Sports Betting Prediction Project - Current Code Documentation

**1. Overview**

This project is a football (soccer) betting prediction system. It collects match data from a PostgreSQL database, computes historical features (like Elo ratings, team form, head-to-head advantage, and bookmaker odds), trains an XGBoost model, and exposes predictions via a FastAPI REST API.

**2. Components**

**2.1 api.py**

Defines the FastAPI application and the /predict endpoint. Receives match details (home\_team, away\_team, league, country, date), calls prepare\_single\_match\_features from train\_and\_save.py, and returns prediction probabilities. Includes error handling.

**2.2 train\_and\_save.py**

Contains all training, feature engineering, and prediction logic.

Main responsibilities:

- Load historical matches from PostgreSQL.  
- Parse results into labels (0=Home, 1=Draw, 2=Away).  
- Compute features: Elo ratings, team form, head-to-head, bookmaker implied probabilities.  
- Train an XGBoost model on these features.  
- Save artifacts (model + encoders + features).  
- Provide functions to prepare single match features and generate predictions.

**2.3 sample\_out.py**

A simple client script to test the API. Sends a POST request with match details, prints the raw response, and displays prediction probabilities.

**3. Data Flow**

1. Historical data is stored in a PostgreSQL database (schema: bettingschema.odds).  
2. train\_and\_save.py loads this data and computes features.  
3. Model is trained and saved in ./artifacts.  
4. API loads model artifacts when serving predictions.  
5. Client (sample\_out.py) calls API with new match info.  
6. API computes features for the match, runs prediction, and returns probabilities.

**4. Features Computed**

- Elo ratings (home, away, difference).  
- Team form: wins, points in last N matches.  
- Head-to-head advantage (last meetings).  
- Implied bookmaker probabilities (from odds).  
- Encoded categorical features (country, league).

**5. Model**

Model: XGBoost multiclass classifier (3 outcomes: home win, draw, away win).  
Training uses a time-aware split (last 10% validation).  
Saved artifacts include model, country/league encoders, and feature list.

**6. API**

Endpoint: POST /predict  
Input: JSON with home\_team, away\_team, league, country, date.  
Output: JSON with success flag, probabilities (p\_home, p\_draw, p\_away), and error message if applicable.

**7. Current Warnings & Issues**

- FutureWarning messages from pandas about DataFrame concat and fillna downcasting.  
- If encoders don't recognize new leagues/countries, they return -1.  
- Feature engineering assumes structured historical data is present.

1️⃣ Data in your database

From what you described:

Historical matches: Finished matches from 2021 up to now.

Future matches: Scheduled matches in 2025.

Scope: First league of ~20 countries.

Columns include: season, date, time, home\_team, away\_team, result, odd\_1, odd\_X, odd\_2, bets, country, league.

2️⃣ Loading the data for training

The function load\_matches() does:

SELECT ... FROM bettingschema.odds

WHERE date IS NOT NULL

ORDER BY date ASC;

So all matches with a date (finished matches and scheduled matches with a date) are loaded.

df['date'] = pd.to\_datetime(df['date']) ensures dates are in timestamp format.

Important: Only matches that have a result will contribute to the training labels. Future matches (result = NULL) are ignored for training.

3️⃣ Feature engineering

The function compute\_features(df, n\_last=5) does the following:

Converts result into a numeric target:

0 = Home win

1 = Draw

2 = Away win

Calculates implied probabilities from bookmaker odds (odd\_1, odd\_X, odd\_2).

Calculates Elo ratings for home and away teams.

Builds recent form features:

Last n match points/wins for home and away teams.

Calculates head-to-head advantage for home vs away team.

Fills missing values with defaults.

Output: df with features for every match, including future matches.

4️⃣ Training the model

The function train\_save\_model(df\_feats, feat\_cols):

Filters only matches with a known target:

df\_train = df[df[target\_col].notna()].copy()

So only finished matches are used for training.

Future matches (next 2025 matches) are ignored for training.

Encodes categorical features: country and league.

Defines features:

used\_features = feat\_cols + ['country\_enc','league\_enc']

Splits data into train/validation using time-based split (90% train, 10% validation).

Trains an XGBoost model with multi:softprob (predicts probabilities for Home, Draw, Away).

5️⃣ Predicting future matches

The function prepare\_single\_match\_features():

Loads all historical matches before the match date.

Appends a dummy row for the future match.

Recalculates features with compute\_features() for the historical + future row.

Uses the trained XGBoost model to predict probabilities for the future match.

✅ Key point: The model does not train on future matches, only historical data is used. Future matches are only for feature computation and prediction.

Summary

Training data: Historical matches with known results (from 2021 to now).

Validation: Last 10% of historical matches by time.

Future matches: Only used for computing features and prediction, never used in training.