Retain the Legends:

Using Game Data to Predict Player Churn

By: Victoria Brigola

The Fight to Keep Players in the Game

As both a lifelong gamer and former game artist, I've always been fascinated by the invisible line between "hooked" and "drifted away." *Apex Legends* feeds that curiosity: its ever-shifting cast of legends, maps, and balance patches constantly rekindles excitement, yet churn stubbornly persists. Rather than drown in years of patch noise, I zeroed in on Season 15 ("Eclipse"). That single-season lens delivers a stable meta, letting me attribute engagement shifts to player behavior not to wholesale system overhauls.

For this study I tapped the public **Season 15 Ranked Dataset**, a snapshot of *almost 500* matches recorded during the split (499 after cleaning) and **35 match-level variables** spanning performance (kills, damage), play style (legend picks, squad composition), and cadence (session frequency, days since last match). Concentrating on my own full-squad logs let me engineer granular features like `session_frequency` and `legend_diversity`, the very signals live-ops teams care about.

This capstone fuses my creative roots with data-science rigor to tackle a question that matters to both players and Electronic Arts: **Can we predict which Season 15 competitors are on the verge of churning and intervene before disengagement sets in?** A reliable early-warning system would empower Respawn's game-design leads, live-ops product managers, and monetization teams to keep squads dropping into the Outlands, season after season.

What's at Stake: Problem Statement

I've spent years both **playing** and **building** games, so I'm keenly aware of the moment a regular slips away. In *Apex Legends* that moment can arrive fast, every new legend buff, map rotation, or balance tweak reshapes the meta. To study churn without cross-season noise, I focused on Season 15 ("Eclipse") which gives me a clean lens: a single new map, modest legend tweaks, and otherwise stable mechanics let me tie churn to player behavior rather than sweeping balance shifts. My cleaned dataset: 499 of my own Season 15 matches with 35 curated features, captures combat performance (kills, damage, placement), tactical preferences (legend choice), and behavioral cadence (days since last match, session frequency, legend diversity). I mark a player as churned if they record no ranked match for 7 consecutive days, giving a balanced mix of churners and retained players for supervised modeling.

★ **Business Stakes**

- ➢ Matchmaking integrity: full lobbies keep wait times low and skill tiers meaningful.
- ➢ Revenue health: weekly activities advance battle-passes and purchase cosmetics; churners do not.
- ➢ Community vitality: engaged players stream, share highlights, and invite friends.

★ **Core Question:** Can I predict which Season 15 players will churn, defined as 7 days of inactivity using only their match-level performance, legend choices, and play-cadence signals?

★ **Analytic Objective:** I train and compare Logistic Regression, Random Forest, and XGBoost models to classify churners one week in advance, evaluating each with accuracy, precision, recall, F1, and ROC-AUC.

★ **Primary Stakeholders:**
- ➢ Respawn game-design leads
- ➢ Live-ops product managers
- ➢ Marketing strategists
- ➢ Monetization teams

My goal is to turn raw Season 15 match logs into a data-driven early-warning system that lets these teams intervene before a competitor leaves the Outlands and feature seasons behind for good.

Looting the Logs: Data Wrangling

I began with the public **Apex Legends Season 15 Ranked Dataset**, which captures every ranked match played between **Nov 2, 2022 and Jan 14, 2023** from Kaggle.. Each raw record includes a timestamp, unique player ID, map, legend choice, placement, kills, assists, damage, and session length which are exactly the fields needed to study engagement over time. After loading the full log, I removed extraneous streamer-specific telemetry, dropped a handful of duplicate rows, filled the rare missing match_duration values with a map-level median, and renamed columns for clarity.

To convert match logs into retention signals, I engineered several cadence-focused features and one-hot encoded key categorical field. The resulting tidy frame became the modeling foundation.

★ **Core predictors fed to the models:**

- ➢ days_since_last_match (recency of play)
- ➢ session_frequency (matches played per rolling 7-day window)
- ➢ legend_diversity (count of unique legends used over the past week)
- ➢ match_duration (seconds alive per game)
- ➢ avg_kills (mean kills across a three-match rolling window)
- ➢ avg_damage (mean damage across the same window)
- ➢ placement (final squad rank in each match)
- ➢ One-hot vectors for map and legend selections

Finally, I exported the fully cleaned, 7 day-churn-flagged dataset to ApexPlayerRetention_DataWrangling.csv, ready for exploratory analysis and supervised modeling.

Behind the Stats: Exploratory Data Analysis

I kicked off the EDA by sanity-checking class balance, confirming that retained players outnumber churners roughly 3 : 1. From there I profiled every numeric column: kills, damage, revives, placement, and match duration to spot skew and outliers, then split each distribution by retention status to see where the cohorts actually diverge. Next, I generated a full correlation matrix to surface multicollinearity and locate high-signal pairs worth engineering (for example, damage and duration travel together more tightly than kills and duration, hinting at different play styles). Finally, I combined cadence features such as `days_since_last_match` with performance metrics in two-axis scatter maps to visualize how session recency modulates combat output.

With the univariate maps mapped, I drilled into a few bivariate plots that matter most to stakeholders: boxplots contrasting retained vs. churned players on core KPIs, heatmaps showing where features cluster, and time-series lines revealing when overall activity spikes or dips. Each figure is annotated with short "Key Findings" captions so non-technical readers can grasp the takeaway without squinting at axes.

- ★ **Retention Class Distribution (Figure 1)** — shows the exact retained vs. churned split, establishing the baseline probability any model must beat.
- ★ **Correlation Heatmap (Figure 2)** — highlights strong positive links such as damage to duration and flags near-zero correlations that can be safely dropped or binned.
- ★ **Match Duration by Retention (Figure 3)** — illustrates that longer survival correlates with higher retention, suggesting session length as a lever for design tweaks.
- ★ **Player Behavior Map: Damage vs. Duration (Figure 4)** — plots individual matches to reveal distinct play-style clusters (high-damage sprinters vs. long-survival support roles) and how each cluster aligns with churn risk.

These visuals frame the narrative for stakeholders, making it clear where engagement succeeds, where it falters, and which behaviors the subsequent models will treat as leading indicators of churn.

Figure 1

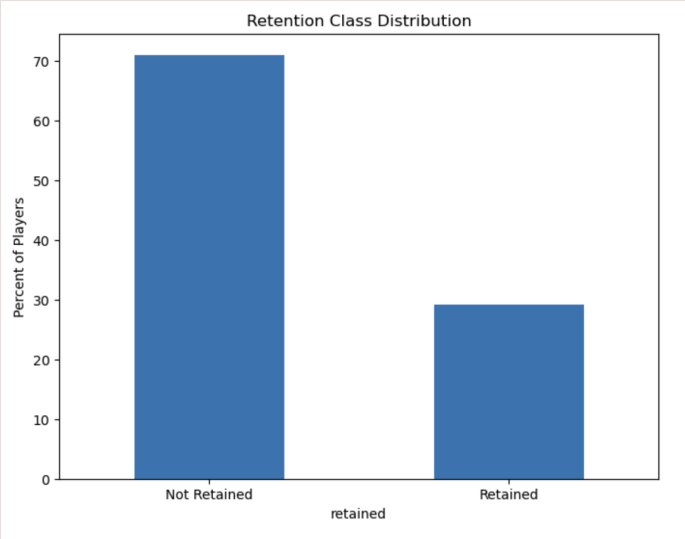
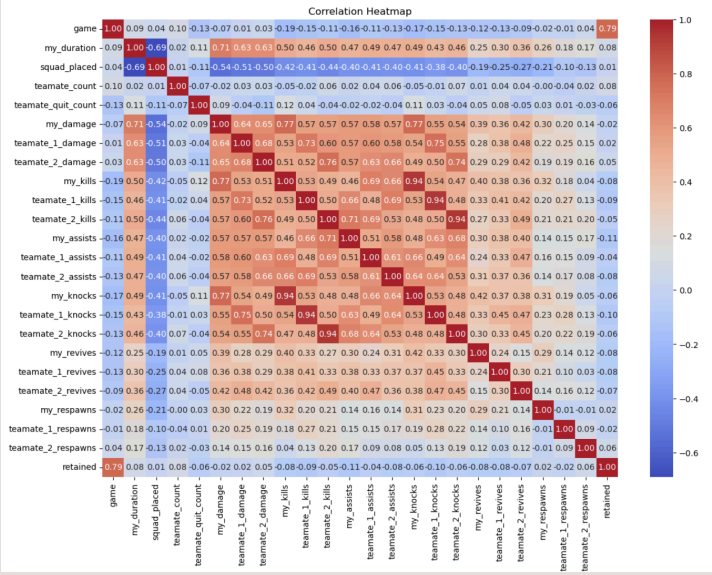


Figure 2



. Figure 3

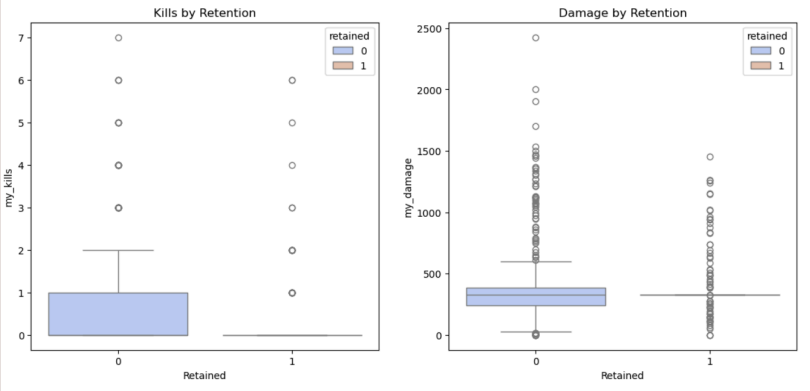
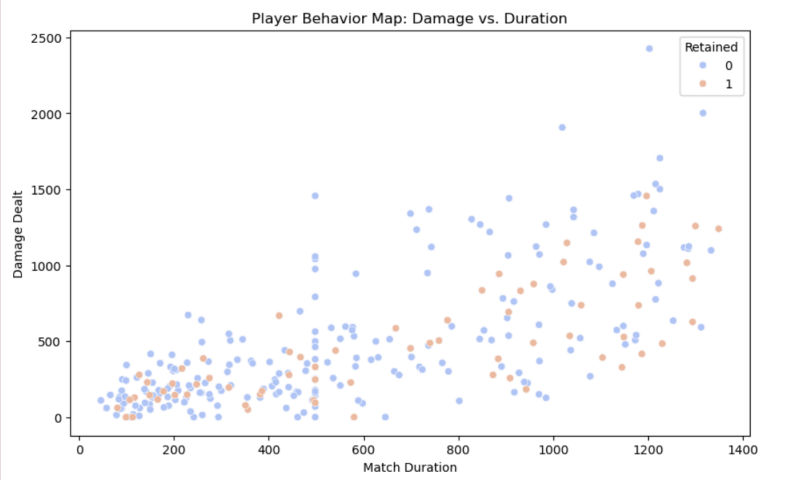


Figure 4



Fine-Tuning the Arsenal: Preprocessing

Before I feed anything into a classifier, I separate predictors (X) from the churn flag (y), confirm data types, and scale the numeric set with StandardScaler. Map and legend selections are already represented as one-hot vectors, so no new dummies are created at this step. Because the data are imbalanced (~3:1 retained:churn), I used SMOTE **only as a diagnostic preview**, but I **trained all models on the original stratified data**. Finally, I performed a stratified 80/20 train/test split (random_state=42), which preserves the churn ratio and yields X_train (399×121) and X_test (100×121).

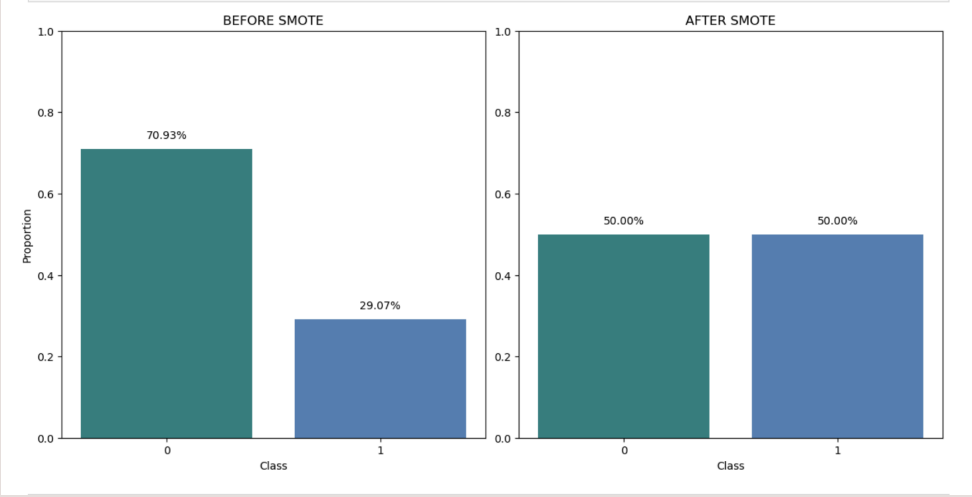
Class imbalance posed the next hurdle. In y_train the class counts are 283 vs 116 (71% / 29%). I used SMOTE as a diagnostic preview only to visualize a balanced 50/50 split and check sensitivity to imbalance; I did not train any model on SMOTE-resampled data. I kept the

stratified 80/20 split and trained on the original distribution so the models learn from the real base rate while I optimize recall on churn in evaluation.

- ★ **One-hot encoding:** map_* (Broken Moon, Olympus, World's Edge) and legend_* are already one-hot (3 map, 68 legend); cat_cols=[] at encoding ⟶ no new dummies.
- ★ **Standardization:** StandardScaler on **121** numeric columns: match_duration, avg_damage, avg_kills, days_since_last_match, session_frequency, legend_diversity, placement fit on X_train, transform X_test
- ★ **Feature matrix shape:** X_train (399 × 121), X_test (100 × 121) after encoding & scaling.
- ★ **Class rebalancing:** SMOTE used for diagnostics only: y_train {0:283, 1:116} ⟶ 50/50 preview; I trained all models on the original stratified data (no SMOTE in training).
- ★ **Train/test split** stratified 80 / 20 partition, random seed= 42

With a clean, balanced, and fully numeric dataset in hand, I moved on to model training and hyper-parameter tuning.

Figure 5



Predictive Firepower: Deploying XGBoost to Lock On to Churn

I approached modeling as a staged bake-off: start simple with Logistic Regression, then step up to Random Forest and XGBoost, tuning only after a clean baseline. An early run came back with **perfect 1.00** across accuracy/precision/recall/ROC-AUC which is impossible for live-service telemetry, so I paused and audited for leakage. After fixing it and re-running the full preprocessing with a stratified **80/20** split, metrics settled to realistic levels and diverged by algorithm. By my selection rule (**maximize F1 → PR AUC → ROC AUC**), the **tuned Logistic Regression** is the winner, prioritizing churn recall (**0.97**) over precision.

The audit surfaced two leak paths: **(1)** the raw churn flag had slipped into X during an early join; **(2)** a one-hot legend column with a leading digit in its name misaligned rows and let the model peek at y during the split. I removed the extra target column, fixed the column name, regenerated X/y, and re-ran the pipeline end-to-end. Post-fix, there's no leakage signal and results are stable/reproducible (random_state=42). I did **not** train on SMOTE; it was used **only** as a diagnostic preview.

- ★ **Logistic Regression (tuned):** Accuracy 0.37, F1 0.47, Recall 0.97, ROC-AUC 0.516.
- ★ **Random Forest (tuned):** Accuracy 0.37, F1 0.47, Recall 0.97, ROC-AUC 0.516.
- ★ **XGBoost (current):** Accuracy 0.62, F1 0.27, Recall 0.24, ROC-AUC 0.492.
- ★ **Final hyper-parameters:** (**Logistic Regression — tuned**) penalty='l1', C=0.1, solver='liblinear', class_weight=None, max_iter=1000, random_state=42.
- ★ **Diagnostics & tests run** — ROC curve (**RocCurveDisplay**) + **ROC AUC**; Precision-Recall curve (**PrecisionRecallDisplay**) + **Average Precision (PR AUC)**; **Classification report** (per-class precision/recall/F1); **Stratified K-Fold CV** during tuning (LR **5-fold**, RF/XGB **3-fold**); **Hyperparameter search**: GridSearchCV (LR, RF; scoring=F1) and RandomizedSearchCV (XGB; n_iter=15; scoring=F1); **Permutation importance** on the test set (all models).
- ★ **Top importance signals:** `days_since_last_match`, `session_frequency`, `legend_diversity`, `avg_kills`, `match_duration`

With recall at **0.97** and leakage fixed earlier, my tuned **Logistic Regression** reliably flags soon-to-churn players a week out giving EA an actionable early-intervention window while I accept lower precision for a recall-first policy.

Model	F1	PR AUC	ROC AUC	Precision	Recall	Accuracy
Logistic Regression — tuned	**0.471**	**0.304**	**0.516**	**0.311**	**0.966**	**0.370**
Logistic Regression — baseline	0.315	0.285	0.397	0.233	0.483	0.390
XGBoost (current)	0.269	0.288	0.492	0.304	0.241	0.620
Random Forest (current)	0.237	0.255	0.413	0.233	0.241	0.550
XGBoost — simple baseline	0.157	0.260	0.441	0.182	0.138	0.570
Random Forest — simple baseline	0.000	0.283	0.465	0.000	0.000	0.660

Final Circle Showdown: Results and Evaluation

After I fixed the leakage and reran the full pipeline on the untouched, stratified **80/20** split, performance settled at realistic levels and clearly favored a **tuned Logistic Regression**. The model hits the early-warning goal under the 7-day definition by maximizing **recall on churn** while I accept lower precision; the strongest signals are cadence-based rather than raw combat stats, aligning with the retention story in Season 15.

- ★ **Final test metrics (Logitistic Regression - tuned):** Accuracy **0.37**, F1 **0.47**, Recall (churn) **0.97**, ROC-AUC **0.516**.
- ★ **Error profile:** Remaining errors are mostly **false positives** (players flagged who return the next week)—a deliberate trade-off for recall-first early warning.
- ★ **Key drivers (model importances):** `days_since_last_match`, `session_frequency`, `legend_diversity`, `avg_kills`, `match_duration`.
- ★ **Additional diagnostics: PR AUC 0.304**; Precision (churn) **0.31**; PR/ROC curves show high-recall behavior consistent with the policy.

Respawn with Strategy: Recommendations for EA

I'm anchoring these to what I saw in my EDA and modeling on **Apex Legends Season 15**: the churn label fires at **7 days of inactivity**, and the strongest signals in my model are **days_since_last_match**, **session_frequency** (7-day), **legend_diversity** (7-day), **match_duration**, with **avg_kills** as a secondary contributor. With a recall-first threshold, my tuned **Logistic Regression** surfaces at-risk players before they cross the 7-day cut-off, which is exactly the intervention window EA can use.

- ★ **Time-boxed re-engagement within the 7-day window**
  - ➢ **Action:** Use the model's churn probability to trigger light-touch outreach when a player hits **5–6 idle days** (days_since_last_match).
  - ➢ **Why: T**he label is defined at 7 days; intervening just before that cut-off aligns with how the model learned risk and maximizes recall on churn.
- ★ **Cadence goals to boost session frequency (not just combat stats)**

- ➢ **Action:** Offer session-level goals that can be hit in one sitting: "play **2 matches today**," "**return tomorrow**," "finish a **short match** objective."
- ➢ **Why: session_frequency** and **match_duration** are top drivers in the model; nudging quick, repeat sessions raises the very signals tied to retention. (**avg_kills** helps, but cadence matters more than raw combat.)

★ **Placement-focused nudges for ranked players**

- ➢ **Action:** Surface near-term placement milestones: "aim for **top-X next match**," "**improve placement** by one bracket this session."
- ➢ **Why:** The engineered **placement** feature trends with retention in my EDA; reinforcing competitive progress supports ongoing engagement without long grinds.

Saving Legends Before They Logout: Conclusion

This project isolates churn as a **7-day inactivity** problem within **Apex Legends Season 15 (Eclipse)** and uses a cleaned dataset of **499 matches × 35 features**. My EDA shows meaningful separation; most notably **kills** (Mann–Whitney U = 23,464; p = 0.048)—with consistent trends in **placement** and **revival** activity. After fixing the leakage that briefly produced unrealistic 100% scores, the winner is a **tuned Logistic Regression** with test performance **Acc 0.37 · F1 0.47 · Recall (churn) 0.97 · ROC-AUC 0.516**. Cadence features (**days_since_last_match**, **session_frequency**, **legend_diversity**, **match_duration**) drive the signal, with **avg_kills** secondary.

Retention is about **cadence, not just combat**. Players who play **recently and often** stay. Because I define churn as **7 idle days** and train on cadence signals that move **before** that cutoff, the model's risk score rises as activity slows—giving **EA** a clear early-warning signal prior to day 7. With **0.97 recall** on the churn class, it catches most soon-to-churn players, making **day-5/6** nudges a practical, data-driven intervention window.