Yard Reaas

Giovanni Minuto

October 2025

1 Introduction

This report documents the work carried out as part of the Data Scientist – Take-Home Exercise. The test was structured into a series of assignments:

1. Data Engineering – building a Spark-based ingestion pipeline and materialising a bronze $\rightarrow$ silver $\rightarrow$ gold architecture to ensure reproducibility, performance, and data quality.
2. Exploratory Analysis – computing engagement KPIs such as DAU/WAU/MAU, analysing diversity and session length, and investigating data quirks (e.g., sparsity, leakage, seasonality).
3. Feature Engineering & Modelling – framing churn as a binary classification problem, designing features from behavioural logs, training a predictive model, tuning hyperparameters, and applying model explainability techniques. Churn definition: an entity (in our case a page title with his feature) is considered churned if it is present in the observation window but absent from the label window.
4. BI Dashboard and Data Storytelling – developing an interactive dashboard to communicate churn dynamics, cohort behaviour, and model-driven risk scores in an accessible way for non-technical stakeholders.

The purpose of this report is to provide a clear overview of the methodology, the decisions taken at each stage, and the analytical insights obtained.

2 Preliminary Info

2.1 Code

All the code is available at the repository [1] for the installation part a readme is provided

2.2 Setup

The setup is described in Table 1.

Environment	**Specifications**
Databricks Community Edition	Single-node cluster, 15 GB RAM, 2 vCPU
Local Laptop	Apple M1 (8 cores), 16 GB RAM, 512 GB SSD
Programming Language	Python

Table 1: Setup

2.3 Database

The dataset provided for this exercise consisted of the Wikimedia Pageviews for January 2025[2], amounting to approximately 70 GB of compressed data. This large-scale dataset required careful engineering of ingestion pipelines, schema design, and data-quality checks to enable efficient exploration and modelling.

The global database was extremely large. To make the machine learning workflow computationally feasible on the available hardware, I restricted the predictive modelling to the Italian section of Wikipedia.

The code, however, was written in a general way and could be applied to any language. In this report, results of the churn prediction are therefore shown for the Italian subset, but the methodology remains language-agnostic. In contrast, the exploratory KPI analysis (e.g., DAU, MAU and stickiness,) was conducted across multiple languages, providing a broader view of user behaviour beyond the Italian case.

3 Data Engineering

In this section, we describe the data ingestion and transformation pipeline designed to process the Wikimedia Pageviews dataset. The workflow follows a multi-layered architecture inspired by the *Bronze–Silver–Gold* paradigm commonly adopted in modern data lakes. [3]

Silver layer: The Bronze data was then refined through quality checks and domain-specific transformations, producing a cleaner and analytics-ready dataset. At this stage, information encoded in the raw domains (e.g., *page_title*, *namespace*) was decoded using official Wikimedia documentation [4].

In the following subsections, we present in detail the steps taken at each stage of the pipeline.

3.1 Bronze Step: Raw Ingestion & Minimal Cleaning

The Bronze layer is responsible for ingesting the raw Wikimedia Pageviews and storing them in a structured format suitable for downstream processing. To support efficient querying and parallelism, we adopted a daily partitioning strategy, resulting in 31 partitions corresponding to the days of January 2025.

A fixed schema, provided by Wikimedia[4] and reported in Table 2, was applied to ensure consistency across files and enable schema-on-read validation. This schema defines the core fields of the pageviews dataset:

Schema Feature	**Description**
domain_code	Domain name of the request, abbreviated.
page_title	Title of the unnormalized part after `/wiki/` in the request URL.
count_views	Number of times this page has been viewed in the respective hour.
$total\_response_size$	Always zero (not used).

Table 2: Wikimedia Pageviews Schema

During ingestion, only valid files are preserved. Metadata from the filename itself is parsed to extract the ingestion date and time, ensuring that each record is enriched with its collection timestamp. This guarantees full traceability of the data source.

Finally, the Bronze table is stored in Delta format, which provides durability, ACID guarantees, and efficient storage management. The chosen partitioning key is the ingestion day, which balances performance and storage overhead while aligning with the temporal nature of the dataset.

3.2 Silver Step: Cleaning & Structuring

The Silver layer takes the raw-but-structured Bronze data and applies quality checks and domain-specific transformations to make it cleaner and analytics-ready.

The Silver step represents the intermediate refinement phase of the pipeline. Its purpose is to convert the raw pageview logs from the Bronze layer into a standardized dataset suitable for downstream aggregation and analysis. The following transformations are applied:

1. **Quality Guards:** Records are filtered to ensure validity. Only entries with a number of views in the range $[0, 10^9]$ are retained, while rows with missing $domain_code$, null titles, or placeholder titles ("-") are discarded[4].

2. **Domain transformation:** The *domain_code* is parsed to derive three attributes: (i) the *language*, identified through a mapping expression; (ii) the *database name*, determined from domain substrings (e.g., `voy` $\rightarrow$ Wikivoyage, `s` $\rightarrow$ Wikisource); (iii) a Boolean flag *is_mobile*, indicating requests originating from mobile subdomains. Special domains such as *commons*, *meta*, or *species* are explicitly excluded.

3. **Removal of redundant columns:** The column *total_response_size* is dropped, as it is composed only by zeros.

4. **Page title transformation:** The field *page_title* is split into two components: (i) the *namespace*, capturing prefixes such as `Special:`, `Category:`, or `User:`; (ii) the article title. If no prefix is present, the default *namespace* `Article` is assigned. (iii) To ensure consistency, article titles are normalized by: removing leading/trailing underscores, replacing underscores with spaces, discarding parenthetical content, truncating text following slashes, replacing unwanted symbols (e.g., quotes, dollar signs) with spaces, collapsing multiple spaces, trimming whitespace, and converting all text to lowercase.

Now the output is a clean, consistently structured Silver table in Delta format, suitable for feature engineering, aggregations, and the higher-level transformations of the Gold layer.

3.3 Gold Step: Aggregation & Analytics Outputs

In the Gold layer, the refined Silver data was aggregated into higher-level datasets specifically designed for analysis and machine learning. These aggregations facilitated the efficient computation of KPIs and provided the feature sets required for churn modelling.

As a first step, the Silver database was split by language; for the churn analysis described in this report, we focus on the Italian subset. This transformation from Silver to Gold was necessary because the Silver dataset remained too granular to be used directly for machine learning. Instead, we constructed aggregated and engineered features that summarize the behavior of each key entity over a defined observation window. A binary churn label was then assigned based on whether the same entity was present in the subsequent label window.

In addition, the Gold layer implements a dedicated function to handle *late records*, ensuring that delayed ingestions or out-of-order files do not compromise data consistency. This mechanism guarantees that the final machine learning database remains reliable and reproducible.

The detailed design choices and methodology used to create the machine learning dataset are described in Section 4.2.

4 Exploratory Analysis

This section is divided into two parts. In the first part, we investigate the main properties of the Wikimedia pageviews database by computing engagement KPIs, examining content stickyness, and visualizing temporal trends. The goal of this analysis is to generate feature ideas and formulate retention hypotheses based on observed usage patterns.

In the second part, we describe the construction of a feature-rich dataset tailored for churn prediction. Particular attention is given to handling the inherent challenges of this data, such as seasonality, sparsity, and potential leakage. By addressing these issues, we ensure that the final dataset is both robust and suitable for training reliable predictive models.

4.1 Trend Wikipedia Pageviews Database

DAU for Wikipedia pageviews In this analysis we investigate the distribution of daily active usage (DAU) across five languages of Wikipedia pageviews: Italian, English, German, Spanish, and French. The process begins by extracting language-specific pageview data from wikipedia pageview database. From these sampled datasets we compute DAU, defined as the number of distinct pages viewed per language per day. To make results comparable across languages, we then compute the daily total DAU (summing over all five languages) and normalize each language's DAU by this total. This produces a percentage share of distinct pages viewed per language per day. The normalized values allow us to track the relative contribution of each language, independent of differences in absolute scale. From these plots, in Fig. 1, we observe that English consistently holds the largest daily share of distinct pages viewed. The other four languages together make up the remainder, each contributing a smaller but still noticeable portion. Day-to-day fluctuations reflect natural variation in traffic patterns, but the general trend is stable: English always exceeds any single other language, and in percentage terms it usually constitutes the majority of daily distinct pages.

Stickiness metric for Wikipedia pageviews In this analysis we compute and visualize the DAU/MAU stickiness metric for Wikipedia pageviews across five languages: Italian, English, German, Spanish, and French, focusing on January 2025. For each language, we first calculate DAU. We then calculate MAU

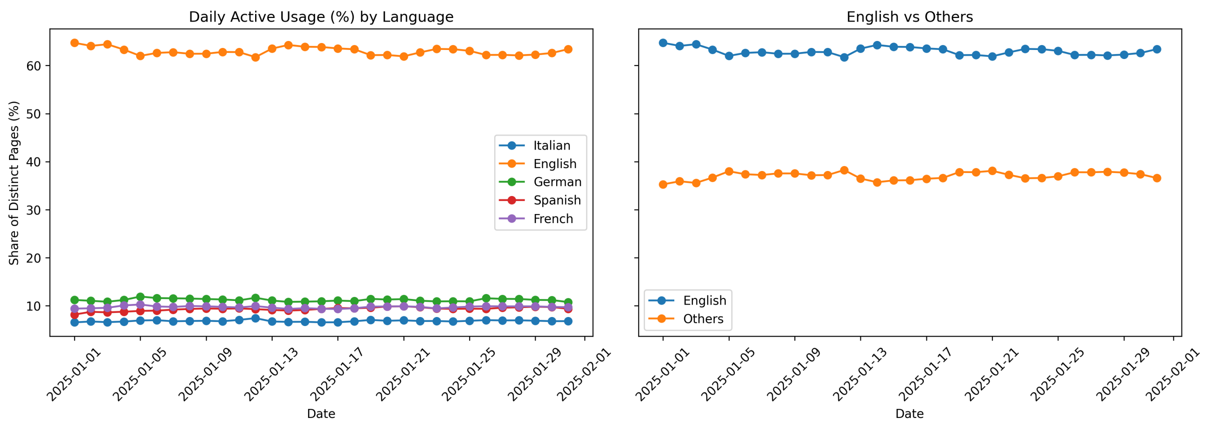

Figure 1: DAU across five languages. In the left plot a line chart showing the daily percentage share of DAU for each of the five languages individually. In the right plot a side-by-side comparison where English is contrasted against the aggregate of the other four languages ("Others").

(Monthly Active Usage), defined as the number of distinct pages viewed at least once within the month. By combining these two kpis, we obtain stickiness, expressed as the percentage ratio DAU/MAU. Stickiness therefore measures the extent to which the monthly active set of pages is also active on a given day. A higher value indicates more stable and consistent engagement, while a lower value reflects greater day-to-day variability in pageviews.

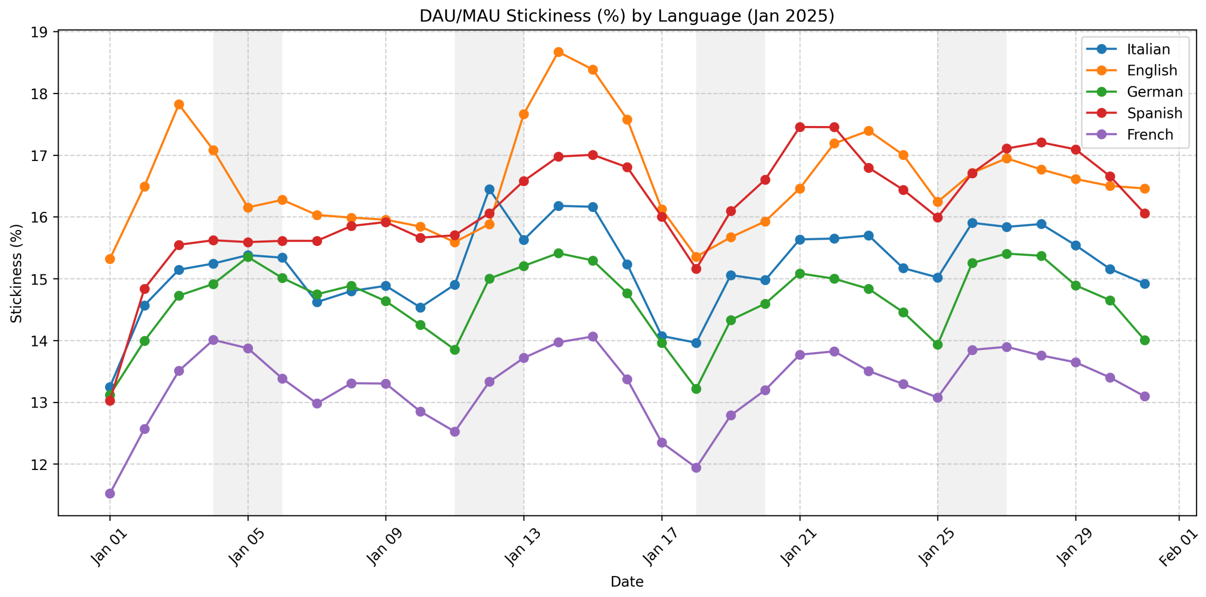

Figure 2: Daily DAU/MAU stickiness for Wikipedia pageviews across five languages (January 2025). Shaded areas correspond to weekends, where a systematic decrease in stickiness is observed.

The results, shown in Fig. 2, display daily stickiness curves for each language, with weekends highlighted for reference. All languages exhibit a similar trend: stickiness decreases during weekends and holiday periods, such as the beginning of January. This behavior is expected. On weekdays, users tend to revisit a relatively stable set of pages—such as reference material, news, or academic content—leading to higher overlap between daily and monthly activity. On weekends, however, browsing shifts toward more diverse or leisure-oriented content, reducing the overlap with the monthly active set and lowering stickiness.

4.2 Construction of a feature-rich Dataset

In churn prediction (or more broadly in behavioral modeling), raw logs are typically too granular to be directly useful. Each pageview entry corresponds to a single event, but machine learning models require structured feature vectors that summarize behavior over time. Aggregating features serves several purposes, such as reducing noise and sparsity, capturing meaningful signals, and ensuring comparability across entities. In our case, we rely on **daily aggregation** as the fundamental temporal unit. This choice balances granularity and interpretability: while hourly data is subject to high variance due to short-lived bursts of activity, daily totals provide smoother patterns that reflect sustained behavior. At this level, meaningful temporal properties such as recency, weekly seasonality, and long-term decay can be observed, making daily aggregation a natural basis for feature construction.

In this part, we present the aggregate features that compose our database for machine learning. For each feature we selected, we provide the underlying motivations as well as some alternatives that were considered but not adopted, highlighting the reasons for these design decisions.

The dataset is divided into four disjoint time windows:

1. **Training observation window:** 2025-01-01 to 2025-01-23. Used to compute activity features (e.g., total views, variability, recency). Only information from this period is available to the model, preventing leakage.

2. **Training label window:** 2025-01-24 to 2025-01-27. Defines the churn target: an entity is labeled *churned* ($churn = 1$) if it appears in the observation window but not in this period; *active* ($churn = 0$) otherwise.

3. **Test observation window:** 2025-01-05 to 2025-01-27. Features are computed with the same procedure as in the training observation window.

4. **Test label window:** 2025-01-28 to 2025-01-31. Churn labels are assigned following the same rule as in the training label window.

This design mirrors a real-world prediction scenario: the model is trained on past behavior, summarized over an observation period, and evaluated on its ability to anticipate future churn in a non-overlapping label window.

4.2.1 Feature Visualization

To facilitate the interpretation of the features, we relied primarily on the `sns.boxplot` function from the `seaborn` library[5]. This visualization method plots the distribution of a feature conditioned on the churn label, allowing us to compare the behavior of churned versus active entities.

In each boxplot, the x-axis represents the churn label ($churn = 0$ for active, $churn = 1$ for churned), while the y-axis corresponds to the feature values. The blue box indicates the interquartile range (IQR), i.e., the middle 50% of the data. The line inside the box denotes the median, while the "whiskers" extend up to 1.5 times the IQR beyond the first (Q1) and third (Q3) quartiles. Values beyond this range are treated as outliers. For clarity of interpretation, these outliers are not plotted, ensuring that the focus remains on the bulk of the data distribution.

Rolling windows (last-day snapshot, 3-day and 7-day sums) Recent activity was summarized over short- and medium-term periods, under the assumption that churn is strongly linked to **recency of activity**. Entities that are inactive in the final days of the observation window are more likely to churn. Figure 3 illustrates the discriminative power of three recency-based features. The `views_last_-day` feature proved to be the sharpest indicator of churn: churned entities typically show zero or very low activity, while survivors exhibit substantially higher values. The 3-day rolling sum (`sum_3d`) remained informative, smoothing out accidental inactivity (e.g., dips during weekends) and thus complementing the last-day snapshot. In contrast, the 7-day rolling sum (`sum_7d`) showed considerable overlap between churners and survivors, as longer windows dilute short-term signals with older activity.

Overall, we found that the last-day and 3-day features provide complementary predictive value, while the 7-day feature contributed little additional signal and was excluded to reduce redundancy.

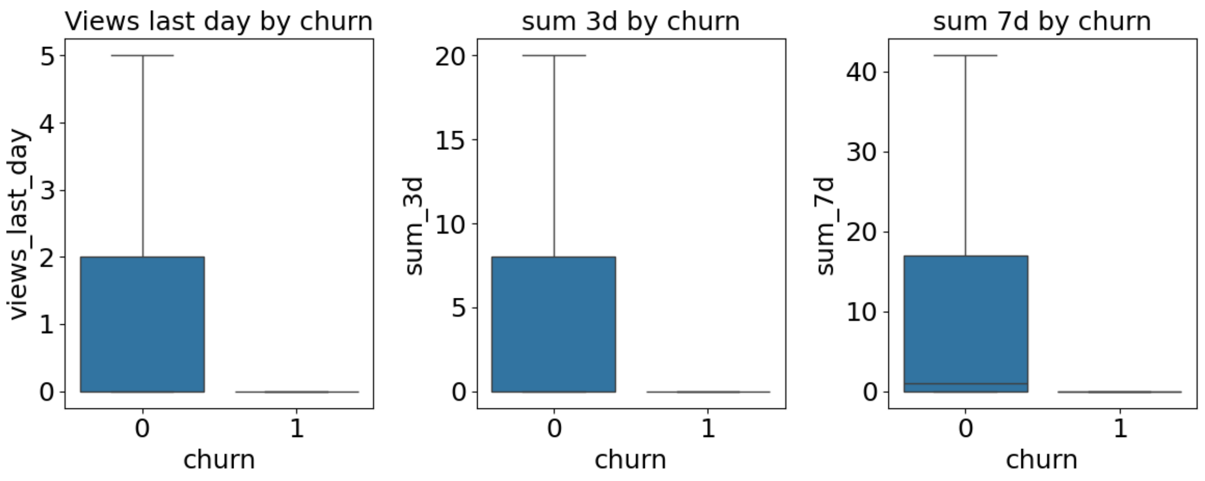

Figure 3: Distribution of short-term activity features by churn status. Each boxplot compares active entities (churn=0) and churned entities (churn=1) with respect to (i) the number of views on the last day of the observation window, (ii) the cumulative number of views over the last 3 days, and (iii) the cumulative number of views over the last 7 days. These features capture recency of activity, which is often a strong predictor of future survival.

Measures of intensity and variability of activity. In order to capture the behavioral dynamics of entities during the observation window, we extracted a set of features that reflect both the *intensity* and the *variability* of activity. These metrics allow the model to distinguish between consistently engaged entities and those with irregular or declining activity patterns. In particular:

- **Total views** (`views_total`): the sum of all pageviews over the observation window, measuring overall activity intensity.
- **Mean views** (`views_mean`): the average number of daily views, providing a baseline for typical activity.
- **Median views** (`views_median`): the median daily count, more insensitive to extreme outliers than the mean.
- **Maximum views** (`views_max`): the highest number of views recorded in a single day, capturing peak popularity.
- **Standard deviation** (`views_std`): the degree of fluctuation in daily activity; high values correspond to spiky, unstable patterns, while low values indicate stability.
- **Trend slope** (`trend_slope`): the slope of a linear regression fitted through daily activity over time, capturing whether engagement is increasing (positive slope) or declining (negative slope).

Feature evaluation. An exploratory analysis of these distributions reveals that not all metrics provide the same predictive utility:

1. **Trend slope (`trend_slope`):** Both churned and active entities exhibit distributions centered around zero. This suggests that whether activity is increasing or decreasing does not meaningfully distinguish between churners and non-churners. For this reason, the feature was discarded.
2. **Total views (`views_total`):** A clear separation is observed: active entities with `churn=0` display significantly higher totals compared to those with `churn=1`. This feature was therefore retained as a strong indicator of sustained engagement.
3. **Median vs. mean (`views_median` vs. `views_mean`):** While the two measures show similar distributions, the mean better captures the influence of outliers, which are common in Wikipedia traffic due to viral or high-profile pages. Since including both would be redundant, only the mean was retained.

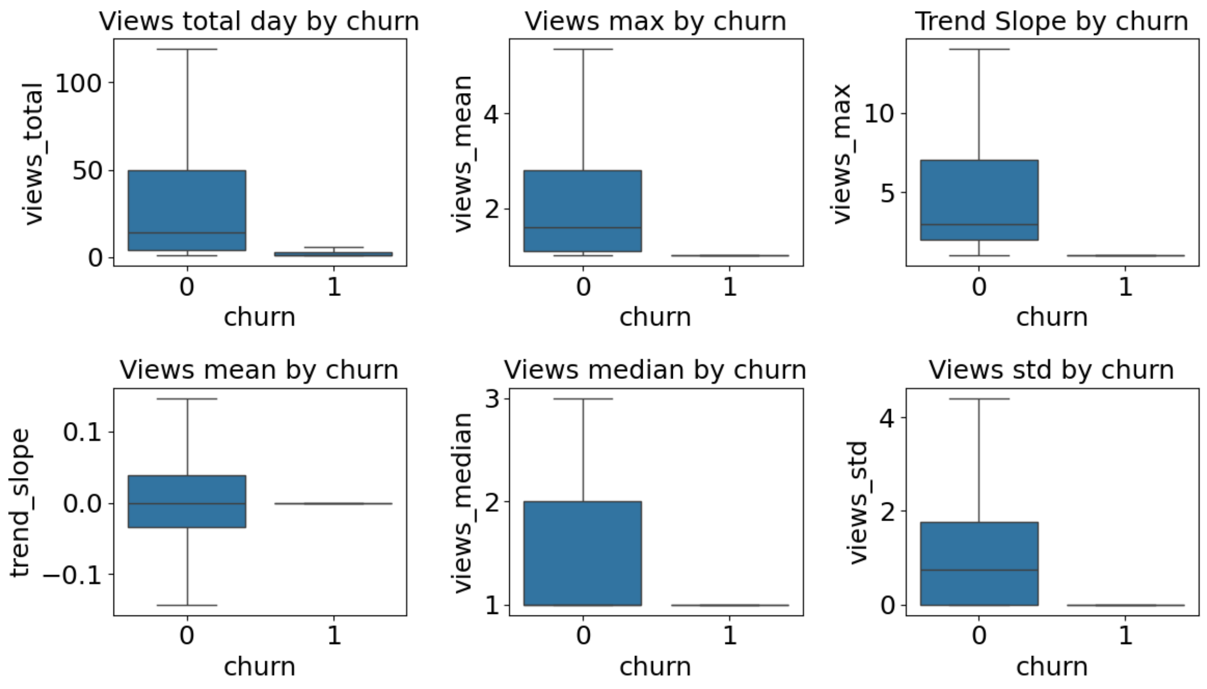

Figure 4: Boxplots of aggregated activity features during the observation window, stratified by churn outcome. Each subplot compares the distribution of a feature (total views, mean, median, maximum, trend slope, and standard deviation of daily views) between entities that churned (`churn=1`) and those that remained active (`churn=0`). The plots highlight differences in activity intensity and variability that may serve as predictive signals for churn modeling.

4. **Maximum vs. standard deviation (`views_max` vs. `views_std`):** Both features capture aspects of *spikiness*, albeit differently: the maximum highlights the single peak, while the standard deviation reflects overall volatility across the time frame. Given its more holistic nature and reduced sensitivity to isolated events, `views_std` was retained.

Seasonality-aware features With these features we aim to address the data quirk of *seasonality*, namely the presence of weekly cycles in pageview activity. Such cycles are natural in many contexts (e.g., weekday work vs. weekend leisure) and, if not properly modeled, could be mistakenly interpreted as early signs of churn. To mitigate this, we considered three seasonality-aware features:

- **Unique Weekdays** (`unique_weekdays`): the number of distinct weekdays with observed activity (1 = activity concentrated on a single weekday, 7 = activity distributed across all days).
- **First Dow** (`first_dow`): the weekday of the first recorded activity (categorical, 1 = Sunday, ..., 7 = Saturday).
- **Vies Std Dow** (`views_std_dow`): the standard deviation of daily activity across weekdays (higher values indicate uneven or spiky activity, lower values indicate uniform activity).

The empirical analysis, Fig. 5, shows that `unique_weekdays` provides strong discriminatory power. Non-churned entities (`churn = 0`) typically display activity across approximimately five weekdays, with values generally ranging between three and seven. In contrast, churned entities (`churn = 1`) concentrate their activity on only one to three weekdays, with the median close to 1.5. This suggests that entities exhibiting more diverse weekday activity patterns are less likely to churn, while narrow activity patterns are predictive of higher churn risk.

For `first_dow`, no meaningful difference is observed between churned and non-churned entities: the distributions overlap almost completely. This indicates that the weekday of first activity is essentially arbitrary and provides little predictive value. Accordingly, this feature can be safely discarded.

Finally,`views_std_dow` also emerges as a valuable feature. Non-churned entities show higher variability across weekdays, with values ranging up to 4 and typically falling between 0 and 1.8. By contrast,

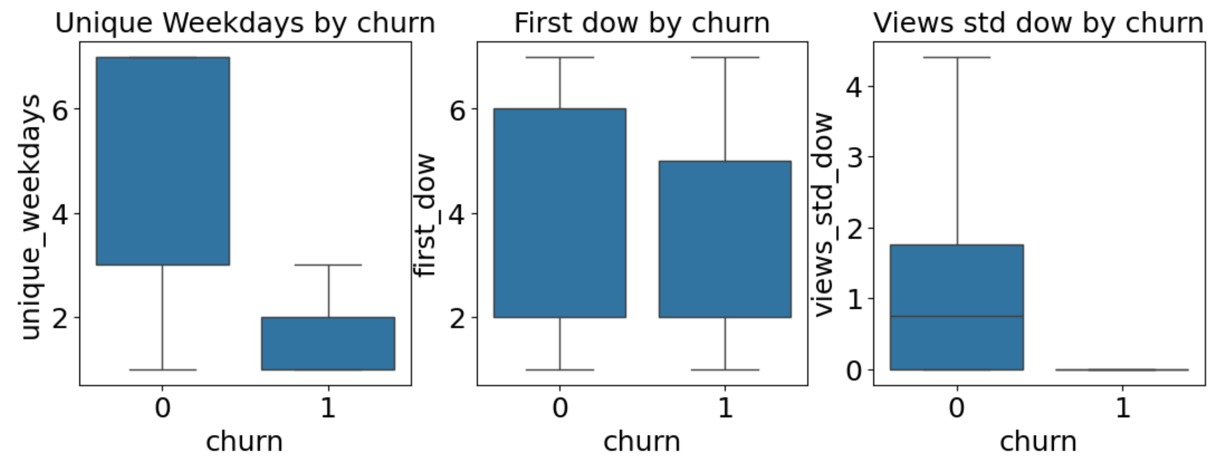

Figure 5: Distribution of temporal activity features across churned and non-churned entities. The first panel compares the number of unique weekdays with activity, the second shows the weekday of the first activity, and the third illustrates the variability of activity across weekdays (standard deviation). The boxplots highlight systematic differences between entities that survive and those that churn, reflecting the role of seasonality and temporal regularity as predictive signals.

churned entities display much lower variability, with values clustered near zero, indicating activity concentrated on only one or two weekdays. This pattern implies that uneven or irregular activity across weekdays is correlated with survival, while consistently low variation is associated with fragility and higher churn risk. Thus, `views_std_dow` is a relevant predictor to retain.

We observe as *views_total*, *views_mean* these features capture intensity and stability of activity over the entire observation window, but they are agnostic to the Seasonality calendar structure (they don't know which days matter). *unique_weekdays* and *views_std_dow* explicitly capture temporal regularity and seasonality patterns.

Sparsity guards Sparsity in activity serves as a natural proxy for *engagement strength*. Entities that exhibit only a few active days within the observation window display weak and irregular engagement, which translates into a higher likelihood of disappearing in the label window (i.e., churning). In contrast, entities that are active on many days demonstrate consistent and sustained usage, and are therefore more likely to remain present in the future.

By encoding this signal as a categorical feature (`sparsity_level`), the dataset captures a *graded spectrum of engagement* rather than a simplistic binary indicator of activity versus inactivity. This discretization not only enhances the interpretability of the model but also highlights the predictive value of regularity in activity patterns.

As shown in Fig. 6, the distribution of churn rates across sparsity categories exhibits a clear monotonic trend: entities with very sparse activity (two or fewer active days) almost always churn, whereas those with frequent activity display substantially lower churn probabilities. This inverse relationship not only provides strong predictive power but also yields intuitive interpretability, justifying the inclusion of the `sparsity_level` feature as a key indicator of engagement dynamics.

Daily Aggregation To evaluate the impact of temporal resolution, we compared raw hourly pageviews with their daily aggregates. While hourly data exhibits high variance due to short-lived bursts of activity (e.g., news events, bots, time-zone effects), aggregating to the daily level reveals smoother trends that are more suitable for feature construction.

Figure 7 illustrates how daily pageviews reduce hourly variance, thereby exposing stable signals such as recency, weekly seasonality, and long-term decay. This justifies the use of daily aggregation for retention analysis and churn prediction.

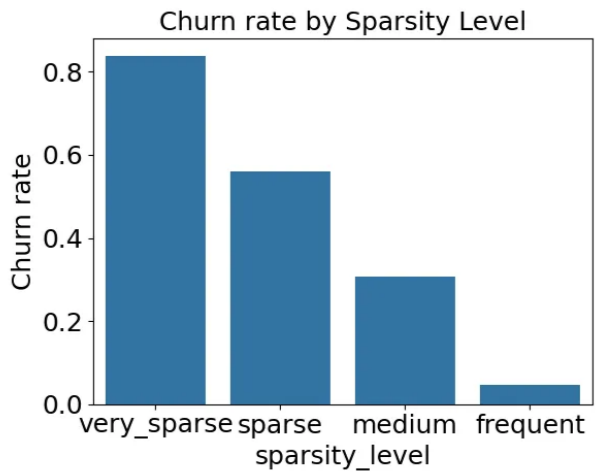

Figure 6: Churn rate by sparsity level. The bar plot shows the average churn probability across entities grouped by their activity sparsity, as derived from the observation window in the `build_ml_dataset` function. Sparsity levels are defined based on the number of active days (very sparse, sparse, medium, frequent). The figure highlights the strong inverse relationship between activity regularity and the likelihood of churn in the subsequent label window.

5 Churn Prediction Assignment

The churn prediction task is formulated as a binary classification problem, where the goal is to predict whether a Wikipedia page will *churn* (1) — i.e., disappear from the label window — or remain *active* (0).

This section is organized into three parts. First, we describe the data preprocessing and feature engineering steps, including the encoding pipeline applied to categorical and numerical variables. Second, we motivate the choice of the predictive model, discussing why it is well-suited to the churn prediction problem. Finally, we present the evaluation of the trained model, report its performance metrics, and draw conclusions from the results.

The library used for training is scikit-learn [6].

5.1 Data Preprocessing

The input features were divided into categorical and numerical groups, each preprocessed according to its nature.

Categorical features: `page_title`, `language`, `database_name`, `namespace`, `is_mobile`, and `sparsity_level`. These variables were encoded using a `OneHotEncoder`. This approach is appropriate because categorical features are discrete and non-ordinal, and binary encodings preserve their informational content without introducing artificial orderings. Tree-based models in particular benefit from one-hot representations, which allow them to learn splits independently for each category.

Numerical features: `days_active`, `views_total`, `views_mean`, `views_std`, `unique_weekdays`, `views_std_dow`, `views_last_day`, and `sum_3d`. These were standardized using a `StandardScaler`. Although gradient boosting methods are generally robust to unscaled features, normalization ensures that variables contribute on a comparable scale, prevents dominance of high-magnitude features, and improves the stability of the preprocessing pipeline.

Pipeline: A `ColumnTransformer` applied the appropriate preprocessing (`OneHotEncoder` for categorical and `StandardScaler` for numerical features). This step was embedded into a unified `Pipeline` together with the classifier.

5.2 Model choice

The predictive model selected for this study was the `HistGradientBoostingClassifier`[7], scikit-learn's efficient implementation of gradient-boosted decision trees. This choice reflects both the characteristics of the churn prediction problem and the computational constraints of the Databricks working environment.

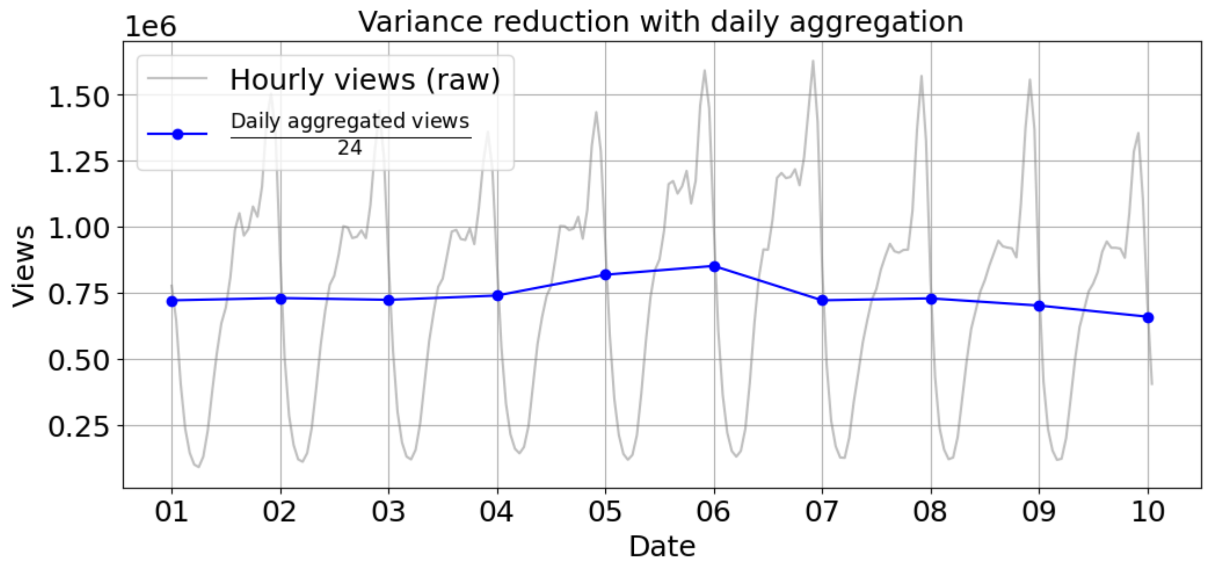

Figure 7: Comparison of raw hourly pageviews (gray) with daily aggregated values (blue). The hourly series exhibits strong variance due to short-lived fluctuations, whereas daily aggregation smooths these irregularities and exposes stable behavioral trends. Dividing daily totals by 24 provides a directly comparable scale to the hourly series, highlighting how aggregation reduces noise while retaining meaningful temporal patterns.

Why gradient boosting is suitable for churn prediction Predicting churn is inherently complex due to the following reasons:

- **Nonlinear interactions:** As we saw in sec. 4.2, churn rarely depends on a single feature but instead arises from combinations (e.g., "low views + low recency + sparse activity"). Gradient boosting captures such nonlinear interactions naturally.
- **Mixed feature types:** The dataset contains both categorical variables (`language`, `database_name`, `namespace`) and numerical variables (`days_active`, `views_total`, `sum_3d`). After one-hot encoding, boosted trees can effectively leverage both.
- **High-cardinality features:** Certain categorical variables, such as `page_title`, expand into thousands of one-hot dimensions. Tree-based models can handle such sparse binary encodings more gracefully than linear models.

Among boosting implementations, `HistGradientBoosting` was selected because it is optimized for efficiency and memory use through histogram-based binning of continuous features. This property was particularly important given the constraints of Databricks Community Edition (single-node cluster with 15 GB RAM and 2 vCPUs).

Comparison with alternatives

1. **Logistic Regression**[8] scales well but cannot capture nonlinear churn drivers.
2. **Random Forests**[9] handle nonlinearities but are more expensive to train and less efficient with high-dimensional one-hot encoded features.
3. **XGBoost / LightGBM**[10] are strong alternatives, but `HistGradientBoosting` was preferred for its native `scikit-learn` integration and lower resource footprint.

Hyperparameter tuning Hyperparameters were optimized using `RandomizedSearchCV` with 3-fold cross-validation and ROC-AUC as the evaluation metric.

$$\text{CV-ROC-AUC} = \frac{1}{k}\sum_{i=1}^{k} \text{ROC-AUC}_i, \qquad \text{with} \quad \text{ROC-AUC} = \int_0^1 \text{TPR}(\text{FPR})\, d(\text{FPR}). \tag{1}$$

The search space, represented in table (3), included tree depth (`max_depth`), learning rate (`learning_rate`), number of iterations (`max_iter`), minimum samples per leaf (`min_samples_leaf`), and L2 regularization strength (`l2_regularization`). A `FunctionTransformer` ensured dense output from the encoder, which is required by the classifier.

Hyperparameter	Search Space	Explanation
`max_depth`	[3, 5, 7, None]	Controls tree complexity and interaction depth.
`learning_rate`	[0.01, 0.05, 0.1, 0.2]	Lower values reduce overfitting, requiring more trees.
`max_iter`	[100, 200, 500]	Number of boosting rounds; higher values for stability.
`min_samples_leaf`	[10, 20, 50]	Regularizes trees by avoiding very small leaves.
`l2_regularization`	[0.0, 1.0, 5.0]	Adds penalty to improve generalization.

Table 3: Hyperparameter search space for `HistGradientBoostingClassifier`.

Randomized search was chosen over full grid search because it explores a wide parameter space efficiently without exhaustively testing all combinations, thus reducing runtime and memory usage. Three folds were sufficient to balance performance estimation with computational cost.

After tuning, the optimal configuration (Tab. 4) achieved a cross-validated ROC-AUC of approximately 0.883.

Best Hyperparameters
`max_depth = 7`
`learning_rate = 0.05`
`max_iter = 500`
`min_samples_leaf = 10`
`l2_regularization = 5.0`

Table 4: Best configuration of the `HistGradientBoostingClassifier`.

These settings achieve a balance between model complexity and regularization, which is crucial for churn prediction problems where signals are often noisy and sparse.

5.3 Evaluation

The best model was evaluated on two disjoint test sets to assess both *in-sample generalization* and *temporal robustness*. The first test set was obtained by splitting the initial dataset into training and test ranges (`obs_start = 2025-01-01`, `obs_end = 2025-01-23`, `lbl_start = 2025-01-24`, `lbl_end = 2025-01-27`). The second test set was shifted to a later interval not used during training (`obs_start = 2025-01-05`, `obs_end = 2025-01-27`, `lbl_start = 2025-01-28`, `lbl_end = 2025-01-31`). This design allowed us to evaluate both conventional hold-out performance and the model's ability to generalize to future time periods, which is critical in churn prediction.

Evaluation metrics. For the evaluation we use *threshold-independent* metrics as :

- **ROC-AUC**: area under the Receiver Operating Characteristic curve, measuring global separability between churned and active entities.
- **PR-AUC**: area under the Precision–Recall curve, it focuses on the positive (churned) class.

We also report *threshold-dependent* metrics for interpretability:

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}, \quad F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2}$$

where TP, FP, and FN denote true positives, false positives, and false negatives. Precision quantifies the reliability of churn predictions (few false alarms), recall measures the ability to detect churners (few missed positives), and F1 balances the two.

Results on Test Set 1 (random split). On a hold-out sample of 1000 instances, the model achieved:

- ROC-AUC = 0.876 (strong separability between classes),
- PR-AUC = 0.738 (substantially above the random baseline, equal to churn prevalence).

Class	Precision	Recall	F1-score	Support
0 (Active)	0.85	0.76	0.80	504
1 (Churn)	0.78	0.86	0.82	496
Accuracy	//	//	0.81	1000

Table 5: Evaluation results on Test Set 1 (random split).

The model shows a balanced trade-off between precision and recall, with recall slightly higher for churn, indicating that most churners are successfully identified.

Results on Test Set 2 (temporal shift). When tested on a shifted temporal window, the model maintained robust performance:

- ROC-AUC = 0.826,
- PR-AUC = 0.703.

Class	Precision	Recall	F1-score	Support
0 (Active)	0.82	0.73	0.77	504
1 (Churn)	0.75	0.83	0.79	496
Accuracy	//	//	0.79	1000

Table 6: Evaluation results on Test Set 2 (temporal shift). Support counts omitted for clarity.

Although performance declined slightly compared to the random split, the model still retained high separability and strong recall, demonstrating robustness to temporal drift. Overall, the evaluation confirms that the model performs reliably both the test sets. This indicates that the learned patterns generalize well over time, an essential requirement in churn prediction where user behavior naturally evolves.

5.3.1 Permutation Importance Analysis

To assess the relative contribution of the features defined in Section 4.2 to the churn prediction task, we applied *permutation importance*[11] on the second test set. This model-agnostic method evaluates feature relevance by measuring the decrease in predictive performance (here, ROC-AUC) when the values of a given feature are randomly shuffled, thereby breaking its relationship with the target variable. Features that cause larger performance drops are considered more influential.

The top eight features ranked by permutation importance are presente in Tab. 7.

Feature	Importance
`days_active`	0.141
`views_total`	0.037
`sum_3d`	0.024
`sparsity_level`	0.008
`namespace`	0.004
`unique_weekdays`	0.003
`views_std`	0.002
`database_name`	0.001

Table 7: Top features ranked by permutation importance (measured as mean ROC-AUC decrease).

These results confirm several of our earlier design hypotheses. First, `days_active` emerges as the most important driver of churn, highlighting that engagement regularity across the observation window is the single strongest predictor of survival. This is consistent with the findings in Fig. 6, where higher activity regularity correlated with lower churn rates.

Second, activity intensity features such as `views_total` and short-term recency signals like `sum_3d` also rank highly, validating the design choice to emphasize both long-term intensity and short-term engagement in our feature set.

Third, seasonality-aware features such as `unique_weekdays` and measures of volatility like `views_std` provide weaker but non-negligible signals. This indicates that while seasonal regularity helps explain some differences between churners and non-churners (see Fig. 5), it is secondary to overall activity and recency.

Finally, metadata-driven attributes such as `namespace` and `database_name` contribute marginally, suggesting that structural context matters far less than behavioral patterns.

In summary, the permutation analysis corroborates the intuition from Section 4.2: churn prediction in this setting is primarily governed by **engagement intensity and recency**, complemented by weaker signals from **seasonality** and **metadata**.

References

[1] Giovanni Minuto. *wikimedia_yard_reaas_test: End-to-end Spark + ML pipeline for Wikimedia Pageviews.* Available at: https://github.com/giovanniminuto/wikimedia_yard_reaas_test.

[2] Wikimedia Foundation. *Wikimedia Pageviews — January 2025.* Available at: https://dumps.wikimedia.org/other/pageviews/2025/2025-01/.

[3] Databricks / O'Reilly. *Delta Lake: The Definitive Guide.* eBook available at Databricks link

[4] Wikimedia Foundation. *Pageview Data Documentation.* Available at: wikitech.wikimedia.org – Pageviews schema.

[5] Seaborn documentation. *sns.boxplot.* Available at: seaborn.pydata.org – boxplot.

[6] Scikit-learn. *scikit-learn: Machine Learning in Python.* Available at: scikit-learn.org.

[7] Scikit-learn. *HistGradientBoostingClassifier.* Available at: scikit-learn.org – HistGradientBoostingClassifier documentation.

[8] Scikit-learn. *LogisticRegression.* Available at: scikit-learn.org – LogisticRegression documentation.

[9] Scikit-learn. *RandomForestClassifier.* Available at: scikit-learn.org – RandomForestClassifier documentation.

[10] Scikit-learn. *GradientBoostingClassifier.* Available at: scikit-learn.org – GradientBoostingClassifier documentation.

[11] Scikit-learn. *permutation_importance.* Available at: scikit-learn.org – Permutation Importance documentation.