Injury Analytics Project

Amina ADDI, Anis MOUNSI, Chloé BURKOVIC, Mahek MAKWANA, Harolde KONNON

What this is.

The Injury Analytics app estimates a player's injury risk and helps staff spot player profiles with similar patterns. It has three parts:

1. A prediction model that turns player/match info into a risk score and a clear "Injured / Not Injured" verdict,
2. An exploration module that groups players into profiles
3. A simple Streamlit interface for loading data, training or reusing a saved model, and scoring a player in seconds.

The data

We use two tables :

1. Main match-level table (real data).
   Each row is an injury event with: player info (team, position, age, FIFA rating), injury and return dates, and how the team and player performed in the three matches before, the period missed, and after the injury (results, goal difference, player ratings). From this, we create a label telling us if the player has a future injury within a set time window. This makes it a time-based prediction task.
2. Fitness table (synthetic, for demos).
   Contains practical metrics like knee strength, hamstring flexibility, reaction time, sleep, stress, nutrition, and warm-up adherence, plus a balanced "injury next season" label. Great for demos and charts—not for final decisions.

We keep processing light and consistent: parse dates, sort by player and time, compute simple rolling averages just before the injury (e.g., average rating, average goal difference), and turn categories (team, position, injury type) into model-friendly indicators.

- Typical columns: Age, FIFA rating, Position, Team, Injury type, Injury/return dates, recent averages of rating and goal difference, recent match counts.
- Derived features: rolling means (rating, GD), recent workload counts, one-hot flags for injury families.

How predictions work

We combine two techniques:

- Random Forest (main model).
  Many decision trees vote to produce a score. It handles mixed data, captures non-linear

patterns, and works with little tuning. We save the trained model and the exact input columns together, so scoring always matches training.

- Logistic Regression (simple, transparent baseline).
  Produces a probability and lets us adjust the decision threshold (useful when missing a real injury risk is worse than a false alarm). It's easy to interpret and great for sanity checks.

For exploration :

- K-Means clustering (+ PCA view).
  Groups players into profiles after standardizing numbers. We pick a sensible number of groups using the silhouette score (and an elbow check) and display them on a 2D PCA plot for an easy visual map.

At a glance :

- Random Forest: impute → one-hot encode → RF (~300 trees). Robust; probabilities may need calibration.
- LogReg: impute → scale numbers → one-hot encode → Logistic Regression. Interpretable; may miss complex patterns.
- Clustering: standardize → try several k → pick by silhouette → PCA plot.

What you see in the app

Streamlit gives a single, clean screen:

1. Load data. Use the default CSV or upload your own; the app caches it and applies the same cleaning every time.
2. Train or load. If a model already exists, it loads instantly; otherwise it trains once and saves it.
3. Score a player. Fill a short form or select a row. The app rebuilds inputs in the same format as training and returns:
   - A probability of injury
   - A clear verdict (“Injured” / “Not Injured”)
   - Optional drivers (feature importance)
4. Visuals. Confusion matrix, optional ROC/PR curves, global feature importance, and the cluster map to understand profiles.

How we judge quality (and why)

Because we predict future injuries, we avoid using future info during training. Instead of random splits, we use time-aware splits (train on earlier seasons, test on later) or a rolling window. We report standard metrics (Accuracy, Precision, Recall, F1) and add PR-AUC when the injury class is rare (it focuses on catching true at-risk cases).

- Good metrics to show: Confusion matrix; Precision/Recall/F1; PR-AUC; optionally ROC-AUC and Brier score.
- Probability calibration: If we say "30% risk," roughly 3 out of 10 should get injured. Platt or isotonic calibration improves this alignment.
- Explainability: Use permutation importance for a stable global ranking (Random Forest) and SHAP for per-player "why" explanations.

Strengths, caveats, next steps

What works well.

Clear separation of data prep → modeling → evaluation → serving; saved model and input schema prevent mismatch; clustering adds context so staff think in profiles, not just scores.

Watch outs.

Random splits can leak future info. Raw probabilities may be uncalibrated, calibrate when decisions rely on the number. Prove generalization to new teams/seasons with proper hold-outs.

Next steps

- Adopt time-aware cross-validation and show results by season.
- Calibrate probabilities; track the Brier score.
- Add SHAP in the app for per-player transparency.
- Log runs and artifacts with a light model registry (e.g., MLflow).
- Enforce an input schema (columns + dtypes) at upload to avoid scoring errors.