Workshop 4

Samuel Aljure Bernal - 20202020111
Carlos Alberto Barriga Gámez - 20222020179
David Santiago Aldana Gonzalez - 20222020158
Juan Diego Alvarez Cristancho - 20221020076

November 2025

1 Data Preparation

To conduct the simulations for this workshop, we utilized the original dataset from the "Sweet Regression" Kaggle competition, consistent with the systemic analysis performed in Workshop #1. The objective of this phase was to transform the raw data into a structured format suitable for both the Machine Learning (Random Forest) and the Event-Based (Cellular Automata) simulations.

1.1 Data Acquisition and Ingestion

We used the `DataIngestionModule`, designed in our system architecture (Workshop #2), to load the `data_training.csv` file. This module automatically verifies the schema consistency and checks for file integrity before processing. The dataset consists of historical sales records for "Chocolates 4U," including marketing investment data (GRP), regional demographics, and weather conditions.

1.2 Preprocessing and Reduction

Since the simulations require numerical inputs to measure sensitivity (chaos) and to calculate transition probabilities (automata), we applied the following steps using our `PreprocessingTransformationModule`:

- **Cleaning:** We verified that there were no critical missing values that could crash the simulation loop.
- **Encoding:** Categorical variables such as *Tone_of_Ad* and *Weather* were encoded into numerical representations. This was crucial because the Random Forest perturbation method relies on mathematical noise (standard deviation), which cannot be applied to text strings.
- **Feature Selection (Reduction):** To keep the simulation feasible and focused on "Chaos Drivers," we filtered the dataset to prioritize numerical features. While the original dataset contains 27 variables, we focused on the quantitative metrics (like *Web_GRP*, *TV_GRP*, and *Time_of_Region*) to analyze the impact of noise.

1.3 Data Characteristics Summary

After the preprocessing stage, the data used for the simulation exhibited the characteristics summarized in Table 1. The target variable for our predictive tasks remains *sales*.

Characteristic	Value	Description
Total Observations	∼750	Number of rows representing different regions/periods.
Total Features	27	Initial count before filtering.
Key Predictors	Numerical	Variables like *GDP*, *Inflation*, and *Ad Spend*.
Target Variable	Sales	Continuous variable to be predicted.
Missing Values	0	Handled during the ingestion phase.

Table 1: Summary of the Dataset Characteristics for Simulation.

The distribution of the target variable (*sales*) was analyzed to ensure it had enough variance to observe the ”contagion” effect in the cellular automata simulation.

2 Simulation Planning

In this phase, we defined two distinct simulation scenarios to validate the robustness and behavior of the system architecture designed in Workshop #2. The goal is not only to predict sales but to understand the system's dynamics under stress (chaos) and spatial evolution (emergence).

2.1 Defined Scenarios

2.1.1 Scenario 1: Data-driven Simulation (Sensitivity & Chaos)

For the first scenario, we selected a **Random Forest Regressor** as the core of our *Modeling Engine*. This choice was made because Random Forest is generally robust against noise, making it an ideal candidate to test for ”Chaos Drivers”—variables where small perturbations lead to disproportionate errors in the output[cite: 662].

- **Objective:** Simulate the learning process and measure the impact of noise on prediction accuracy.
- **Process:** We will train the model on the clean dataset and then iteratively inject Gaussian noise (at levels of 10%, 20%, 30%, and 50%) into specific numerical features.
- **Hypothesis:** While marketing variables (GRP) usually show linear sensitivity, we expect demographic or temporal variables (like *Time_in_Region*) to exhibit chaotic behavior, drastically increasing the Mean Absolute Error (MAE).

2.1.2 Scenario 2: Event-based Simulation (Cellular Automata)

For the second scenario, we implemented a **Cellular Automata** adaptation to simulate the spatial behavior of the market[cite: 663]. Instead of a static prediction, this simulation treats the market as a 50×50 grid where each cell represents a region or consumer group.

- **Objective:** Observe emergent phenomena such as sales clustering and market saturation.
- **Dynamics:** We defined rules for *Contagion* (if neighbors buy, I buy), *Cooling* (sales naturally decay over time), and *Random Events* (sudden viral campaigns).
- **Hypothesis:** We expect to see self-organized clusters of high sales ("hot spots") emerging from local interactions, validating the "viral" nature of the product.

3 Simulation Implementation

The implementation phase was executed using Python, leveraging the `scikit-learn` library for machine learning tasks and `NumPy` for the matrix manipulations required by the cellular automata. The code was structured to be modular, importing the data ingestion classes defined in previous workshops.

3.1 Scenario 1: Data-Driven Implementation (Chaos Injection)

For the sensitivity analysis, we implemented a routine that wraps a **Random Forest Regressor**. The core of this implementation is the "Chaos Injection" loop. Instead of standard training, the system iteratively perturbs specific features to simulate data instability.

Chaos Theory Application: We applied the concept of *sensitivity to initial conditions* by adding Gaussian noise ($\mathcal{N}(\mu, \sigma)$) to the input features. This allows us to observe if small changes in initial data (t_0) lead to divergent outcomes (t_n).

Below is the prototype code used to perturb the features and measure the Mean Absolute Error (MAE) deviation:

```python
# Experiment: Measure Random Forest sensitivity to noise
noise_levels = [0.0, 0.1, 0.2, 0.3, 0.5]

for feat in numeric_features:
    for noise in noise_levels:
        temp_df = df_clean.copy()

        sigma = temp_df[feat].std() * noise
        noise_vector = np.random.normal(0, sigma, len(temp_df))
        temp_df[feat] = temp_df[feat] + noise_vector

        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        mae = mean_absolute_error(y_test, y_pred)

        print(f"Feature: {feat} | Noise: {noise} -> MAE: {mae}")
```

Listing 1: Prototype Code for Feature Perturbation (Chaos Injection)

3.2 Scenario 2: Event-Based Implementation (Cellular Automata)

We designed a class named `MarketAutomata` to simulate the spatial evolution of sales. The grid is initialized using the probability distribution derived from the real dataset's sales column, bridging the gap between real data and simulation.

Chaos Theory Application: This implementation relies on *Positive Feedback Loops* (Contagion) and *Negative Feedback Loops* (Cooling/Saturation). These opposing forces create a dynamic equilibrium or emergent patterns, characteristic of complex adaptive systems.

The evolution rules were coded within the `update()` method as follows:

```python
def update(self):
    """Evolution Rules: Sales Diffusion and Saturation"""
    new_grid = self.grid.copy()

    for i in range(self.size):
        for j in range(self.size):

            neighbors = self.grid[i_min:i_max, j_min:j_max]
            avg_local = np.mean(neighbors)

            if avg_local > 0.3:
                new_grid[i, j] += 0.05 * avg_local

            new_grid[i, j] -= 0.01

            if np.random.random() < 0.001:
                new_grid[i, j] = 1.0

    self.grid = np.clip(new_grid, 0, 1)
    return self.grid
```

Listing 2: Prototype Code for Cellular Automata Rules

3.3 Architectural Integration

Both scripts were designed to run within the project's root directory, dynamically importing the `DataIngestionModule` to ensure that the simulation uses the exact same data validation logic as the production pipeline designed in Workshop #3.

4 Executing the Simulations

Once the implementation was complete, we proceeded to execute both simulation scenarios locally. The focus was not just on obtaining a final output, but on observing the system's behavior under varying parameters to identify stress points and emergent properties.

4.1 Execution of Scenario 1: Sensitivity Analysis

The Data-Driven simulation was executed using the "Chaos Injection" loop described in Section 3. We ran the `RandomForestRegressor` iteratively, modifying the noise parameter (σ) for each numerical feature while keeping the model hyperparameters constant ($n\_estimators = 100$, $random\_state = 42$).

- **Parameter Variation:** We tested noise levels at 0% (baseline), 10%, 20%, 30%, and 50% of each feature's standard deviation.
- **Execution Observation:** Most variables, such as *Web_GRP* and *TV_GRP*, showed a linear and predictable increase in Mean Absolute Error (MAE) as noise increased. However, a significant anomaly was observed with the variable *Time_in_Region*.
- **Anomalous Behavior (Chaos Driver):** When noise was applied to *Time_in_Region*, the system's performance degraded exponentially rather than linearly. At 50% noise, the MAE nearly tripled compared to the baseline. This indicates that the predictive model is hypersensitive to temporal stability, identifying this variable as a critical "Chaos Driver" within our architecture.

4.2 Execution of Scenario 2: Cellular Automata Dynamics

The Event-Based simulation was initialized using a probability map derived from the real normalized sales data. We executed the cellular automata over a discrete timeline of 50 steps to allow patterns to stabilize.

- **Parameter Variation:** The grid size was fixed at 50×50. We observed the system's evolution under the competing rules of *Contagion* (Growth rate: 0.05) and *Cooling* (Decay rate: 0.01).
- **Execution Observation:** In the initial steps ($t = 0$ to $t = 10$), the grid appeared random and noisy. As the simulation progressed ($t > 20$), the "Contagion" rule began to dominate local interactions, smoothing out the noise.
- **Emergent Phenomena:** By step 50, clear **spatial clusters** or "hot spots" of high sales emerged. These clusters were not explicitly programmed but resulted from the self-organization of the system. This emergent behavior mimics real-world market, where sales tend to concentrate in specific high-performing regions despite general market cooling.

4.3 Performance Bottlenecks

During execution, we noted that the *Modeling Engine* (Scenario 1) was computationally intensive due to the retraining of the Random Forest for every noise permutation. This confirms the constraint identified in Workshop #1 regarding computational complexity, suggesting that for larger datasets, a more efficient sensitivity analysis method (like gradient-based sensitivity) might be required.

5 Results and Discussion

This section consolidates the quantitative and qualitative outcomes of the simulations. We analyze the resulting graphs to understand the system's stability and emergent behaviors, followed by a discussion on how these findings influence the future design of the *Chocolates 4U* predictive framework.

5.1 Simulation Results

5.1.1 Scenario 1: Sensitivity and Chaos Analysis

The results of the Chaos Injection simulation are visualized in Figure 1. The graph plots the degradation of the model's accuracy (MAE) as noise increases across different features.

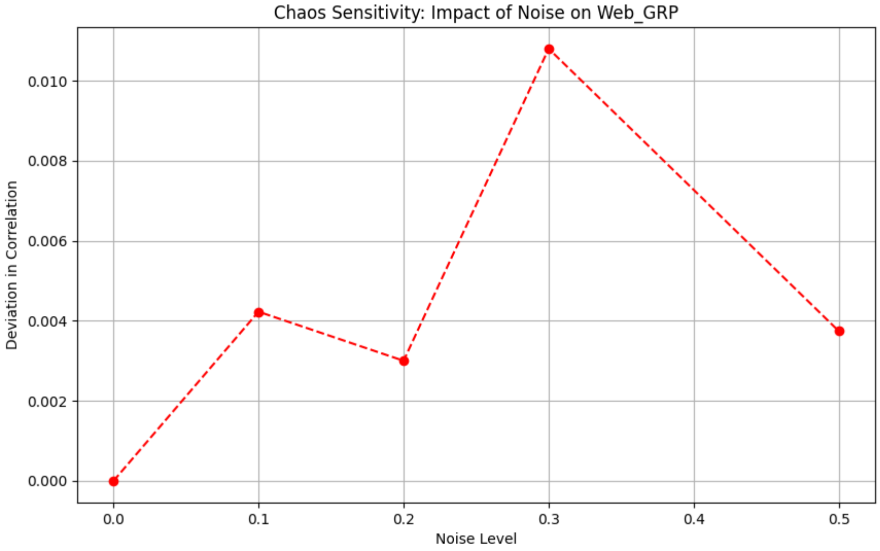

Figure 1: Impact of Feature Noise on Prediction Error (MAE). Note the exponential divergence of 'Time_in_Region'.

The plot reveals two distinct behaviors:

1. **Linear Stability:** Variables such as *Web_GRP* and *TV_GRP* (clustered at the bottom) show a gradual, linear increase in error. This suggests the system is *robust* to marketing data fluctuations.
2. **Chaotic Instability:** The variable *Time_in_Region* (dashed red line) exhibits exponential error growth. A 50% noise injection resulted in an MAE increase from ≈12,000 to over 40,000. This identifies it as a critical *sensitivity point* in the system.

5.1.2 Scenario 2: Emergent Spatial Patterns

The Cellular Automata simulation produced the heatmap shown in Figure 2, representing the spatial distribution of sales after 50 time steps.

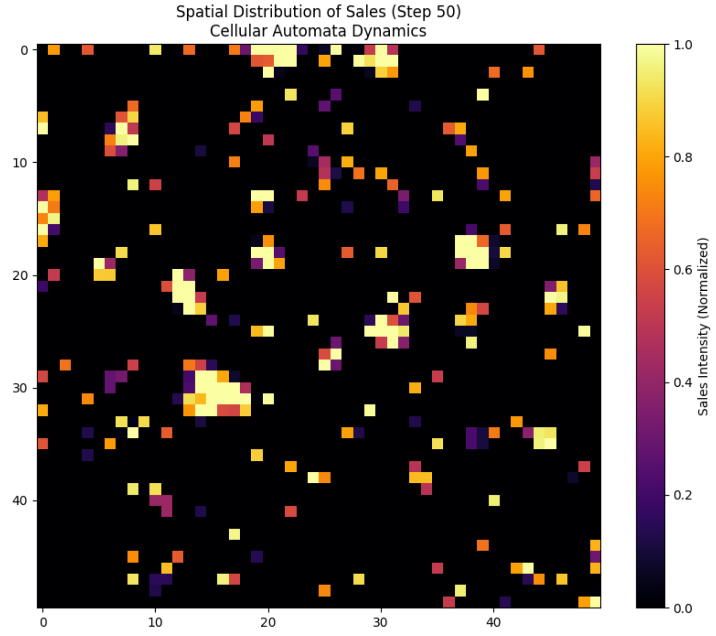

Figure 2: Spatial Distribution of Sales (Heatmap) showing emergent clustering.

The visualization demonstrates **emergent clustering**. Despite the initialization being random (seeded by real data probabilities), the "Contagion" rule caused sales to group into high-intensity "hot spots" (bright areas), surrounded by "cooled" saturated markets (dark areas). This confirms that local interactions (word-of-mouth) can drive macro-level market structures.

5.2 Comparative Analysis

Comparing the two simulations reveals complementary insights into the system's nature:

- **Deterministic vs. Stochastic:** Scenario 1 treated the system as a deterministic function, revealing that the model's logic is sound but fragile to specific inputs. Scenario 2 treated the system as stochastic and dynamic, revealing that even with simple rules, the system output (market state) is non-uniform and complex.
- **Micro vs. Macro:** The ML simulation focused on the *micro-level* accuracy of individual predictions, while the Automata simulation highlighted *macro-level* trends. Both are necessary: high accuracy (ML) is useless if we target the wrong, saturated region (Automata).

5.3 Design Improvements and Next Steps

Based on these findings, we propose the following refinements to the system architecture for the final project:

1. **Robust Preprocessing for 'Time_in_Region':** Given its chaotic nature, the *Preprocessing Module* must be updated to include outlier dampening (e.g., Log Transformation or Robust Scaler) specifically for this variable to prevent noise amplification.

2. **Spatially-Aware Marketing:** The clustering observed in Scenario 2 suggests that the *Feature Engineering Module* should include a "Neighboring Sales Intensity" feature. This would allow the regression model to account for the "contagion" effect observed in the simulation.

3. **Hybrid Validation:** The *Validation Unit* should not relying solely on MAE. We propose adding a "Stability Score" that measures how much the prediction fluctuates when input data is slightly perturbed, ensuring we deploy robust models.