Universidad Distrital Francisco José de Caldas

Systems analysis and desing GR020-85

Workshop 2
Competition: Web Traffic Time Series Forecasting

Members:

Johan Sebastián Beltrán Merchán code: 20222020019
Edison David Álvarez Varela code: 20222020043
Yader Ibraldo Quiroga Torres code: 20222020034
Julián David Celis Giraldo code: 20222020041

Lecturer:

Carlos Andrés Sierra Virgüez

Bogotá, Colombia
October 17, 2025

1 Review Workshop 1 Findings

1.1 Objective:

Prediction of visits to Wikipedia articles.

1.2 Article Data:

Daily time series of visits per article, language, platform, and agent type.

1.3 System Constraints:

During the development of the first workshop, the following critical system constraints were identified:

- **Massive data scale**: The dataset includes approximately 145,000 pages with more than 800 temporal points per series, which implies significant computational cost and the need for a scalable architecture capable of efficiently processing this volume of information.
- **Fixed temporal window**: The training data spans from July 2015 to September 2017 (550 days of temporal history), establishing a limited historical period that restricts the system's ability to capture long-term trends and adapt to emerging patterns outside this temporal range.
- **Zero-value ambiguity**: Many pages contain long periods with zero or empty values, the problem lies in the dataset's inability to distinguish whether this represents genuinely null traffic or missing data due to collection issues; both cases are represented in the same way, generating uncertainty in the modeling process.
- **Pattern heterogeneity**: Pages from different topics (music, technology, movies, education) show completely different traffic patterns, making it difficult to apply a single universal model for all data and requiring differentiated modeling strategies.
- **Noise and automated traffic**: The presence of bots, spiders, and web crawlers generates outliers that distort real user behavior, complicating the prediction process and negatively affecting result quality.

1.4 Critical Data Characteristics:

- **Sparsity**: Many pages have sporadic traffic, with long periods of zero or minimal views. This characteristic represents a significant challenge as traditional time series models struggle with sparse and intermittent data.
- **Embedded metadata structure**: The article name encodes critical metadata (language, Wikipedia project, access type desktop/mobile, and traffic agent) that is vital for modeling and must be parsed and systematically leveraged.
- **Consistent temporal sampling**: Regular daily samples provide temporal consistency, facilitating time series analysis by maintaining uniform intervals without gaps in the temporal grid.

1.5 Key Relationships

- Article -> date -> Visits (train$_*$.*csv*)
- Language -> Cultural / temporal patterns
- Platform (desktop/mobile) -> User behavior

1.6 Chaos Factors

- Unpredictable viral events
- Changes in search algorithm
- Breaking news that affects visits
- Complex seasonalities (day, week, month, holidays)

2 Define System Requirements:

Functional Requirement: Defined as what the system must do, functionalities, behaviors, and services, focused on system actions or processes.

Article Data: Daily time series of visits per article, language, platform, and agent type.
Requirements include:
FR1: Data management

- **FR1.1:** Process large datasets efficiently
- **FR1.2:** Predict daily visits for specific article-date combinations
- **FR1.3:** Handle multiple cultural, social, and viral trend dimensions to reduce bias.
- **FR 1.4:** Produce submission file in specific CSV format
- **FR 1.5**: Analyze complex structured identifiers
- **FR 1.6**: Manage time series data across different date ranges (2015-2017)

FR2: Prediction engine:

- **FR 2.1:** Generate daily visit predictions for specific article-date combinations
- **FR 2.2:** Handle multiple access types (all-access, desktop, mobile-web)
- **FR 2.3:** Process different agent types (all-agents, spider)
- **FR 2.4:** Produce a submission file according to Kaggle platform requirements

FR3: Feature Engineering:

- **F.R 3.1:** Extract temporal features (day of week, month, year, holidays)
- **F.R 3.2:** Calculate continuous statistics (7-day, 30-day moving averages)
- **F.R 3.3** Generate lag features (previous day, week, months visits)
- **F.R 3.4** Creation of metadata specific to article characteristics
- **F.R 3.5** Handling of temporal patterns in the language spatial domain

FR4: Validation & testing :

- **FR 4.1:** Implement temporal validation strategies
- **F.R 4.2:** Validate prediction format against submission samples
- **F.R 4.3:** Provide model performance metrics (SMAPE, MAE, RMSE)
- **F.R 4.4** Generate error reports and analysis

Non-functional requirements: Describe how the system should behave, regarding quality characteristics and technical or performance constraints.

Performance requirements: NFR1: Scalability for large dataset ingestion

- **NFR1.2**: Generate predictions for all key entries in under 2 hours.
- **NFR 1.3:** Support batch processing of a minimum of 100,000 records
- **NFR 1.4:** Achieve inference latency less than 1 second per 1000 predictions
- **NFR 1.5:** Maintain system responsiveness during processing peaks

Scalability requirements:

- **NFR 2.1:** Horizontal scaling for handling datasets larger than 10 GB
- **NFR 2.2:** Model with support for concurrent training and inference
- **NFR 2.3:** Efficient memory management for matrices with too many features
- **NFR 2.4:** Modular architecture that allows replacement or upgrades

Reliability Requirements:

- **NFR 3.1:** Data processing rate of 99%
- **NFR 3.2:** Automatic recovery from transient failures
- **NFR 3.3:** Data integrity validation at each processing stage

Accuracy Requirements:

- **NFR 4.1:** Achieve a competition score of SMAPE <35 (baseline)
- **NFR 4.2:** Maintain consistency in prediction across different time periods
- **NFR4.3 :** Handle edge cases (zero visits, empty spaces, viral peaks, data loss)
- **NFR 4.4:** Provide confidence intervals for predictions.

3 High-Level Architecture:

Each module of the data flow is responsible for a specific process.

1. **Data Ingestion:** The first step involves receiving the data, establishing the file format, and performing batch loading, which allows for handling large volumes of information. This stage works with immense datasets and ensures they are available for subsequent processing.
2. **Preprocessing:** Once the raw data is received, key variables are identified, and tasks such as cleaning, unification, segmentation, and handling of values that may generate bias are performed. This stage ensures the data is consistent, reliable, and ready for the next module.
3. **Feature Engineering:** In this module, data is transformed and adjusted to maximize its utility for the model. Variables that generate noise are removed, relevant features are created or modified, and data is prepared so that the model can interpret it correctly, improving the precision and effectiveness of the predictions.
4. **Model Training:** Here, the model is trained, in this case using regression algorithms, to learn from the historical data. The model adjusts its internal parameters and learns the relationship between the input variables and the expected outcomes.

5. **Prediction Engine:** This module uses the trained model to generate predictions on new data, applying the transformations and features defined in the previous stages.

6. **Validation:** It ensures that data ingestion, preprocessing, and information format between modules comply with the established rules, guaranteeing the integrity and consistency of the data throughout the flow.

7. **Submission:** This module is responsible for sending the results to the competition via the Kaggle API. The platform assigns a score to the submission, which is analyzed in the Evaluation module.

8. **Evaluation:** The prediction results are evaluated and adjusted to the format required by the Kaggle platform. This module also analyzes the model's performance and generates feedback.

9. **Feedback and Iterative Cycle:** Based on the evaluation results, feedback can be sent to **Feature Engineering**, to analyze and adjust variables that generate noise, or to **Model Training**, to optimize the model's weights, parameters, or hyperparameters. This process is repeated iteratively until the predictions reach an acceptable margin of error according to Kaggle criteria.

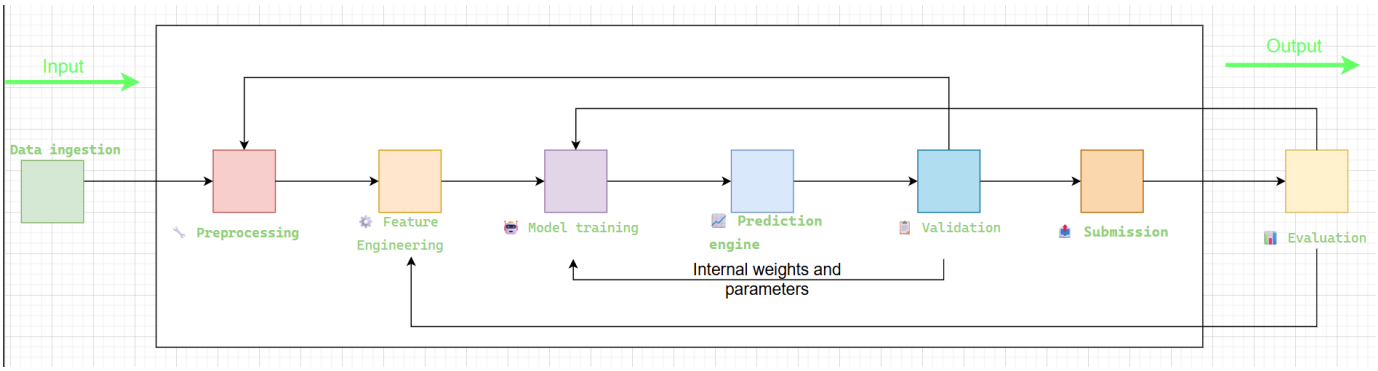

Figure 1: Conceptual Architecture: Inputs, Core Processes, and Quality Loop

The high-level architecture has been defined; the sub-modules and responsibilities of each module are described below.

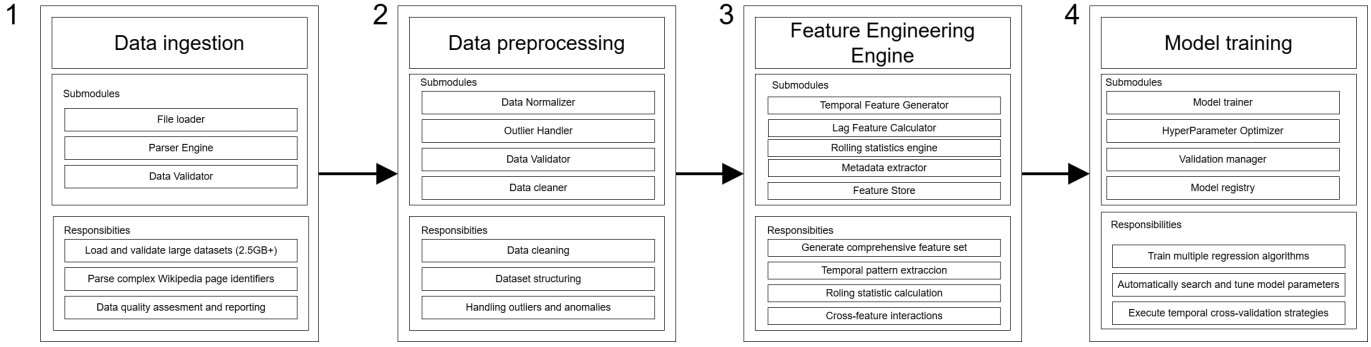

Figure 2: Modular Architecture: Data Acquisition and Model Training Subsystem Breakdown

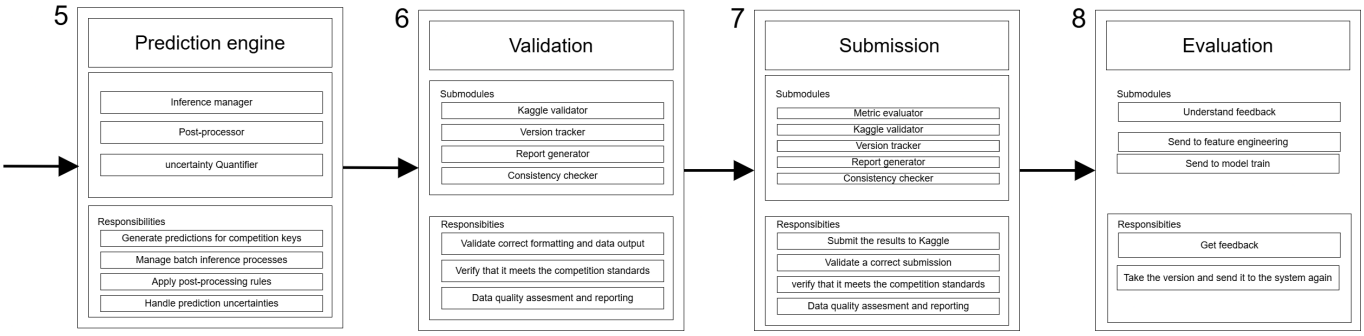

Figure 3: Modular Architecture: Inference, Validation, and Evaluation Subsystem Breakdown

The systems engineering principles applied to the system design are as follows:

1. **Systemic (Holistic) Approach:** Viewing the system as an interconnected whole, not a set of isolated scripts. Each module interacts as a subsystem that transforms inputs into outputs, making the whole greater than the sum of its parts.

2. **Modularity and Functional Decomposition:** Division of a complex system into connected and mutually influencing modules. Each module is well-defined, with clear interrelationships and responsibilities. Any change in one subsystem reconfigures the behavior of the whole.

3. **Feedback:** Self-regulating systems use feedback for correction and adaptation, allowing for adaptive learning, equivalent to cybernetic homeostasis.

4. **Open System:** Systems exchange energy, information, or matter with their environment. The system receives external data, transforms it, and returns processed information to the external environment. Additionally, it receives feedback from the environment (SMAPE), which is used for internal improvement, managing information exchange and system permeability.

5. **Homeostasis:** The validation and evaluation modules act as homeostatic mechanisms, correcting deviations (poor performance) through automatic adjustments, seeking equilibrium between precision and generalization.

6. **Equifinality:** The open system can achieve the same goal from different initial conditions, through various models or strategies.

7. **Adaptation and Evolution:** The Evaluation module detects poor performance, and algorithms or features are modified to better adapt to Kaggle's competitive environment. The system learns, reconfigures, and evolves, imitating processes of biological self-regulation.

8. **Entropy and Negentropy:** Every system tends toward disorder (entropy), but open systems can reverse it through the input of information (negentropy). The preprocessing and validation modules reduce this entropy, maintaining the life of the prediction system.

9. **Goal Orientation:** The system is oriented towards a purpose or objective: minimizing the prediction error. The entire flow, decisions, and feedback loops are geared toward meeting this goal.

4 Addressing Sensitivity and Chaos

4.1 Strategies for Highly Sensitive Variables

4.1.1 Handling Viral Spikes and Unpredictable Events

The system faces the challenge of viral events that generate massive, unpredictable traffic spikes. An article can go from 100 to 50,000 daily visits due to breaking news or viral content, exemplifying the "butterfly effect" identified in Workshop 1.

Multi-level Anomaly Detection: The system implements adaptive statistical filters based on rolling standard deviation. It calculates dynamic thresholds for each article using 30-day windows: if traffic exceeds three standard deviations from the recent average, it is flagged as an anomaly. This three-sigma threshold captures 99.7% of normal behavior, making values outside this range statistically suspect. Segmentation by traffic type (desktop, mobile, all-access) and agent type allows for contextualizing anomalies. A mobile spike in sports articles over a weekend may be normal, while the same pattern in academic content would be anomalous.

Robustness Strategy: When a spike is detected, the system applies logarithmic transformation to smooth out extremes without losing information, generates "days since last spike" features to capture post-viral decay, and activates specialized models trained specifically for viral behavior.

4.1.2 Management of Bot and Spider Traffic

The dataset already provides separation between "all-agents" (humans + bots) and "spider" (bots only) through page identifiers. It is not possible to filter bots from the all-agents record because the data comes pre-aggregated.

Implementable Approach: The system extracts the agent type from the page name and uses it as an input feature. The models automatically learn that spider traffic shows more regular and predictable patterns, while human traffic is more volatile and reactive to external events. This differentiation allows for specialized predictions according to the record type.

4.1.3 Sensitivity to Missing Data and Zeros

The dataset does not distinguish between true zeros and missing data. The system analyzes temporal context: an isolated zero in an article with a consistent history is interpreted as missing data and imputed via interpolation. Multiple consecutive zeros in low-traffic articles are preserved as legitimate signals. Quality features are generated, such as the percentage of missing data in the last 30 days and the length of zero streaks. These metrics allow the model to calibrate its confidence: articles with 40% missing data receive more conservative predictions.

4.2 Handling Chaotic Factors

4.2.1 Continuous Adaptation and Drift Monitoring

The system implements retraining with a sliding window: it discards old data and incorporates recent data to adapt to emerging trends. This is critical because web traffic is not stationary: user preferences, search algorithms, and world events constantly change. Statistical tests (Kolmogorov-Smirnov) are applied to detect changes in traffic distribution between periods. If the pattern of the last two weeks differs significantly from the previous month, the system activates alerts and adjusts models specifically for the affected subgroups (by language, access type, etc.).

4.2.2 Complex Seasonalities and Non-Linear Patterns

Temporal Feature Engineering: Temporal variables are transformed using sine and cosine functions to capture natural cyclicity. The day of the week is represented as a continuous circle where Monday and Sunday are adjacent, not numerically distant. Holiday calendars segmented by article language (Thanksgiving for English articles, Latin American festivities for Spanish) are incorporated. Compound features are generated that capture interactions: "weekend during summer" is different from "weekend during winter," especially for educational content.

Hierarchical Ensemble Architecture: The system groups articles with similar behavior using time series clustering, recognizing that sports, educational, and entertainment articles require different approaches.

1. Level 1: Base models (ARIMA for stationary series, Prophet for robust seasonalities, LSTM for non-linear dependencies) generate independent predictions.

2. Level 2: A Meta-model analyzes article features and assigns weights to each Level 1 prediction. For stable articles, it favors ARIMA; for complex seasonalities, it favors Prophet; for non-linear patterns, it favors LSTM.

4.2.3 Handling Low-Traffic Series

For sporadic articles where noise dominates the signal, the system applies adaptive exponential smoothing with a reduced alpha, heavily averaging new observations with historical data to avoid overreacting to random fluctuations. Prediction intervals (10th, 50th, 90th percentiles) are generated using the median as the final prediction because it is more robust to outliers. L1/L2 regularization prevents overfitting by forcing the model to focus on consistent patterns and ignore occasional noise.

4.3 Monitoring and Error Handling Routines

4.3.1 Validation Checkpoints

The system implements validations at each stage of the pipeline:

- **Data Ingestion:** Verifies column structure, complete date range (July 2015 - September 2017), absence of duplicates.
- **Preprocessing:** Confirms that transformations do not generate invalid values (NaN, negatives).
- **Feature Engineering:** Validates absence of temporal data leakage (no feature uses future information).
- **Model Training:** Monitors loss convergence and the gap between train/validation SMAPE ($> 20\%$ indicates overfitting).
- **Prediction:** Verifies non-negative values and exact submission format (145K rows $\times$ 60 columns).

4.3.2 Cascading Fallback System

If the main model fails, the system automatically resorts to backup methods:

- Median of the last 7 days (simple but effective for stable traffic)
- Average of the same weekday in the last 4 weeks (captures weekly effects)
- Last observed value (naive forecast as a minimum guarantee)

Structured logs record all decisions in edge cases: article, detected anomalous condition, action taken, and confidence flag. This facilitates post-mortem analysis of poor-performing submissions.

4.3.3 Stratified Post-Prediction Analysis

The SMAPE is decomposed across multiple dimensions:

- Traffic level (low < 100, medium $100 - 1K$, high $> 1K$ visits/day)
- Language (en, es, de, fr, ja, zh, etc.)
- Access type (desktop, mobile, all-access)

This granular analysis identifies specific problematic subgroups, directing improvement efforts. If Japanese mobile articles have SMAPE $> 50\%$, specific hypotheses are investigated (uncaptured local events, poorly encoded holidays), and specialized features or models are adjusted for that segment.

4.3.4 Applied Design Principles

Finding (Workshop 1)	Design Strategy (Workshop 2)
Butterfly effect in viral traffic	Anomaly detection with adaptive thresholds + specialized models for spikes
Strange attractors (non-periodic patterns)	Temporal feature engineering with sine/cosine + time series clustering + hierarchical ensemble
Bot and spider noise	Use of agent type as a feature for automatic pattern differentiation
Ambiguous missing data	Conservative imputation based on context + data quality features
Multiple temporal scales	Ensemble with ARIMA/Prophet/LSTM + meta-model that weights according to article characteristics
Pattern heterogeneity	Clustering of similar articles + specialized models per group
Sporadic/random traffic	Adaptive smoothing + quantile prediction + aggressive regularization

5 Technical Stack and Implementation Sketch

5.1 Recommended Technical Stack and Justification

The selection of the technical stack is based on the functional requirements (FR) of feature generation and algorithm diversity, as well as the non-functional requirements (NFR) of scalability and performance for handling 145,000 parallel time series.

5.1.1 Core Language: Python

- **Choice: Python** (version 3.10+).
- **Justification:** Python is the standard for Data Science and Machine Learning due to its mature ecosystem, which directly aligns with the complex time series requirements of this competition.

5.1.2 Key Frameworks and Libraries

The selected tools address the specific challenges identified (e.g., sparsity, chaos, non-linear patterns and computational cost).

- **Data Manipulation and Preprocessing (FR1.1, FR3.1-3.5):**
  - **Pandas**: Essential for data loading, parsing complex page identifiers and feature engineering (lags, rolling statistics, temporal features).
  - **Apache Spark (for scaling NFR2.1):** Necessary for horizontal scaling and efficient handling of the dataset when it exceeds 10 GB (145K series $\times$ 800+ days), meeting the requirement to process large datasets efficiently.
- **Modeling and Prediction (FR2.1-2.3):**
  - **Scikit-learn/Statsmodels**: Used for classical regression (Level 2 Meta-model), evaluation metrics (SMAPE, MAE, RMSE) and classic time series models like **ARIMA**.
  - **PyTorch:** Essential for implementing **LSTM** (Long Short-Term Memory) models to capture non-linear and complex long-term dependencies.
- **MLOps and Monitoring (NFR3.3, FR4.4):**
  - **ydata-profiling**: Used for initial Exploratory Data Analysis (EDA) and data quality assessment.
  - **Joblib**: Used for parallelizing the training and inference processes across the 145K series.

5.2 Implementation Sketch and Design Patterns

The architecture will follow a **Modular Monolith** approach, where the high-level pipeline stages (Ingestion, Preprocessing, Feature Engineering, Training, Prediction, Evaluation) are clearly separated and managed as distinct services.

5.2.1 Data Flow Integration: Chain of Responsibility Pattern

- **Concept:** The entire pipeline (Figure 1) is implemented as a **Chain of Responsibility**.
- **Application:** Each core module (Ingestion, Preprocessing, Feature Engineering, Training) acts as a handler, the output of one module automatically becomes the input of the next, ensuring data integrity and consistency throughout the flow, this guarantees that only clean, validated data progresses sequentially.

5.2.2 Model Management: Strategy Pattern and Hierarchical Ensemble

- **Concept:** The **Strategy Pattern** is used within the **Prediction Engine** (Module 5) and the **Model Training** module (Module 4) to manage the heterogeneity of traffic patterns (e.g., sports, educational, and entertainment articles).
- **Application (Hierarchical Ensemble):**
  1. **Base Models (Strategies - Level 1):** ARIMA, Prophet, and LSTM are encapsulated as separate prediction strategies.
  2. **Context (Meta-model - Level 2):** The **Meta-model** analyzes article features (language, access type, volatility) and selects or assigns weights to the most appropriate base model for that specific time series, effectively changing the prediction strategy based on context. For example, a stable article favors the ARIMA strategy, while one with complex seasonality favors Prophet.

5.2.3 Code Implementation Plan

1. **Ingestion/Preprocessing (Module 1, 2):** Use **Pandas** for data loading and feature extraction from the `Page` column, because is a must implementing necessary cleaning and data imputation strategies.
2. **Feature Engineering (Module 3):** Implement a unified `FeatureEngine` class that wraps the `Temporal Feature Generator` and the `Lag Feature Calculator`, this class must rigorously apply temporal validation checks to prevent data leakage.
3. **Training/Prediction (Module 4, 5):** Define abstract model interfaces, implement concrete classes for `ARIMAStrategy`, `ProphetStrategy`, and `LSTMStrategy`, using **Joblib** for parallel processing of the 145K time series, meeting scalability demands.
4. **Validation/Feedback (Module 6, 8):** Use **SMAPE** (Symmetric Mean Absolute Percentage Error) as the primary evaluation function.