Data Engineering for Predictive Analytics

Design decisions and assumptions for end-to-end engineering project

Ona Siscart Noguer and Aina Vila Arbusà - Team 11I

Subject: Advanced Databases, BDA-GCED
Professor: Besim Bilalli
Year: 2025–2026

A Data Engineering Backbone

A.1 Business problem definition

Barcelona's rental market has experienced sustained increases in recent years due to both real-estate and contextual urban factors.

This project investigates two explanatory predictors: **Urban Safety**, considering traffic accident density as an indicator of neighborhood safety, and **Cultural Presence**, possible cause for demand amplifier.
Objective: *Predict a category of rental prices (low, medium, high) per square meter (€/m^2) across Barcelona neighborhoods based on accident frequency and cultural accessibility.*

Selected Datasets

D1. Rental Prices by Neighborhood

This dataset represents the dependent variable, providing continuous values of average rental prices per m^2 at the neighborhood level over multiple years. The data were synthetically generated to ensure full temporal coverage and statistical realism, supporting scalable modeling and year-over-year predictive analysis.

Attribute	Details
Source	Barcelona Open Data Portal
Format	JSON
Temporal Scope	2013–2025
Spatial Scope	73 neighborhoods
Update	Daily / on-demand
Target	`PerMeter`, `neigh_name`

Table 1: Summary of Rental Prices Dataset

D2. Traffic Accidents in Barcelona

Traffic accidents serve as a factor to determine the safety perception and urban mobility risk across neighborhoods in Barcelona. High accident concentration is generally correlated with noise exposure, congestion, lower walkability, and less security. These conditions negatively affect residential desirability and, therefore, rental pricing.

Attribute	Details
Source	Barcelona Open Data Portal
Format	CSV (one file per year)
Temporal Scope	2010-2024
Update Frequency	Annual
Key Fields	`Nom_barri`, `Codi_barri`, `Nom_carrer`, `Codi_carrer`, `Numero_morts`, `Numero_lesionats_lleus`, `Numero_lesionats_greus`, `Numero_victimes`, `Latitud`, `Longitud`

Table 2: Summary of Traffic Accidents Dataset

D3. Cultural Interest Points

This dataset captures points of interest throughout the city. Proximity to museums, monuments, and cultural sites tends to increase rental values due to enhanced attractiveness, commerce, and prestige. This helps to assess the valuation of each urban space.

Attribute	Details
Source	Barcelona Open Data Portal
Format	JSON
Update Frequency	Weekly
Key Fields	`neighborhood_name`, `address_name`, `latlon`

Table 3: Summary of Cultural Points Dataset

Hypotheses

1. **Urban Safety Effect:** Higher accident frequency is associated with lower rental value.
2. **Cultural Effect:** Higher proximity to cultural sites correlates positively with rental value.
3. **District Heterogeneity:** Effects differ across districts, suggesting localized market behavior.

A.2 Designing the Data Lake Architecture

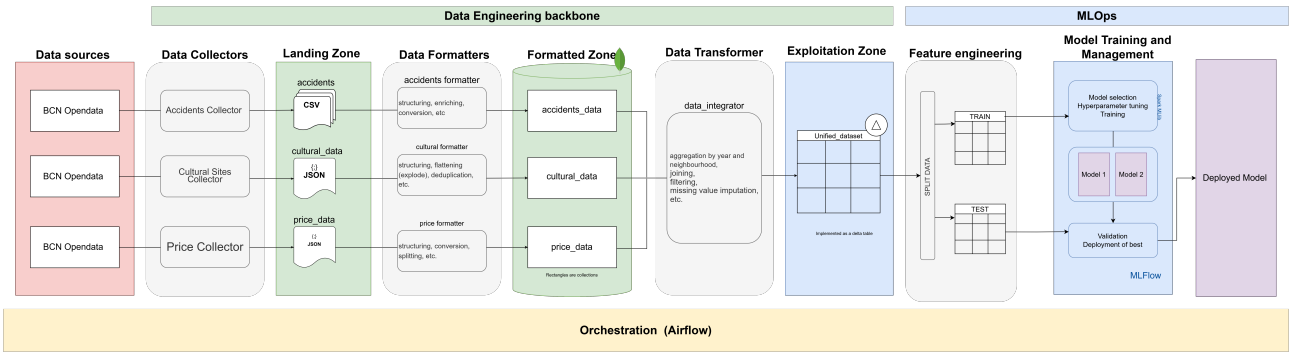

Figure 1: Data Lake Architecture design

All pipelines share a **centralized configuration layer**, implemented via `config.yaml` file, which externalizes paths, scheduling parameters, and source-specific settings to avoid hard-coding. Additionally, we added a **logging mechanism** to track execution flow and errors, enabling traceability which is crucial in a context such as this.

A.3 Data Collection Pipelines

The Landing Zone acts as the initial, unfiltered storage location for all raw data collected from diverse sources. We leverage Apache Spark to develop the data collectors capable of handling heterogeneous data formats and volumes efficiently.

Spark's distributed architecture enables parallel reading and writing of both JSON and large CSV files retrieved either from APIs or local directories, writing them to the local file system with high throughput. This prevents the single-threaded performance bottlenecks often associated with pure Python-based ingestion scripts.

The design inherently supports periodic, scheduled execution, adapting to the varying update needs of different datasets. This information is indicated by the detailed timestamps on the output file names.

- **Price Data (Daily/On-demand):** Rental price information, stored locally, is typically volatile and requires a high refresh rate to maintain relevance for real-time analysis and model retraining. Daily collection ensures the latest neighborhood-level pricing trends are captured.
- **Cultural Sites (Weekly):** Sourced from the Barcelona Open Data API, cultural sites change less frequently than financial data. Consequently, a weekly collection cadence represents an efficient compromise, providing sufficiently updated data without unnecessarily burdening the source API or internal processing resources.
- **Accidents Data (Annual):** Official traffic safety data is published annually by the Barcelona authorities. The collector automatically discovers and downloads all available yearly CSV files, ensuring a comprehensive historical record is built over time. Each year is stored as a separate file to facilitate incremental updates when new annual data becomes available.

This model ensures the pipeline is efficient, avoiding over-collection of slow-changing data, and effective, capturing fast-changing data often enough for timely insights.

A.4 Data Formatting Pipelines

The Formatted Zone serves as the unified data source for all enterprise applications. Prioritizing simplicity over complex integration, we established a **one-to-one mapping** between data sources and storage: three distinct pipelines feeding three corresponding MongoDB collections. This design supports concurrent execution and accommodates independent update frequencies (Daily, Annual, Weekly).

To ensure maintainability, all formatters adhere to three core principles. First, we use an *upsert* strategy to guarantee data **idempotency**, allowing safe re-execution without creating duplicates. Second, a JSON-based logging system enables **incremental loading** at the file level to optimize resources (except on price data, where row-level loading was possible). Finally, we **standardized** the data by normalizing column names to `snake_case`, enforcing strict typing, and populating missing fields with nulls to ensure a uniform schema ready for analysis.

Beyond these general principles, we tailored each formatter to address specific dataset challenges:

- **Price Data Formatter:** The raw data has a highly nested JSON structure. We applied **flattening** (`explode`) to reduce granularity, augmenting the flexibility and possibilities of analysis.
- **Accidents Formatter:** The dataset presents evolving schemas, which we handled in order to avoid losing historical data when merging them. Additionally, we implemented coalescing to standardize diverging column names (e.g.,`Num_postal`, `Postal_Caption`) into a single field.
- **Cultural Data Formatter:** The dataset contains semi-structured data with inconsistencies (e.g., duplicate keys). To avoid losing data, we considered a strict schema enforcement, however we decided on a **dynamic schema** using Python + Spark to prioritize data availability over a strict schema.

This foundation ensures the Formatted Zone acts as a trusted source of truth, setting a base for subsequent analysis and model development.

A.5 Exploitation Zone Design and Implementation

In accordance with the course reference architecture, we have implemented the *Exploitation Zone* using **Delta Lake**. The transformation and unification process (`TransformerPipeline`) has been designed under the following business and technical premises:

- **Target-Driven Join Strategy (Left Join):** We utilized the Prices dataset as the main (left) table for the `LEFT JOIN`. In a Supervised Learning context, the *Target* variable is essential. A row containing accident data but missing the rental price is useless for training (as the error cannot be calculated), whereas a row with a price but no accidents is valid (indicating an absence of incidents). This strategy prevents the generation of noise or rows without a target.
- **Granularity Harmonization (Roll-up):** A granularity mismatch exists between the sources: accidents have daily/point-level detail, while prices are aggregated by year/neighborhood. We applied a temporal and spatial aggregation (*Roll-up*) on the accident and cultural data to align them with our chosen primary key: (Neighborhood, Year).
- **Semantic Null Handling (Coalesce):** We opted for imputation based on business logic rather than pure statistics. During table joins, if a neighborhood lacks records in the accidents table, the system imputes a 0 (`coalesce(col, lit(0))`) instead of leaving a `NULL` value or imputing the mean. Semantically, the absence of a record in an incident dataset implies "0 accidents", a distinction that is crucial to avoid biasing the model. Additionally, if a neighborhood ID is missing, since the information is also encoded in the neighborhood name, we add a -1 to identify the value as missing easily.
- **Cultural Data Broadcasting (Static Attributes):** Open Data sources provide a current snapshot of cultural facilities without a complete historical log of openings and closures. Given the **high structural inertia** of urban landmarks (monuments and museums rarely change location or disappear), we treated this dataset as static attributes. We broadcast the current cultural density to all historical years based on the Neighborhood key. This assumes that the current cultural status of a neighborhood is a valid proxy for its recent history, avoiding the data loss that would result from enforcing strict temporal matching with incomplete metadata.

Final Dataset Structure

The execution of the pipeline results in a consolidated dataset with a granularity of **one record per neighborhood per year**. The final schema stored in Delta Lake consists of the following attributes:

Category	**Attributes (Columns)**
Identifiers	`neighborhood_name`, `(neighborhood_code)`, `year`
Target	`average_price_per_square_meter`
Safety Features	`number_of_accidents`, `number_accident_victims`, `accident_deaths`, `number_injured_lleus`, `number_injured_greu`
Cultural Features	`number_of_cultural_sites`, `average_antiquity_of_cultural_sites`

Table 4: Schema of the consolidated table in the Exploitation Zone

B Data Analysis Backbone

The objective of this stage is to transform the processed data into a predictive engine. To achieve this, we implemented a multiclass classification pipeline to categorize Barcelona neighborhoods into three price groups: Low, Medium, and High.

B.1 Model Training and Validation

Predictive models are trained using the Spark ML DataFrame-based API, which exploits Spark's Catalyst Optimizer to ensure that computations are efficiently distributed across the cluster.

Data is read directly from the Delta Lake. Since the target variable (price) is continuous and the project task requires a classification approach, label engineering was performed by discretizing prices into three classes using statistical tertiles. This approach ensures balanced classes, which is essential for classification accuracy and preventing model bias toward a specific price group.

To prepare the matrix expected by the library, a three-step transformation pipeline was implemented:

- **StringIndexer**: Categorical variables (e.g., neighborhood names) were converted into numerical indices.

- **VectorAssembler**: All features were combined into a single `features` vector.
- **StandardScaler**: Features were standardized to have zero mean and unit variance. This is essential for Logistic Regression to prevent features with larger scales from disproportionately influencing the model.

The resulting dataset was split into 70% training and 30% validation subsets using a fixed seed to ensure reproducibility.

After this, we implemented three distinct classification models to ensure a robust comparison:

- **Multinomial Logistic Regression**
- **Random Forest Classifier**
- **Decision Tree Classifier**

Each model underwent hyperparameter tuning using 3-fold cross-validation, ensuring that the hyperparameters generalize well and are not overfitted to a specific subset of data.

Model performance was evaluated on the validation set using standard classification metrics, with predictive accuracy as the primary one, which was not problematic as classes are balanced. The models were then ranked based on their accuracy, and the system automatically identifies and deploys the best-performing model.

B.2 Model Management

MLflow serves as the centralized management layer for the entire analysis pipeline, providing traceability, versioning, and governance.

Every training run was logged as part of an MLflow experiment, recording the model type, hyperparameters, evaluation metrics, and artifacts. This enables easy comparison across runs and allows models to be reloaded.

We utilized the MLflow Model Registry to manage the deployed predictors. Each execution of the pipeline registers a new model version, ensuring centralized storage and systematic version control.

The best-performing model is explicitly tagged in the registry, linking it to the exact training data, parameters, and metrics; and so providing a complete audit trail and supporting reproducibility.

B.3 ML Results

Figure 2: Models' Performance Comparison

Here we present a visual summary of the models' performance with three complementary visualizations.

The bar chart shows that the Decision Tree achieves the highest validation accuracy, outperforming the others with a substantial margin.

While accuracy is critical, model selection must also consider computational efficiency. The scatter plot reveals that the Decision Tree achieves both the highest accuracy and the shortest training time. In contrast, the Random Forest requires nearly double the training time for lower accuracy, and Logistic Regression is the least efficient.

Although accuracy is important, it does not fully describe model behavior. To evaluate if a model's superiority is consistent across all metrics, we use the parallel coordinates plot on the right, which reveals that the Decision Tree consistently outperforms the other models with no observable trade-offs.

In conclusion, based on the observations mentioned, the Decision Tree classifier is identified as the optimal model for this classification task.