



CAT Datathon 1.0

Team :

El Habida (IEEE ManCSC)

Team Members :

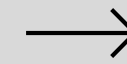
Mohamed Magdy

Abdullah Eldabbousy

Basel Ahmed

Abdelrahman El hussein

Problem Statement



The company aims to accurately predict the metric cost of retail transactions.



The dataset contains incomplete, noisy, and inconsistent data from multiple sources.



Uncleaned data makes it difficult to extract reliable insights or build accurate models.



Our task is to clean, preprocess, and apply machine learning to deliver robust predictions.

Dataset Overview

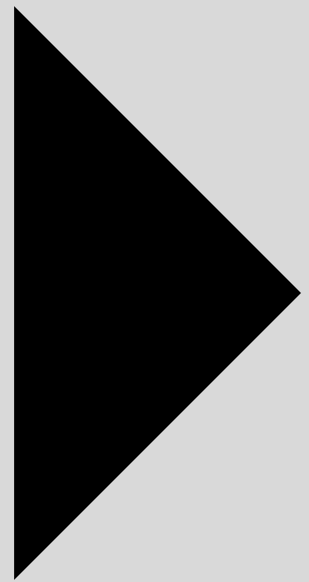


The project is based on the Brazilian Marketplace Orders Dataset, which contains more than 40,000 customer orders collected from various marketplaces across Brazil.

- Customer & Order Information: Person description, yearly income, customer order details, and review scores.
- Product & Packaging: Gross weight, net weight, package weight, and recyclability indicators.
- Store Data: Store type, sales, cost, and area breakdown (grocery, frozen, meat).
- Promotions & Market: Promotion names and additional market features.
- Geographical Data: Customer and seller location (city, state, latitude, longitude).

Target Variable: Cost – representing the total cost of each order.

Problem 1 : Inconsistent Column Names



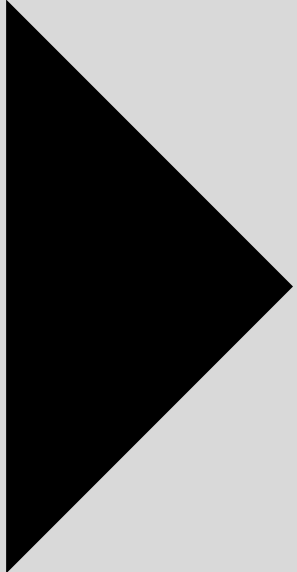
The datasets had different column naming conventions (e.g., personDescription, person_description, Person Description). This prevented us from merging them directly.

Solution:

We created a column mapping dictionary and a standardization function to rename all columns into a consistent format.

After standardization, we successfully merged the training subsets into one dataset.

Problem 2 : Unstructured Text in Person Description



The Person Description column contained multiple pieces of information (e.g., marital status, gender, number of children, education, profession). Since this data was written as free text, it was difficult to use directly in modeling.

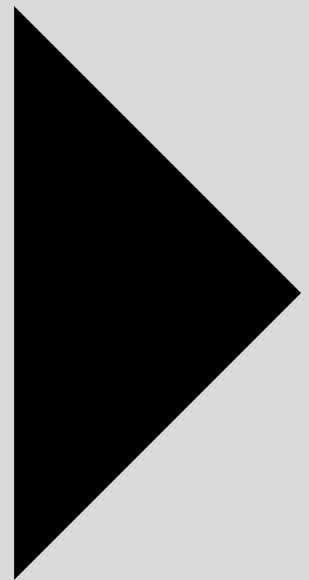
Solution:

We developed a text processing function to extract and structure this information into new columns:

- marital_status
- gender
- children_number
- education
- profession

This transformation allowed us to use the information effectively as input features for our model.

Problem 3 : Inconsistent Yearly Income

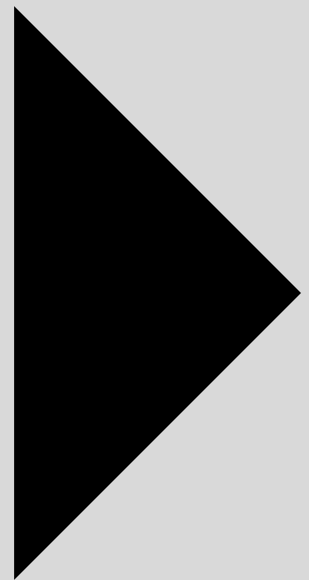


The Yearly Income column contained highly inconsistent formats: some values were written with currency symbols (e.g., “\$5,000”), others in shorthand (e.g., “5k”), while some were monthly instead of yearly (e.g., “\$400 monthly”), and many rows had missing or invalid placeholders (“nil”, “unknown”, “-999”, etc.).

Solution:

We developed a cleaning function that standardizes all values into numeric yearly amounts, handles missing data, and unifies different representations into a consistent format.

Problem 4 : Unstructured Customer Order



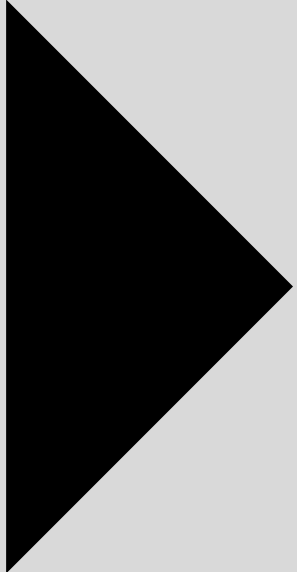
The Customer Order column was highly unstructured. Some rows contained full descriptive sentences mixing product type, department, and brand, while others had missing or invalid values such as “Not Specified”, “Unknown”, or “Nil”. This made it impossible to use the column directly for modeling.

Solution:

We created a parsing function that extracts and standardizes key attributes — Product Type, Department, and Brand — from the unstructured text.

Invalid and noisy entries were converted into clean NaN values for consistent handling.

Problem 5 : Inconsistent Product Weights



The dataset contained three related columns: Gross Weight, Net Weight, and Package Weight. However, many rows were missing values in one or more of these fields, and units were not standardized (e.g., kg, g, lbs, oz). This inconsistency made the weight data unreliable for modeling.

Solution:

We implemented a cleaning function to:

- Standardize all weight values into kilograms.
- Handle invalid or noisy entries (e.g., “missing”, “NaN”, “unknown”).
- Recalculate missing values logically (e.g., $\text{Gross} = \text{Net} + \text{Package}$, $\text{Package} = \text{Gross} - \text{Net}$).

This ensured consistent and complete weight data across the dataset.

Problem 6 : Inconsistent Recyclability Field



The Is Recyclable? column had highly inconsistent values. Some entries were proper (yes, no, recyclable), but many were noisy (non recyclable, not available, missing, nil, etc.). This inconsistency created difficulties in using the field as a categorical variable.

Solution:

We created a cleaning function that:

- Normalized all values to lowercase and removed unnecessary characters.
- Mapped noisy or invalid entries to a standard category (unknown).
- Unified the values into three consistent classes: yes, no, and unknown.

This made the field usable as a clean categorical feature for modeling.

Problem 7 : Inconsistent Promotion Name Field

The 'Promotion Name' column contained highly inconsistent values. Many entries were missing or noisy (e.g., "missing", "tbd", "nil", "not specified"), while others had inconsistent formatting (e.g., extra spaces, unwanted punctuation). This made it difficult to analyze and categorize promotion data.

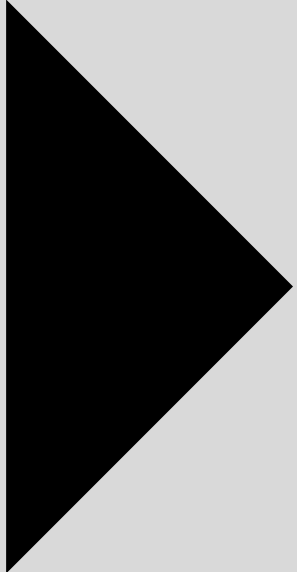
Solution :

Data Cleaning Function

We created a cleaning function that:

- Handled a wide variety of missing or null-like values by replacing them with a standard None.
- Normalized all values to lowercase and removed extra whitespace.
- Removed common punctuation characters to ensure consistency.

Problem 8 : Inconsistent Store Kind Field



The 'Store Kind' column was highly inconsistent, with many entries containing typos, abbreviations, or non-standard variations (e.g., "gourment" instead of "gourmet," or "super mkt"). This made it impossible to group and analyze store types using simple string matching.

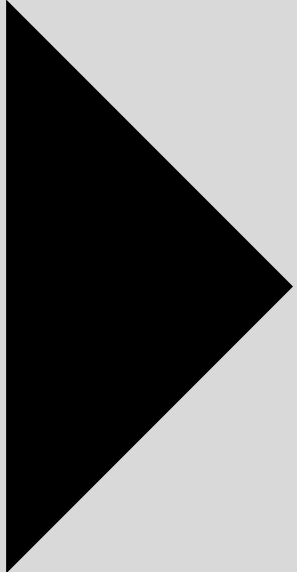
Solution :

Data Cleaning with Fuzzy Matching

We developed a cleaning function that utilized fuzzy string matching to address these inconsistencies. The function performed the following steps:

- It handled missing or empty values by assigning them to a standard "unknown" category.
- It normalized the text by converting it to lowercase and removing all non-alphabetic characters and extra whitespace.
- It then compared the cleaned text to a list of standard store kinds using a fuzzy matching algorithm.
- The function assigned the closest match to the entry, as long as the similarity score was above a defined threshold. If no close match was found, it defaulted to "unknown."

Problem 9 : Inconsistent Sales and Cost Data



The sales and cost data was stored as text, making it difficult to perform numerical calculations. Entries were highly inconsistent, including messy formats (e.g., "1.2 million", "500000", "not available") and various abbreviations and typos.

Solution :

Data Type Conversion and Unit Standardization

We created a robust cleaning function to transform the data into a usable numerical format. The function performed the following steps:

- It identified and handled a wide range of missing or invalid text entries by replacing them with a standard NaN (Not a Number) value.
- It used regular expressions to accurately extract the numerical value from each entry, regardless of surrounding text.
- It standardized all values to a single unit by recognizing keywords like "million" or by inferring the scale based on the number's magnitude, ensuring all values were consistently represented (e.g., in dollars, not millions of dollars).

Problem 10 : Messy and Missing Area Data

The dataset contained four columns for area (Store Area, Grocery Area, Frozen Area, Meat Area) that had two major issues:

1. Inconsistent Data: The values were stored as messy text, including invalid characters and various descriptions of missing data.
2. Missing Values: Many entries were simply missing, even when other related area values in the same row were present.

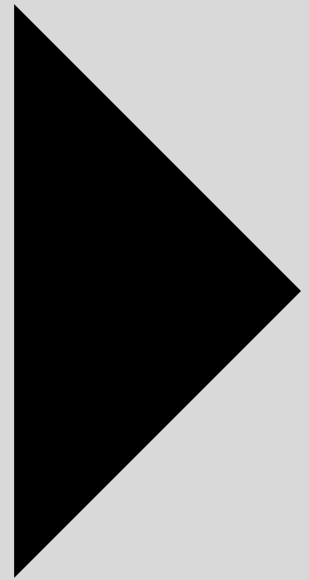
Solution :

Two-Step Area Data Imputation

We implemented a two-part solution to clean and enrich this data:

1. Numerical Extraction: We created a `clean_area` function that handled missing text and invalid entries. This function used regular expressions to robustly extract only the numerical value from the string, converting the data type from text to a clean number.
2. Imputation by Calculation: A `fill_store_area` function was then used to fill in the remaining missing values. This function leveraged the physical relationship between the columns (Store Area = Grocery Area + Frozen Area + Meat Area). If any one of the four areas was missing, it was calculated based on the values of the other three.

Problem 11 : Messy and Inconsistent Location Data



The location data was highly inconsistent, containing various forms of missing data, non-alphabetic characters, and unnecessary geographical qualifiers like "city" or "state." This made it impossible to perform accurate geographical analysis or group locations effectively.

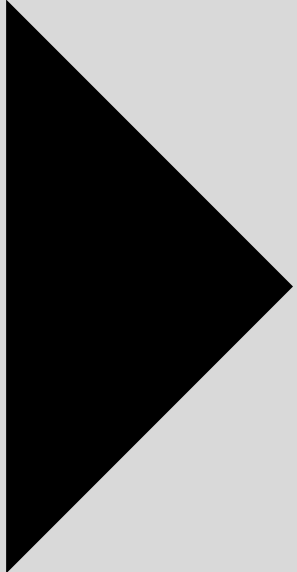
Solution :

Data Standardization and Token Removal

We developed a cleaning function to standardize the location data. The function performed the following key steps:

- It identified and handled a wide range of missing or invalid text entries, assigning them to a standard "unknown" category.
- It used regular expressions to remove all non-alphabetic characters and standardize spacing within the text.
- It then removed common geographical tokens such as "city," "state," and "region" to extract the core location name.

Problem 12 : Inconsistent Review Score Data



The 'Review Score' column contained data in various inconsistent formats, including percentages (e.g., "80%"), fractions (e.g., "4/5"), and scores on different scales (e.g., 1-10 vs. 1-5). This made it impossible to perform direct comparisons or calculate accurate averages.

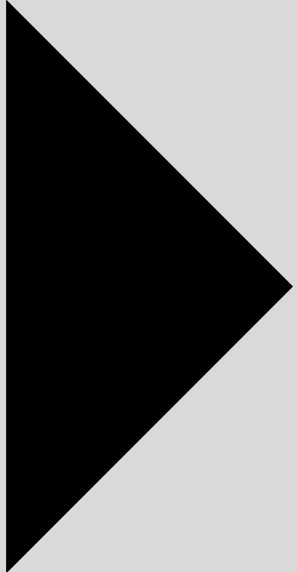
Solution :

Data Standardization and Scale Conversion

We created a comprehensive cleaning function to standardize all review scores to a single, consistent 1-5 scale. The function performed the following steps:

- It first handled a wide range of missing or invalid text entries, replacing them with a standard NaN (Not a Number) value.
- It used conditional logic to identify and correctly parse different formats, including percentages and fractions, converting them to a 5-point scale.
- It also automatically converted scores given on a 1-10 scale (e.g., "8") to the new 1-5 scale.

Problem 13 : Unreliable Cost Data



The cost data in the dataset was highly unreliable, stored as a mix of text and numerical values. Many entries contained non-numeric characters, special symbols, or a variety of text-based placeholders that represented missing or invalid data. This inconsistency made direct numerical analysis or calculation impossible.

Solution :

Data Cleaning and Type Conversion

We created a data cleaning function to transform the cost data into a clean, consistent numerical format. This function performed the following steps:

- It first identified and handled all forms of missing or invalid text entries by replacing them with a standard NaN (Not a Number) value.
- It then used a regular expression to robustly extract only the numerical characters and decimal points from the remaining text.
- Finally, it attempted to convert the cleaned numerical string into a float. This process ensured that only valid numerical data remained in the column.

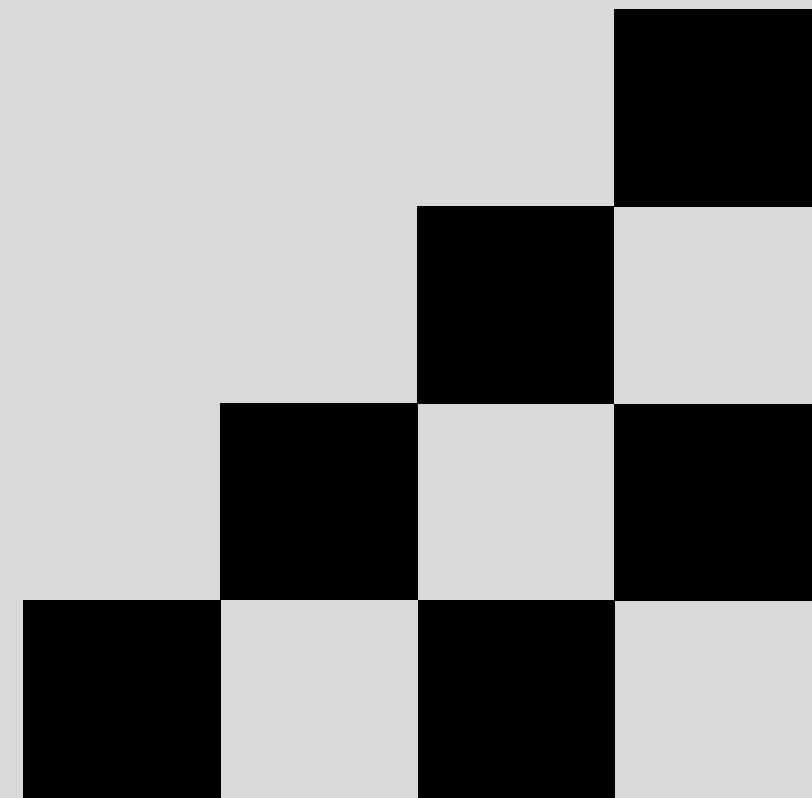
Feature Engineering

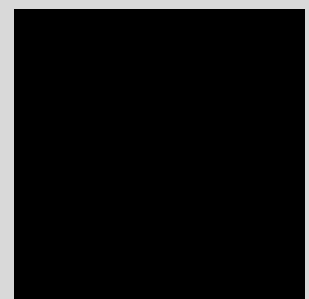
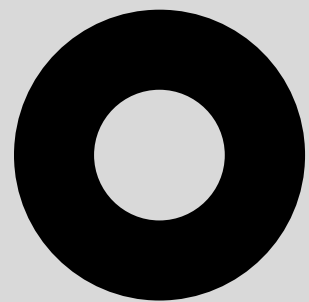
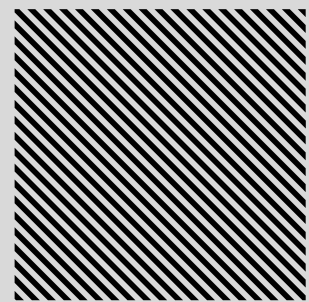
The raw dataset was limited to basic columns, lacking the complex relationships and derived metrics often needed for high-performing machine learning models. Key business insights, such as profitability, store efficiency, or customer-seller distance, were not explicitly represented as features.

Creating New Features with Feature Engineering

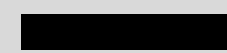
A custom FeatureEngineering class was built to generate new, more powerful features. This class, designed to fit seamlessly into a machine learning pipeline, created several new columns:

- Business Metrics: Calculated Store Profit and Profit Margin to directly represent store performance.
- Ratio Features: Derived ratios like Grocery Ratio and Frozen Ratio to measure the proportions of store areas, providing insight into store type.
- Interaction Terms: Created features like Income_x_StoreSales to capture the combined effect of income and sales.
- Geospatial Feature: Calculated the Customer_Seller_Distance using the Haversine formula, providing a crucial metric for logistics and location-based analysis.





Manual and Inconsistent Data Processing



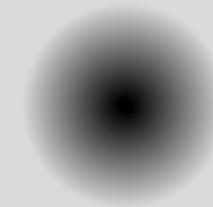
Applying each data cleaning and feature engineering function individually is cumbersome, prone to manual errors, and makes it difficult to maintain a consistent workflow. This manual process is not reproducible, making it a major risk for the integrity of the machine learning project, especially when new data needs to be processed.

Automating the Workflow with a Pipeline

To solve this, we used scikit-learn's Pipeline to create a single, automated workflow for all data preprocessing steps. The `full_pipeline` object chains all of our custom cleaning and feature engineering functions in a predefined order. This ensures that:



Every data point is processed consistently from raw input to clean output.



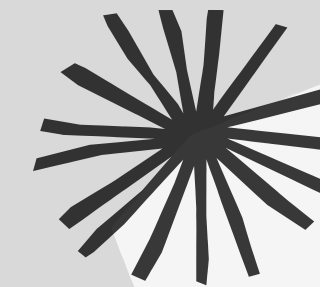
The entire process is reproducible, meaning it can be applied to new data with a single line of code, preventing manual errors.



The code is more organized, modular, and easier to read and maintain.

Problem – Diverse Feature Types

After the initial data cleaning, the dataset still contained a variety of feature types, each requiring a different preprocessing approach. Numerical columns needed scaling and imputation, while categorical columns needed to be encoded. Some columns were binary, others had few categories, and some had many. A single, generic solution would not be sufficient.



Solution :

Building Modular Preprocessing Pipelines

To address the diversity of our features, we created a modular preprocessing framework using scikit-learn pipelines. Each pipeline was specifically designed to handle a distinct type of feature, ensuring that every column received the most appropriate treatment.



Numerical Pipeline: Handled missing values with the median, capped outliers, and scaled the data.



Categorical Pipelines: We used different pipelines for each categorical type: a custom encoder for binary features, one-hot encoding for low-cardinality features, and Target Encoding for high-cardinality features to improve their predictive power.

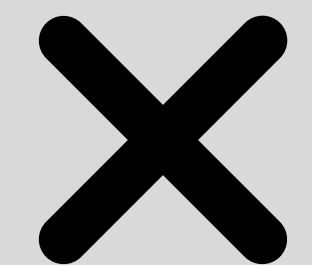
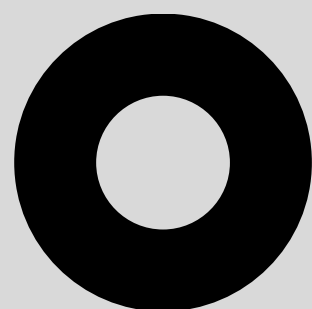
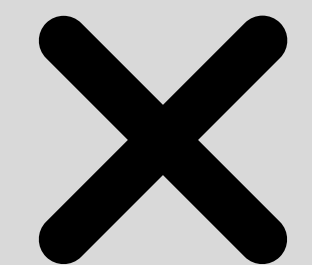
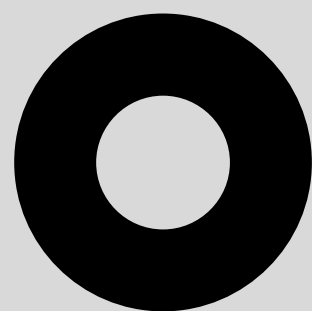


Multi-Label Pipeline: A specialized pipeline was developed to convert multi-label features into a binary matrix, a format usable by machine learning models.

The Challenge – Finding the Optimal Model

Our Solution Selecting and Optimizing the LightGBM Regressor

After preparing the data, the next critical step was to select and train a machine learning model capable of making accurate predictions. This required evaluating several algorithms to find the one with the best performance for our specific dataset.



We chose the LightGBM Regressor as our final model due to its speed and superior performance on our dataset. The model was trained on the clean, feature-engineered data from our pipeline. To maximize its accuracy and prevent overfitting, we:

Best-in-Class Performance

The finalized LightGBM model delivered excellent results, achieving a low RMSE, which confirms its high predictive power. The model's final performance is highlighted by a Root Mean Squared Error (RMSE) of 221.97, demonstrating its ability to make highly accurate predictions on unseen data.

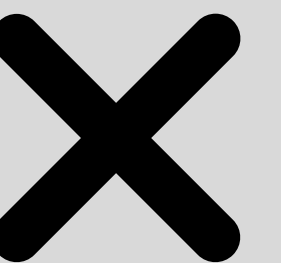
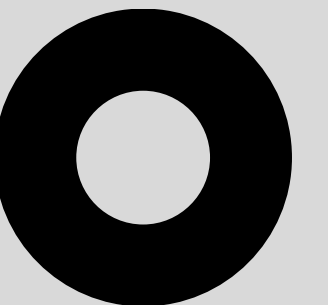
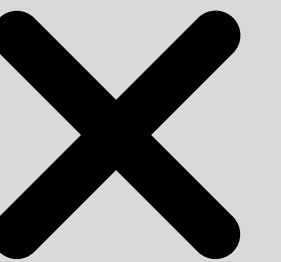
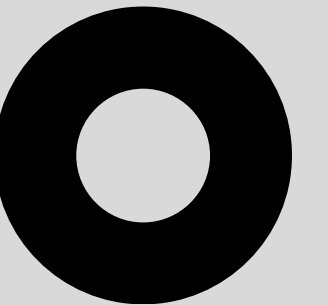
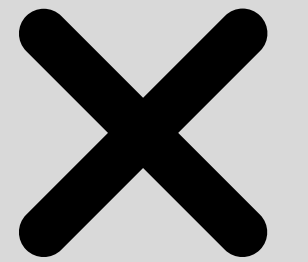
- Applied a log transformation to the target variable to handle its skewed distribution.
- Used early stopping during training, a technique that automatically halts the process when performance on a validation set stops improving.

Hyperparameter Optimization

We performed automated hyperparameter optimization to fine-tune our LightGBM model. We used the Optuna framework, which systematically searches for the best combination of parameters to achieve peak performance.

To ensure the results were reliable, we evaluated each set of parameters using K-Fold Cross-Validation, which provided a robust and stable performance estimate.

This optimization process was highly successful, leading to a significant improvement in our model's accuracy. We were able to reduce the Root Mean Squared Error (RMSE) from 221.97 to 210.49, confirming that our model is now operating at its best potential.



Final Predictions and Submission

We finalized the entire machine learning workflow by generating predictions for the unseen test dataset. We performed the following steps to ensure a robust and reliable outcome:

01

Data Preparation: We applied the same full preprocessing and feature engineering pipeline to the raw test data to ensure it was in the exact same format as the training data. This is a critical step for a consistent and reproducible process.

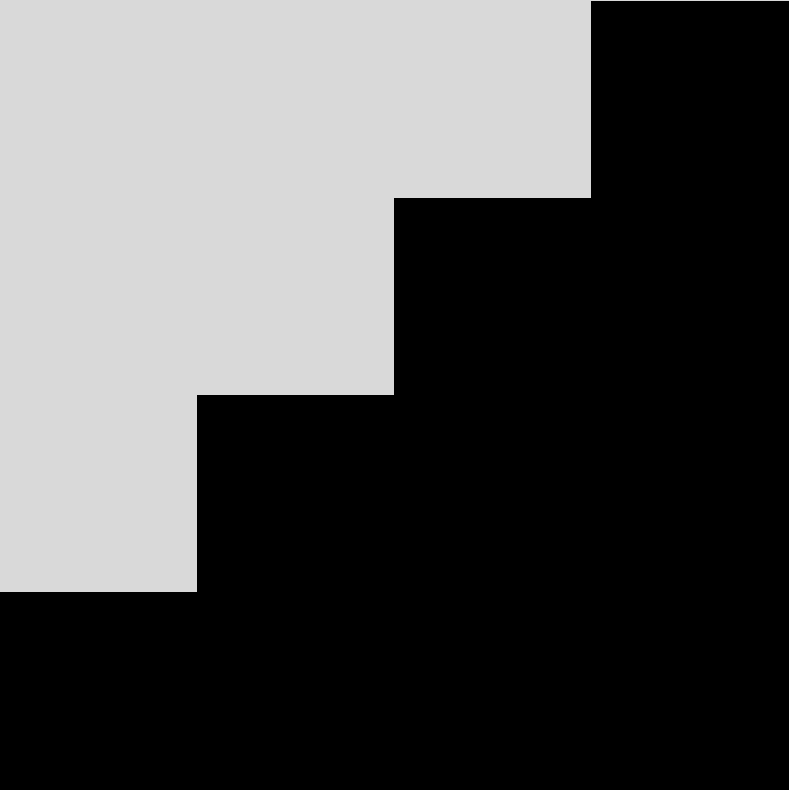

02

Final Model Training: Using the best hyperparameters identified by Optuna, we trained the final LightGBM model on the entire training dataset. This maximized the model's ability to learn from all available information.

03

Prediction Generation: The fully trained model was then used to generate predictions on the prepared test data. The predictions, which were on a log scale, were converted back to their original scale.

The final result of this process was the creation of a submission file containing the predicted cost for each entry in the test dataset. This file represents the final output of our machine learning model.



Q&A

The image features a light gray background with the text 'THANK YOU' centered. In the top-left corner, there is a black geometric shape composed of several rectangular blocks of varying sizes, creating a stepped effect. A similar black geometric shape is located in the bottom-right corner, also made of rectangular blocks of different dimensions, mirroring the style of the top-left shape.

THANK YOU