Workshop II

Universidad Distrital Francisco José de Caldas

Faculty of Engineering

Authors
Juan David Zárate Moya 20222020184
Jesus Mateo Munevar Mendez 2023202042
Juan David Romero Morales 20222020102
Kevin David Rincon Valencia 20232020356

Bogotá D.C
2025

Contents

1 Executive Summary & Review of Workshop #1 Findings

Summary of Workshop #1 Findings (Concise)

Goal: Predict `SalePrice` for houses in Ames, Iowa using structured tabular data (`train.csv`: 1460 rows, `test.csv`: 1459 rows, 79 features).

Key Elements Identified: Datasets (`train/test`), features (numeric, categorical, ordinal), target (`SalePrice`), stakeholders (modelers, end-users, deployers), and external factors (macroeconomics, policy) not present in the dataset.

System View: A socio-technical system where the dataset represents a bounded abstraction of the real housing market, which is open and dynamic in nature.

Sensitivity & Complexity: Presence of missing values, skewed distributions in `SalePrice`, variable dispersion, non-linear relationships, feature interactions, and potential multicollinearity among predictors.

Chaos & Unpredictability: External shocks (e.g., economic cycles), measurement biases, and human-driven feedback loops (e.g., pricing trends) introduce elements of unpredictability and chaotic behavior within the system.

Implications for Design: The system architecture must be modular—separating ingestion, preprocessing, modeling, and monitoring—robust to missing or noisy data, and include mechanisms for monitoring and retraining to mitigate concept drift and unexpected performance variations.

2 System Requirements

2.1 Functional Requirements

- **Data Ingestion:** Load and validate `train.csv` and `test.csv` from local or cloud storage.
- **Data Preprocessing:** Handle missing values, encode categorical variables, scale and transform features, and create a feature engineering pipeline (e.g., interactions, polynomial features, log-transform of `SalePrice` for training).
- **Model Training:** Train regression models and support hyperparameter tuning and cross-validation.
- **Model Evaluation:** Compute RMSE on log-transformed prices and provide model comparison reports.
- **Model Serialization & Deployment:** Save the best-performing model(s) and expose a prediction endpoint or batch prediction runner.
- **Monitoring & Retraining:** Track data drift and performance degradation, scheduling retraining when thresholds are exceeded.
- **Reporting & Explainability:** Provide feature importance analysis, partial dependence plots, and SHAP summaries for interpretability.

2.2 Non-functional Requirements

- **Performance:** The training pipeline should complete within a reasonable time on cloud instances (e.g., ≤ 2 hours for full tuning on a single moderate instance), adjustable based on available infrastructure.
- **Scalability:** Components must be horizontally scalable (processing and inference pods) to handle larger datasets efficiently.
- **Reliability & Fault Tolerance:** Pipelines should resume automatically on failure, using checkpoints for long-running tasks.
- **Maintainability:** Ensure code modularity, clear interfaces, and comprehensive unit tests for preprocessing and model components.
- **Security:** Properly manage data access credentials and enforce least-privilege access to storage and deployment environments.
- **Reproducibility:** Employ deterministic random seeds, containerized environments (e.g., Docker), and version control for datasets and trained models.

2.3 Acceptance Criteria

- Successful end-to-end execution: data ingestion → preprocessing → training → evaluation → model serialization.
- The evaluation metric (RMSE on log-transformed prices) must be reported and compared across candidate models.
- Monitoring alerts should trigger automatically when performance drops by a team-defined percentage relative to the baseline.

3 System Architecture and Component Design

Overview

The proposed architecture follows a modular, layered structure that enables robustness, scalability, and maintainability. It is composed of six main subsystems: **Data Ingestion**, **Processing**, **Modeling**, **Serving**, **Monitoring**, and **Orchestration**. Each component is responsible for a specific stage in the data-to-decision pipeline, supporting reproducibility and continuous improvement.

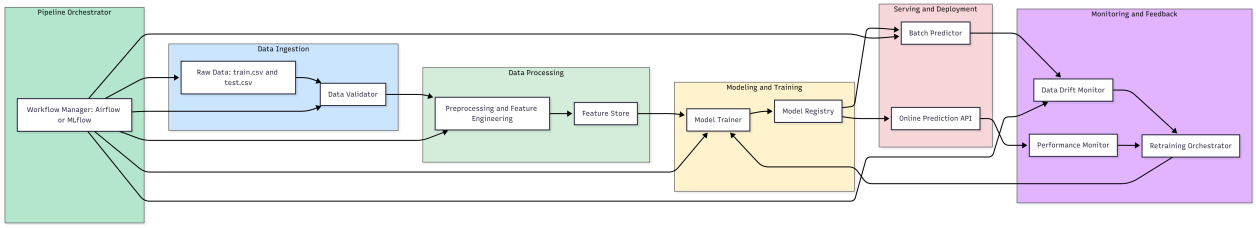

Figure 1: System Design Architecture for the Ames Housing System.

1. Data Ingestion

Components:

- **Raw Data (train.csv / test.csv):** Original datasets containing numerical, categorical, and ordinal variables describing housing features.
- **Data Validator:** Performs schema validation (column names and types), range and uniqueness checks, and logical sanity validations such as ensuring `YearBuilt` $\leq$ current year. Generates missingness and anomaly reports to guarantee data integrity.

2. Data Processing

Components:

- **Preprocessing Pipeline:** Sequential transformer pipeline that handles imputation (numeric/categorical), encoding (target, one-hot, ordinal), transformations (log, sqrt, scaling), outlier mitigation, and engineered feature creation.
- **Feature Store:** Repository of processed features (e.g., Parquet format) and associated metadata describing transformations. Enables consistent access and reproducibility across training and serving phases.

3. Modeling and Training

Components:

- **Model Trainer:** Integrates preprocessing and modeling, executes cross-validation, and performs hyperparameter optimization (Optuna, scikit-optimize). Supports ensembling and stacking strategies for better generalization.
- **Model Registry:** Stores trained model artifacts, metadata (dataset hash, feature version), and evaluation metrics. Provides version control and ensures traceability for deployment.

4. Serving and Deployment

Components:

- **Batch Predictor:** Executes inference over complete datasets (e.g., Kaggle submissions) using the best model from the registry.
- **Online Prediction API:** REST endpoint for real-time, single-record predictions. Supports integration with user applications or monitoring dashboards.

5. Monitoring and Feedback

Components:

- **Data Drift Monitor:** Calculates distribution shifts using metrics like Kolmogorov–Smirnov (KS) or Population Stability Index (PSI) to detect anomalies.
- **Performance Monitor:** Tracks RMSE and other evaluation metrics on a golden dataset, comparing results to defined baselines.
- **Retrain Orchestrator:** Triggers automated retraining upon drift or performance degradation. Handles data snapshotting, model retraining, evaluation, and promotion of the best model.

6. Pipeline Orchestration

Components:

- **Workflow Manager (e.g., Airflow, MLflow):** Coordinates all stages of the pipeline—from ingestion through retraining—using scheduled or event-based triggers. Ensures reproducibility, traceability, and continuous operation.

Summary

This modular design enables a continuous feedback loop: data is ingested and validated, processed and modeled, then deployed and monitored. When significant drift or performance loss is detected, the retraining orchestrator triggers an adaptive learning cycle. This architecture thus ensures long-term stability and adaptability of the system within an evolving real-estate market.

Applied Principles

1. **Holistic Vision:** The system is conceived as an interconnected whole, where ingestion, processing, modeling, deployment, and monitoring modules work together to achieve performance and reliability goals.
2. **Systems Thinking:** The relationships between components are analyzed through cause–effect reasoning, considering feedback loops, dependencies, and possible chaotic behaviors in data or predictions.
3. **Modularity and Hierarchy:** The architecture is organized into independent subsystems (e.g., data validation, model training, monitoring) to improve maintainability, scalability, and component replacement.
4. **Traceability:** Data, features, and model versions are tracked to guarantee reproducibility and complete lifecycle control.
5. **Scalability and Adaptability:** The architecture supports growth in data volume, model complexity, and computational demand by employing optimized formats (e.g., Parquet) and loosely coupled components.
6. **Automation and Control:** The retraining orchestrator manages automatic updates triggered by monitoring alerts, reducing manual intervention and operational risk.
7. **Risk Management and Sensitivity:** Strategies are implemented to handle uncertainty and model sensitivity, consistent with the chaos and sensitivity analysis discussed in Workshop #1.
8. **Continuous Feedback:** Feedback loops and monitoring mechanisms enable ongoing performance evaluation and dynamic adjustment of system operations.

Conclusion

By integrating Systems Engineering principles, the design ensures **coherence, control, adaptability, and sustainability**. This holistic and modular approach supports the complete data science lifecycle—from data acquisition to monitoring—ensuring stable and evolvable behavior under uncertainty.

4 Addressing Sensitivity and Chaos

The housing price prediction system operates within a highly variable environment where both internal and external factors influence results. Sensitivity refers to how small changes in input variables affect the model's output, while chaos arises from unpredictable dynamics in economic and social contexts. The proposed design mitigates these risks through several strategies:

4.1 Normalization and Feature Scaling

Continuous variables such as `LotArea`, `GrLivArea`, and `YearBuilt` are normalized to ensure that no single feature dominates the model due to differences in magnitude. This process stabilizes training and reduces the sensitivity of regression-based algorithms.

4.2 Regularization and Model Robustness

Regularization methods such as Lasso (L1) and Ridge (L2) are implemented to penalize overfitting and prevent extreme coefficients. Ensemble models (Random Forest, Gradient Boosting) are included to increase robustness against noise and random variability in the dataset.

4.3 Cross-Validation

A K-Fold Cross-Validation scheme ensures that model performance is not overly dependent on a single data partition. By averaging across folds, the RMSE metric reflects a more stable and reliable measure of generalization.

4.4 Feedback and Adaptation Loops

The system includes feedback mechanisms where model errors and performance reports inform preprocessing adjustments. For example, recurring high residuals in specific neighborhoods may indicate missing socioeconomic features, prompting data enrichment or feature engineering updates.

4.5 Monitoring and Continuous Improvement

A monitoring log tracks preprocessing warnings (missing data, outliers) and model drift indicators. This allows the team to iteratively refine the pipeline, ensuring sustained accuracy even as market conditions or input data change over time.

—

5 Technical Stack and Implementation Plan

5.1 Programming Environment

The project will be implemented using **Python 3.11**, given its compatibility with data science libraries and Kaggle's ecosystem.

5.2 Libraries and Frameworks

- **pandas** and **numpy**: Data manipulation and numerical operations.
- **scikit-learn**: Machine learning algorithms, pipelines, and validation tools.
- **matplotlib** and **seaborn**: Data visualization and feature analysis.
- **shap**: Interpretability analysis through SHAP values and feature contribution plots.
- **xgboost** or **lightgbm**: High-performance gradient boosting models for regression.

5.3 Design Patterns and Modularity

The system applies the **Pipeline Pattern** to ensure that data flows sequentially through preprocessing, feature engineering, training, and evaluation modules. The **Factory Method Pattern** is used for flexible model selection and initialization.

5.4 Implementation Overview

The implementation process follows these main steps:

1. **Data Preparation:** Load and inspect datasets (`train.csv`, `test.csv`), verify data types and missing values.
2. **Preprocessing:** Handle missing data through imputation, remove outliers, and encode categorical variables.
3. **Feature Engineering:** Generate new variables (e.g., total area, house age) and apply scaling.
4. **Model Training:** Train models using Linear Regression, Random Forest, and Gradient Boosting methods.
5. **Validation:** Apply 5-Fold Cross-Validation to calculate RMSE and compare models.
6. **Interpretation:** Visualize feature importance and residuals using SHAP and correlation plots.
7. **Output:** Save final predictions and metrics to the `/results` folder.

5.5 Integration Between Modules

All components are linked through a main execution script (`main.py`) that manages configuration files and executes the pipeline in sequence. Each module (e.g., preprocessing, modeling, evaluation) is stored in a dedicated folder within the repository to ensure readability and reusability.