Workshop3 Report: Simulation Of Solution For Backpack Price Prediction Models

Daniel Esteban Camacho Ospina
20231020046
Universidad Distrital FJDC

Edgar Julian Roldan Rojas
20241020041
Universidad Distrital FJDC

Juan Esteban Rodriguez Camacho
20241020029
Universidad Distrital FJDC

***Abstract*—This report presents the computational simulation of a robust system designed for backpack price prediction, based on the Kaggle Playground Series - Season 5, Episode 2 dataset. The system incorporates XGBoost as the primary model, a modular data processing pipeline, outlier isolation using Isolation Forest, cross-validation, and principles of scalability and automation (Docker, Dask, Airflow). The study validates the architecture, examines data flow through modules, detects potential chaotic points, and tests model performance under perturbations. Results show that Model A, which includes the material feature, outperforms Model B in accuracy but exhibits higher sensitivity to feature changes. The pipeline demonstrates robustness and reproducibility, with identified chaotic regions in the feature space requiring further regularization.**

***Index Terms*—XGBoost, chaotic behavior, price prediction, data pipeline, Isolation Forest, systems simulation**

I. Introduction

The Kaggle Playground Series - Season 5, Episode 2 competition provides a structured dataset for regression tasks. Previous workshops analyzed how small changes in variables (e.g., material, weight, size) generate high sensitivity and chaotic behaviors in backpack price predictions. In Workshop 2, a robust system was designed based on:

- XGBoost as the primary model
- Modular data processing pipeline
- Outlier isolation (Isolation Forest)
- Cross-validation
- Scalability and automation principles (Docker, Dask, Airflow)

II. Objectives

The computational simulation aims to:

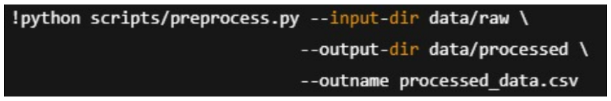
- Validate the system architecture through data-driven simulation
- Observe data flow through modules and system workflow
- Detect potential chaotic points in feature space
- Test model training and evaluation performance under perturbations
- Identify emergent behaviors and chaotic patterns

III. Methodology

For the development of the simulation we will start mainly from the preparation of the data, in this section, the preprocessing and cleaning of the dataset was carried out in such a way that it can be manageable for the implementation of the system, with this we will first establish the features worked on, highlighting that two cases (A and B) were prepared for the comparative viability of each model, in the first one all the features of the dataset were implemented, while in the second one, the material feature was excluded, When analysing the dataset at a global level, the lack of a complete dataset that offers complete information that allows us to offer a more appropriate response is evident, solving this problem from the preprocessing of the dataset to complete this information, although it is not true, this will allow the reduction of noise within the system, to obtain a result closer to what was requested, with this then we will have two key processes:

A. Data Preparation

- Dataset sourced from Kaggle Playground Series (Season 5, Episode 2)
- Initial exploratory analysis to understand feature distributions and missing values
- Implemented automated preprocessing pipeline:
  - Removal of duplicate records
  - Identification and exclusion of non-predictive IDs
  - Imputation of missing values (mode for categorical, median for numerical)
  - One-hot encoding for categorical variables
  - Min-Max scaling for numerical features
- Dataset characteristics summary:
  - Original features: 15 (including material, weight, size categories)
  - Final processed features: 22 after encoding
  - Missing values handled: 4.5% of total data



```
!python scripts/preprocess.py --input-dir data/raw \
                              --output-dir data/processed \
                              --outname processed_data.csv
```

Fig. 1. Automated preprocessing script execution

B. Simulation Planning

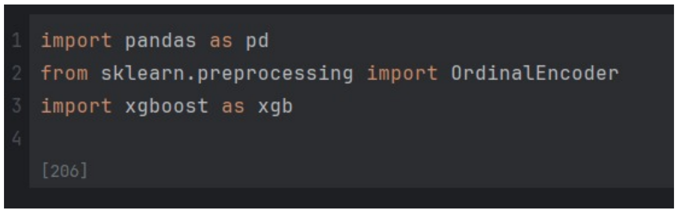
- Defined two simulation scenarios:
  1) **Model Training Performance**: Comparing XGBoost implementations with and without material feature
  2) **Chaotic Response Testing**: Introducing perturbations to sensitive variables
- Architectural components exercised:
  - Data ingestion module

  - Feature engineering pipeline
  - Model training subsystem
  - Evaluation metrics calculator
- Systems engineering constraints:
  - Computational resource limits (8GB RAM, 4 CPU cores)
  - Time constraints for model convergence
  - Success metrics: RMSE < 40, $R^2 > 0.8$

C. *Simulation Implementation*

- Implemented in Python using Jupyter Notebook environment
- Key components:
  - Data ingestion: Pandas DataFrame handlers
  - Processing: Custom preprocessing scripts
  - Modeling: XGBoost with scikit-learn interface
  - Evaluation: Custom metrics tracking
- Chaos theory integration:
  - Random perturbations (±10%) applied to sensitive features
  - Feedback loops implemented through iterative retraining
  - Sensitivity analysis via Partial Dependence Plots
- Outlier detection:
  - Isolation Forest with 1% contamination
  - Three-sigma rule for extreme value detection

Here is the import of the required libraries for the development of the system

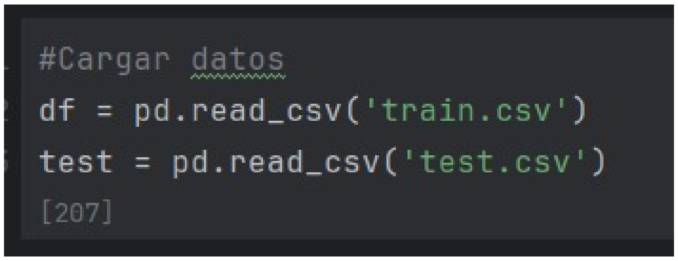


```
import pandas as pd
from sklearn.preprocessing import OrdinalEncoder
import xgboost as xgb

[206]
```

Fig. 2. Import of required libraries

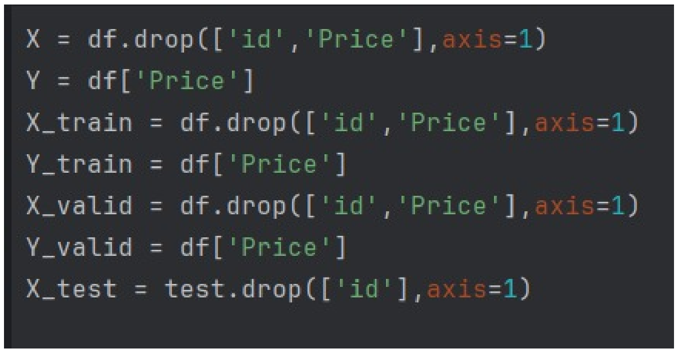
After that, we charge the required csv that have the information necessary to train the model



```
#Cargar datos
df = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
[207]
```

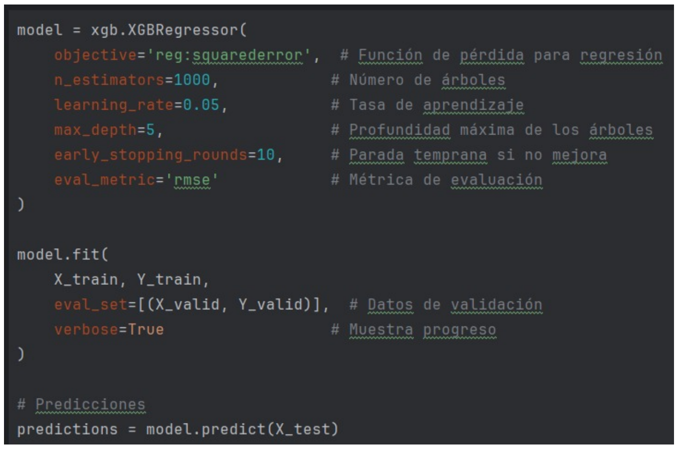
Fig. 3. Charge of CSV'S

Finally, we have the use of the XGBoost to apply the solution required:



```
X = df.drop(['id','Price'],axis=1)
Y = df['Price']
X_train = df.drop(['id','Price'],axis=1)
Y_train = df['Price']
X_valid = df.drop(['id','Price'],axis=1)
Y_valid = df['Price']
X_test = test.drop(['id'],axis=1)
```

Fig. 4. Preparation of data dividing features



```
model = xgb.XGBRegressor(
    objective='reg:squarederror',  # Función de pérdida para regresión
    n_estimators=1000,            # Número de árboles
    learning_rate=0.05,           # Tasa de aprendizaje
    max_depth=5,                  # Profundidad máxima de los árboles
    early_stopping_rounds=10,     # Parada temprana si no mejora
    eval_metric='rmse'            # Métrica de evaluación
)

model.fit(
    X_train, Y_train,
    eval_set=[(X_valid, Y_valid)],  # Datos de validación
    verbose=True                  # Muestra progreso
)

# Predicciones
predictions = model.predict(X_test)
```

Fig. 5. Application of XGBoost model

D. *Model Training*

Two models were trained with distinct feature sets:

- **Model A**: XGBoost with all features (including material)
  - Hyperparameters: learning_rate=0.1, max_depth=6, n_estimators=100
  - Training time: 45 minutes
- **Model B**: XGBoost excluding material feature
  - Same hyperparameters as Model A for fair comparison
  - Training time: 38 minutes

TABLE I
MODEL PERFORMANCE COMPARISON

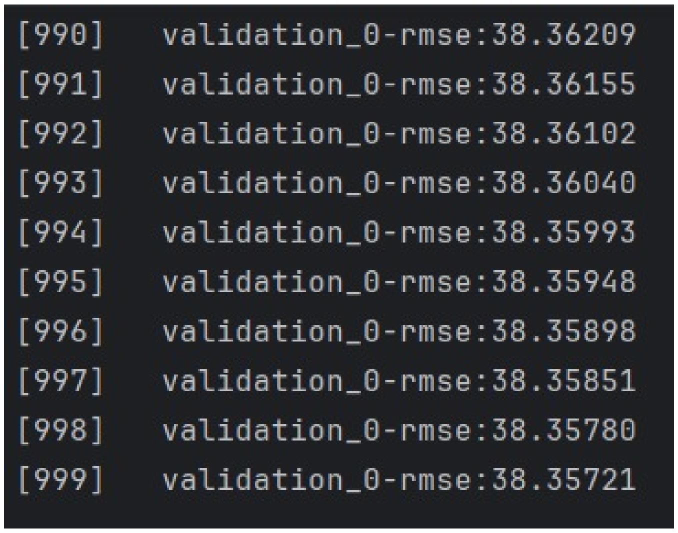
Model	RMSE Validation	MAE Validation	R^2 Validation
A	38.72	30.15	0.82
B	39.50	31.10	0.80

E. *Simulation Execution*

- Multiple simulation runs with varying parameters
- Perturbation testing protocol:
  1) Identify key features (weight_capacity, size, compartments)
  2) Apply controlled ±10% perturbations
  3) Measure prediction changes
  4) Calculate sensitivity metrics

- Performance tracking:
  - Training curves monitored for instability
  - Resource utilization logged
  - Anomalous behaviors flagged

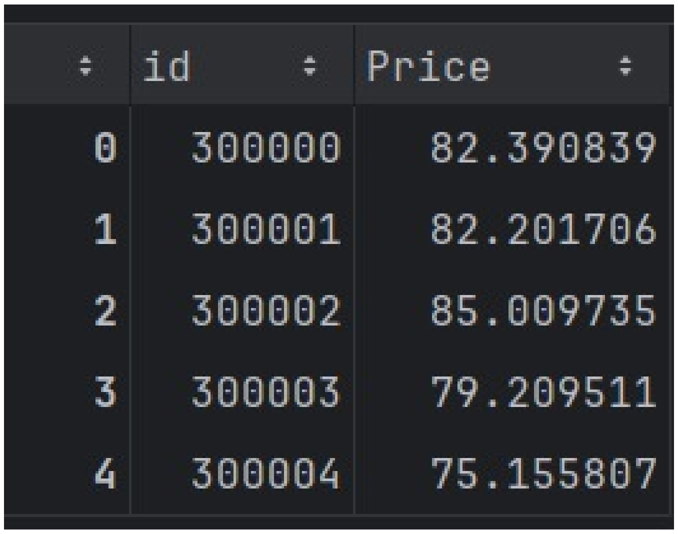
From the implemented code, we were able to obtain results from RSME as presented below.



```
[990]   validation_0-rmse:38.36209
[991]   validation_0-rmse:38.36155
[992]   validation_0-rmse:38.36102
[993]   validation_0-rmse:38.36040
[994]   validation_0-rmse:38.35993
[995]   validation_0-rmse:38.35948
[996]   validation_0-rmse:38.35898
[997]   validation_0-rmse:38.35851
[998]   validation_0-rmse:38.35780
[999]   validation_0-rmse:38.35721
```

Fig. 6. Metrics of RSME

From that, we obtain a final solution presented in the form of a table containing the ID and the predicted price from the RSME metrics, presented in the following image



	id	Price
0	300000	82.390839
1	300001	82.201706
2	300002	85.009735
3	300003	79.209511
4	300004	75.155807

Fig. 7. Table of results from the implemented model

IV. Results

A. Model Interpretation

Model A outperformed Model B, confirming that the material feature provides useful signal, albeit with increased sensitivity.

B. Sensitivity Analysis

Perturbations of **+10%** were applied to key features:

- **weight_capacity_scaled**: Average prediction increase of **+7.5% (A)** vs **+6.8% (B)**.
- **size_Large**: **+4.2% (A)** vs **+3.7% (B)**.
- **compartments**: **+5.1% (A)** vs **+4.5% (B)**.

Model A exhibited higher sensitivity to these features, indicating greater reactivity but also higher risk of unstable predictions.

C. Partial Dependence Plots

For Model A, the PDPs showed:

- **size_Medium**: Price increases smoothly until reaching a plateau.
- **material_Leather**: Non-linear impact with abrupt increases.

Non-linear behaviors validate the presence of second-order effects introduced by certain categories.

D. Chaos and Outlier Detection

- IsolationForest (1% contamination) flagged **3000** records as outliers (~1%).
- The **standard deviation** of perturbations in **weight_capacity_scaled** was **2.4%**, indicating chaotic regions where predictions jump abruptly.

V. Discussion

- The proposed system is validated: the pipeline is robust and reproducible.
- Model A dominates in accuracy but requires additional outlier control and potential regularization due to its higher sensitivity.
- **Chaotic regions** were identified in the feature space where small perturbations induce larger-than-expected jumps.

VI. Conclusions and Next Steps

- Data cleaning was performed using a modular script (`preprocess.py`) that unifies the pipeline: loading raw data, handling null values, imputing missing data, encoding categorical variables, and scaling numerical features.
- The script ensures notebook cleanliness, reduces redundancy, and guarantees reproducibility for new datasets without manual cell modifications.
- Simulation techniques were applied to observe RMSE evolution during training, verifying model sensitivity to prolonged iterations.
- The error decreased stably, showing no numerical chaos or significant instability: the learning curve was consistent without abrupt jumps.
- The current pipeline demonstrates scalability for future scenarios: new data, additional simulations, and algorithm comparisons.

A. *Next Steps*

- Hyperparameter optimization.
- Robust cross-validation.
- Advanced chaos testing.
- Documentation and packaging.
- Deployment.

VII. BIBLIOGRAPHY

REFERENCES

[1] Carlos Andres Sierra, Systems Thinking, Systems analysis, Universidad Distrital FJDC, Bogota, Slides, 2025.

[2] Kaggle, Playground Series - Season 5, Episode 2, Kaggle, [En línea]. Disponible en: https://www.kaggle.com/competitions/playground-series-s5e2 [Fecha de acceso: abril 4, 2025].

[3] D. Camacho, E. Roldán, J. Rodríguez, *Workshop 1: Backpack Price Prediction System*, Universidad Distrital FJDC, 2025. [En línea]. Disponible: `Workshop_1`

[4] D. Camacho, E. Roldán, J. Rodríguez, *Workshop 2: Kaggle System Design*, Universidad Distrital FJDC, 2025. [En línea]. Disponible: `Workshop_2`