyeight.com that underpin many real-time predictive systems. These approaches have all been validated either by peer-reviewed research or by their successful use in betting markets, professional clubs, and popular analytics platforms.

Alright, let's turn that architecture into a *real* Python app you could actually start coding.

I'll give you:

- A **project structure**
- **requirements**
- Concrete **Python files** with code (DB, ingestion from APIfootball, features, models, training, prediction API)

You'll be able to copy–paste into files and have a working skeleton you can extend with more models.

---

1. Project structure

You can use this as your base:

football_predictor/

  requirements.txt

```
.env                    # API keys, DB URL
app/
 __init__.py
 config.py

 db/
  __init__.py
  base.py
  session.py
  models.py

 ingestion/
  __init__.py
  apifootball_client.py
  sync_jobs.py

 features/
  __init__.py
  prematch_builder.py
  inplay_builder.py

 models_core/
  __init__.py
  base.py
  registry.py
  statistical/
```

```
    __init__.py
    elo.py
    poisson_dc.py
  ml/
    __init__.py
    logistic_outcome.py
    random_forest_outcome.py

 services/
  __init__.py
  training_service.py
  prediction_service.py

 api/
  __init__.py
  routes_matches.py
  routes_predictions.py

 schemas/
  __init__.py
  prediction_schemas.py

 main.py

scripts/
 backfill_matches.py
```

train_prematch_outcome.py

We'll fill key files now.

---

2. requirements.txt

```
fastapi
uvicorn[standard]
python-dotenv

SQLAlchemy>=2.0
psycopg2-binary

pandas
numpy
scipy

scikit-learn
xgboost

requests
joblib
```

Install with:

```
pip install -r requirements.txt
```

---

3. Config & DB setup

3.1. .env (example)

```
APIFOOTBALL_API_KEY=YOUR_API_KEY_HERE
```

```
DATABASE_URL=postgresql+psycopg2://user:password@localhost:5432/football
DEFAULT_LEAGUE_ID=152
```

3.2. app/config.py

```python
import os
from dotenv import load_dotenv

load_dotenv()

APIFOOTBALL_API_KEY = os.getenv("APIFOOTBALL_API_KEY", "")
DATABASE_URL = os.getenv("DATABASE_URL")
DEFAULT_LEAGUE_ID = int(os.getenv("DEFAULT_LEAGUE_ID", "152"))  # example EPL
MODEL_STORE_DIR = os.getenv("MODEL_STORE_DIR", "model_store")
```

3.3. app/db/base.py

```python
from sqlalchemy.orm import declarative_base

Base = declarative_base()
```

3.4. app/db/session.py

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from app.config import DATABASE_URL

engine = create_engine(
    DATABASE_URL,
    pool_pre_ping=True,
)
```

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

3.5. app/db/models.py

```
from sqlalchemy import (
    Column, Integer, String, Date, Time, Float, JSON, ForeignKey
)
from sqlalchemy.orm import relationship
from app.db.base import Base


class Team(Base):
    __tablename__ = "teams"

    id = Column(Integer, primary_key=True)  # API team_id
    name = Column(String, nullable=False)
    country = Column(String)

    home_matches = relationship("Match", foreign_keys="Match.home_team_id",
back_populates="home_team")
    away_matches = relationship("Match", foreign_keys="Match.away_team_id",
back_populates="away_team")



class Match(Base):
    __tablename__ = "matches"

    id = Column(Integer, primary_key=True)  # API match_id
    league_id = Column(Integer, index=True)
    home_team_id = Column(Integer, ForeignKey("teams.id"))
```

```
    away_team_id = Column(Integer, ForeignKey("teams.id"))
    date = Column(Date, index=True)
    time = Column(Time)
    status = Column(String)  # "Finished", "Not Started", "13'"
    home_ft_goals = Column(Integer)
    away_ft_goals = Column(Integer)

    raw_json = Column(JSON)

    home_team = relationship("Team", foreign_keys=[home_team_id],
back_populates="home_matches")
    away_team = relationship("Team", foreign_keys=[away_team_id],
back_populates="away_matches")
    stats = relationship("MatchStats", back_populates="match", uselist=False)


class MatchStats(Base):
    __tablename__ = "match_stats"

    id = Column(Integer, primary_key=True)
    match_id = Column(Integer, ForeignKey("matches.id"), index=True)
    period = Column(String)  # e.g. "full"

    home_shots_total = Column(Integer)
    away_shots_total = Column(Integer)
    home_corners = Column(Integer)
    away_corners = Column(Integer)
```

```python
    home_cards_yellow = Column(Integer)
    away_cards_yellow = Column(Integer)
    home_cards_red = Column(Integer)
    away_cards_red = Column(Integer)
    home_fouls = Column(Integer)
    away_fouls = Column(Integer)

    raw_stats = Column(JSON)

    match = relationship("Match", back_populates="stats")


class RatingHistory(Base):
    __tablename__ = "rating_history"

    id = Column(Integer, primary_key=True)
    team_id = Column(Integer, ForeignKey("teams.id"))
    date = Column(Date, index=True)
    rating_type = Column(String)  # "elo", etc.
    rating = Column(Float)
```

Run migrations quickly by:

```python
# small helper script
from app.db.base import Base
from app.db.session import engine
from app.db import models  # noqa
```

```python
Base.metadata.create_all(bind=engine)
```

---

4. Ingestion from APIfootball

4.1. app/ingestion/apifootball_client.py

```python
import requests
from typing import Any, Dict, List, Optional
from app.config import APIFOOTBALL_API_KEY

BASE_URL = "https://apiv3.apifootball.com/"

class APIFootballClient:
    def __init__(self, api_key: Optional[str] = None):
        self.api_key = api_key or APIFOOTBALL_API_KEY

    def _get(self, params: Dict[str, Any]) -> List[Dict[str, Any]]:
        params["APIkey"] = self.api_key
        resp = requests.get(BASE_URL, params=params, timeout=30)
        resp.raise_for_status()
        data = resp.json()
        if isinstance(data, dict) and data.get("error"):
            raise RuntimeError(f"API error: {data}")
        return data

    def get_events(
        self,
        date_from: str,
```

```
        date_to: str,
        league_id: Optional[int] = None,
        match_live: Optional[int] = None,
    ) -> List[Dict[str, Any]]:
        params = {
            "action": "get_events",
            "from": date_from,
            "to": date_to,
        }
        if league_id is not None:
            params["league_id"] = league_id
        if match_live is not None:
            params["match_live"] = match_live
        return self._get(params)
```

4.2. app/ingestion/sync_jobs.py

```
from datetime import date
from sqlalchemy.orm import Session
from app.ingestion.apifootball_client import APIFootballClient
from app.db.models import Team, Match, MatchStats


STAT_MAP = {
    "Shots Total": ("shots_total", int),
    "Corners": ("corners", int),
    "Yellow Cards": ("cards_yellow", int),
    "Red Cards": ("cards_red", int),
    "Fouls": ("fouls", int),
```

```
}


def get_or_create_team(session: Session, team_id: int, team_name: str, country: str | None):
    team = session.get(Team, team_id)
    if not team:
        team = Team(id=team_id, name=team_name, country=country)
        session.add(team)
        session.flush()
    return team


def sync_day(session: Session, target_date: date, league_id: int | None = None):
    client = APIFootballClient()
    events = client.get_events(
        date_from=target_date.isoformat(),
        date_to=target_date.isoformat(),
        league_id=league_id,
    )

    for ev in events:
        match_id = int(ev["match_id"])
        league_id = int(ev["league_id"])
        home_team_id = int(ev["match_hometeam_id"])
        away_team_id = int(ev["match_awayteam_id"])

        home_team = get_or_create_team(
            session, home_team_id, ev["match_hometeam_name"], ev.get("country_name")
```

```
)
away_team = get_or_create_team(
    session, away_team_id, ev["match_awayteam_name"], ev.get("country_name")
)

match = session.get(Match, match_id) or Match(id=match_id)
match.league_id = league_id
match.home_team_id = home_team.id
match.away_team_id = away_team.id
match.date = date.fromisoformat(ev["match_date"])
match.time = None  # parse ev["match_time"] if you want
match.status = ev["match_status"]
match.home_ft_goals = int(ev.get("match_hometeam_ft_score") or 0)
match.away_ft_goals = int(ev.get("match_awayteam_ft_score") or 0)
match.raw_json = ev

session.add(match)

# stats
stats_full = ev.get("statistics", [])
home_stats = {}
away_stats = {}
for s in stats_full:
    t = s["type"]
    if t not in STAT_MAP:
        continue
```

```
            fname, cast = STAT_MAP[t]
            home_stats[fname] = cast(str(s["home"]).replace("%", "") or 0)
            away_stats[fname] = cast(str(s["away"]).replace("%", "") or 0)

        ms = session.query(MatchStats).filter_by(match_id=match_id,
period="full").one_or_none()
        if not ms:
            ms = MatchStats(match_id=match_id, period="full")
            session.add(ms)

        ms.home_shots_total = home_stats.get("shots_total")
        ms.away_shots_total = away_stats.get("shots_total")
        ms.home_corners = home_stats.get("corners")
        ms.away_corners = away_stats.get("corners")
        ms.home_cards_yellow = home_stats.get("cards_yellow")
        ms.away_cards_yellow = away_stats.get("cards_yellow")
        ms.home_fouls = home_stats.get("fouls")
        ms.away_fouls = away_stats.get("fouls")
        ms.raw_stats = stats_full

    session.commit()
```

4.3. script scripts/backfill_matches.py

```
# scripts/backfill_matches.py
from datetime import date, timedelta
from app.db.session import SessionLocal
from app.ingestion.sync_jobs import sync_day
```

```
from app.config import DEFAULT_LEAGUE_ID


if __name__ == "__main__":
    session = SessionLocal()
    # example: last 30 days
    today = date.today()
    for i in range(30):
        d = today - timedelta(days=i)
        print(f"Syncing {d}")
        sync_day(session, d, league_id=DEFAULT_LEAGUE_ID)
    session.close()
```

---

5. Feature builders

5.1. app/features/prematch_builder.py

```
from dataclasses import dataclass
from typing import List, Optional
import numpy as np
from sqlalchemy.orm import Session
from app.db.models import Match, MatchStats, RatingHistory
from app.config import DEFAULT_LEAGUE_ID


@dataclass
class PreMatchExample:
    match_id: int
    X: np.ndarray
    y_outcome: Optional[int]
```

```python
    y_home_goals: Optional[int]
    y_away_goals: Optional[int]


class PreMatchFeatureBuilder:
    def __init__(self, session: Session, rating_type: str = "elo"):
        self.session = session
        self.rating_type = rating_type

    def _get_team_rating(self, team_id: int, match_date) -> float:
        rh = (
            self.session.query(RatingHistory)
            .filter_by(team_id=team_id, rating_type=self.rating_type)
            .filter(RatingHistory.date <= match_date)
            .order_by(RatingHistory.date.desc())
            .first()
        )
        return rh.rating if rh else 1500.0

    def build_examples(self, league_id: int = DEFAULT_LEAGUE_ID) -> List[PreMatchExample]:
        matches = (
            self.session.query(Match)
            .filter_by(league_id=league_id)
            .order_by(Match.date)
            .all()
        )
        examples: List[PreMatchExample] = []
```