Workshop #2:
Kaggle Systems Design
Loan Default Prediction

Team #9

Juan Esteban Avila Trujillo - 20251020054
Juan Jose León Gomez - 20212020055
Juan Pablo Diaz Ricaurte - 20222020076
Miguel Angel Hernandez Medina - 20222020035

October 18, 2025

Table of Contents

Contents

Table of Contents **2**

1 Introduction **3**

2 Workshop No. 1 Results **4**

3 System Requirements Definition **4**
3.1 Functional Requirements 4
3.2 Non-Functional Requirements 4

4 High-Level Architecture **5**
4.1 Module Responsibilities 5
4.2 How Systems Engineering Principles Shaped Decisions 6

5 Addressing Sensitivity and Chaos **6**

6 Technical Stack and Implementation Sketch **7**
6.1 Programming Languages and Frameworks 7
6.1.1 Libraries: 7
6.2 System Modules 7
6.3 Design Principles 8
6.4 Implementation Plan 8

1. Introduction

This document presents the outcomes of **Workshop No. 1: Systems Engineering Analysis for Loan Default Prediction**, establishing the analytical and architectural foundation for designing a robust and scalable system.

The initial analysis of the Kaggle competition revealed that the default prediction problem operates as a **dynamic system sensitive to initial conditions**, strongly influenced by chaos and complexity theory. The quality and integrity of the input data, along with economic feedback effects, were identified as the main sources of instability and variability in the predictive model's performance.

Based on this diagnosis, the work is structured around three fundamental pillars of Systems Engineering:

1. **Requirements Definition:** Establishing the functional and non-functional requirements (such as scalability, reliability, and transparency) needed to address the inherent complexity of the dataset and the non-linear nature of credit risk.
2. **High-Level Architecture:** Proposing a modular and traceable architecture, applying principles such as **Modularity**, **Reproducibility**, and **Robustness** to mitigate system sensitivity and contain chaotic effects.
3. **Chaos and Sensitivity Mitigation:** Detailing design strategies, such as preprocessing pipeline control, feature stability analysis, and the use of **Model Ensembles**, to ensure consistent predictive performance in volatile environments.

Finally, the technical stack required for implementation is outlined, ensuring that the system not only meets the competition metric but is also designed with the robustness and resilience necessary for deployment in a real-world financial risk management environment.

2. Workshop No. 1 Results

Based on the aforementioned analysis, we can define the competition as a **dynamic and sensitive system**, where small variations in input data or preprocessing can generate large changes in model performance (chaos). The analysis identified the system elements, its various constraints, and critical sensitivities that directly influence the system's stability and predictive capability.

One of the most critical constraints is the **quality and integrity of the dataset**. The presence of missing values, noise, and outliers can distort the relationships between variables, introducing instability into the model's learning process. Furthermore, data variability (resulting from the combination of data types, categorical variables, and different scales) requires rigorous preprocessing methods—such as normalization, encoding, and feature selection—to ensure desired consistency and final performance.

From a systems perspective, the dataset acts as the input and the source of the entire system. Its structure determines the system's behavior under various conditions. When preprocessing is insufficient or inconsistent, **chaotic effects** can emerge: small errors can propagate through the model's workflow, amplifying prediction inaccuracies. This phenomenon reflects the sensitivity to initial conditions characteristic of chaotic systems.

Therefore, the system must include mechanisms to detect and mitigate instability, such as continuous and periodic validation, feedback loops, and error monitoring. These mechanisms will lay the foundation for the design stage, ensuring that the architecture developed in the next phase can effectively manage data variability and preserve the model's robustness.

The system's performance depends on maintaining stability against data irregularities. **Robust preprocessing**, feature validation, and controlled feedback are essential to contain chaotic effects and ensure reliable predictions. These findings provide the analytical basis for the system design developed in Workshop 2, guiding the definition of requirements, architecture, and validation mechanisms.

3. System Requirements Definition

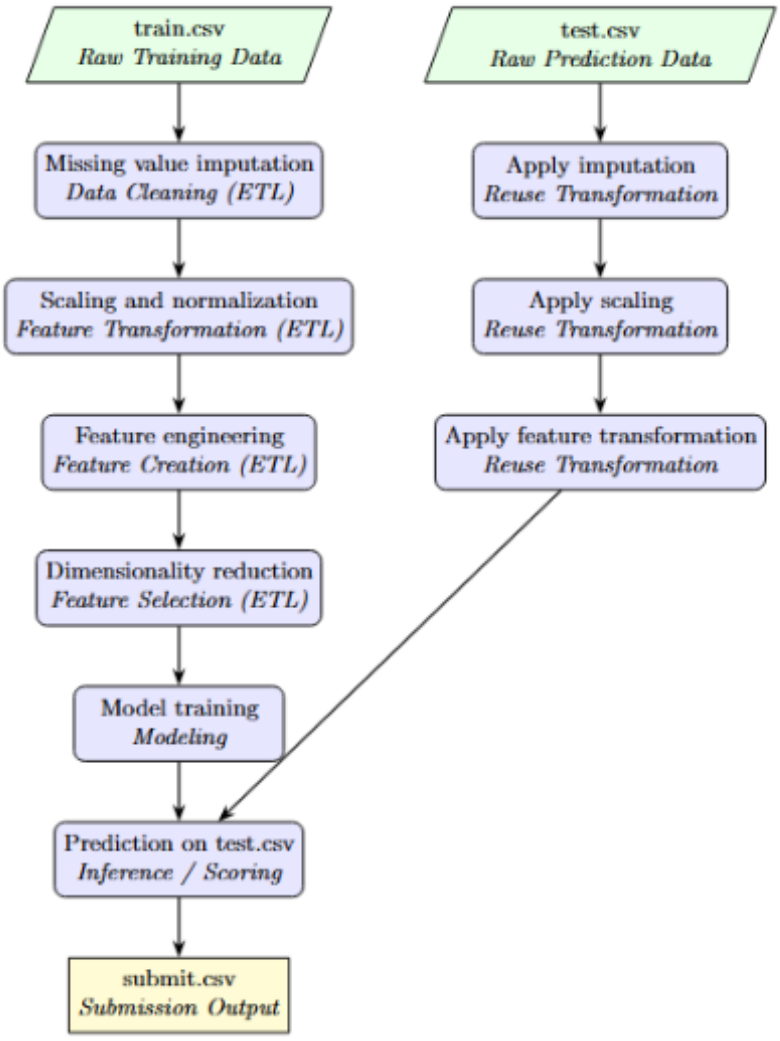
3.1. Functional Requirements.

- The system must generate a predictive output that estimates the client's probability of default based on historical financial and behavioral data.
- It must be capable of data preprocessing, including handling missing values, outliers, and categorical encoding.
- It must evaluate predictions using the **Mean Absolute Error (MAE)** metric to ensure quantitative performance tracking.
- It must allow for feedback and retraining to continuously improve prediction accuracy.

3.2. Non-Functional Requirements.

- **Scalability:** The system must efficiently manage large datasets typical of financial institutions.
- **Reliability:** Predictions must remain consistent in the face of small data disturbances.
- **Transparency:** The model results and decision-making process must be explainable to stakeholders.
- **Maintainability:** The code and data flows must be modular and easily adjustable for future updates.

4. High-Level Architecture



4.1. Module Responsibilities.

Module	Responsibility	Functional Tag
`train.csv`	Historical data with features + target variable.	Raw Data Input (Training)
`test.csv`	New data without labels, only with features.	Raw Data Input (Prediction)
Missing value imputation	Filling null values with statistical rules or fixed values.	Data Cleaning (ETL)
Scaling and normalization	Transforming data scale for uniformity.	Feature Transformation (ETL)
Feature engineering	Creating or combining variables to increase predictive value.	Feature Engineering (ETL)
Dimensionality reduction	Eliminating redundancy or noise by reducing the number of variables.	Feature Selection (ETL)
Model training	Learning a predictive model from the clean data.	Modeling
Prediction on `test.csv`	Applying the model to generate predictions on new data.	Inference / Scoring

Module	Responsibility	Functional Tag
`submit.csv`	File with predictions in the format required for Kaggle.	Submission Output
Apply imputation	Applying the same imputation strategy used in training.	Transformation Reuse
Apply scaling	Applying scaling with the training parameters.	Transformation Reuse
Apply feature transformation	Applying the same selected or generated features.	Transformation Reuse

4.2. How Systems Engineering Principles Shaped Decisions.

- **Modularity:** Clear separation of responsibilities (ingestion, preprocessing, modeling, deployment, monitoring).
  - **Benefit:** Facilitates unit testing, maintenance, and component replacement without affecting the entire system.
- **Reproducibility / Traceability:** Versioned pipelines (`sklearn.Pipeline`), metadata logging, and a model registry.
  - **Benefit:** Allows auditing results, reproducing experiments, and complying with contest restrictions (submission limit).
- **Robustness and Resilience:** Validations on ingestion, anomaly detection, and fallback models.
  - **Benefit:** Mitigates sensitivity to minor changes in preprocessing or noisy data.
- **Scalability:** Design compatible with distributed training, chunked I/O, and container deployment.
  - **Benefit:** Supports large datasets (200k+ rows and ∼800 features) and inference loads.
- **Observability:** Instrumentation to measure drift, performance metrics, and structured logs.
  - **Benefit:** Allows detecting feedback effects and responding with retraining or adjustments.
- **Security and Data Governance:** Access control to datasets, approval for using external data, and masking/anonymization.
  - **Benefit:** Complies with contest restrictions and privacy policies.
- **Iterativity and Maintenance:** CI/CD for preprocessing tests and integrity checks; scheduled or trigger-based retraining pipeline.
  - **Benefit:** The model adapts to changes in distribution and avoids silent degradation.

5. Addressing Sensitivity and Chaos

In the loan default prediction system, sensitivity arises mainly from the preprocessing stage and from the high dimensionality of the dataset. Small variations—such as changes in missing-value treatment, feature scaling, or threshold calibration—can produce large differences in the predicted probability of default. Likewise, chaotic factors stem from economic and human unpredictability: borrowers' decisions, macroeconomic shifts, and feedback between defaults and institutional policies.

To mitigate these effects, the proposed design includes several mechanisms:

- **Data Preprocessing Control:** Each preprocessing step (imputation, normalization, encoding) is version-controlled and reproducible. Using pipelines in `scikit-learn` ensures that the same transformations applied to training data are identically applied to test and deployment data, reducing random variations.
- **Feature Stability Analysis:** A correlation and variance-threshold analysis removes redundant or unstable predictors. Regularization (L1/L2) helps maintain stable coefficients despite noise or multicollinearity.

- **Model Ensemble and Robust Validation:** Instead of relying on a single model, the system combines multiple algorithms—such as LightGBM, XGBoost, and Neural Networks—through stacking or weighted averaging. Cross-validation with stratification maintains class balance and limits overfitting to specific folds.
- **Feedback Loop Monitoring:** A monitoring module compares recent predictions with real defaults as new data arrives. If model drift or feedback effects are detected (e.g., more rejections causing higher default rates), the system triggers retraining or parameter recalibration.
- **Chaos Mitigation Measures:** Random seeds are fixed for reproducibility, and Bayesian optimization is used instead of brute-force grid search to avoid overreacting to random fluctuations in validation metrics.

By integrating these strategies, the system gains resilience to sensitivity and chaotic influences, maintaining consistent predictive behavior even under volatile economic or data conditions.

6. Technical Stack and Implementation Sketch

To implement the architecture efficiently and ensure scalability, the following technical stack is proposed:

6.1. Programming Languages and Frameworks.

- **Python** — primary implementation language due to its versatility in data science.

6.1.1 Libraries:

- Pandas, NumPy for data manipulation.
- `Scikit-learn` for preprocessing pipelines and baseline models.
- LightGBM and XGBoost for high-performance gradient boosting.
- TensorFlow / Keras for deep-learning architectures.
- Optuna for hyperparameter optimization.
- Matplotlib / Seaborn for exploratory and sensitivity visualization.

6.2. System Modules.

- **Data Ingestion Module:** Reads and validates CSV files, handles large datasets, and detects missing or inconsistent records.
- **Preprocessing Module:** Performs encoding, scaling, and imputation; logs preprocessing metadata for traceability.
- **Feature Engineering Module:** Generates interaction features, performs dimensionality reduction (PCA), and evaluates feature importance.
- **Model Training Module:** Trains and validates multiple models under k-fold cross-validation; saves the best ensemble configuration.
- **Evaluation and Deployment Module:** Uses F1-score as the main metric; stores trained models and exposes predictions through an API or dashboard.
- **Monitoring and Feedback Module:** Collects post-deployment data to detect drift and retrain models periodically.

6.3. Design Principles.

- **Modularity:** Each module functions independently for easier debugging and updates.
- **Scalability:** Compatible with distributed training using frameworks like Dask or Spark.
- **Reproducibility:** Environment and dependencies controlled via Docker or virtual environments.
- **Maintainability:** Clear folder structure (`data/`, `src/`, `models/`, `reports/`) and version control with GitHub.
- **Resilience:** Exception handling for data anomalies and automated alerts when input distributions shift significantly.

6.4. Implementation Plan.

- Build preprocessing and modeling pipelines in Python notebooks, then migrate to scripts for automation.
- Integrate version control and continuous integration tests for preprocessing and model evaluation.
- Deploy trained models using FastAPI for inference and connect monitoring dashboards (e.g., Streamlit).
- Schedule periodic retraining to adapt to new data and changing default patterns.