Polythecnic University of Catalonia (UPC)

Bachelor's Degree in Artificial Intelligence (GIA)

Advanced Databases (BDA)

Large-Scale Data Engineering for AI

Roger Baiges **Adrià Flores** **Cai Selvas**

April 13, 2025

Abstract

This project was developed as part of the subject Advanced Databases (*Bases de Dades Avançades – BDA*) of the Bachelor's Degree in Artificial Intelligence (*Grau en Intel·ligència Artificial – GIA*) at the Polytechnic University of Catalonia (*Universitat Politècnica de Catalunya – UPC*). It consists of the implementation of a complete data engineering pipeline applied to socio-economic data from Catalonia.

The objective is to integrate multiple data sources and to generate a clean, standardized dataset that can support basic analysis tasks such as visualization and predictive modeling. This work aligns with the course requirements, emphasizing data quality, reproducibility, and the application of DataOps principles in a practical context.

The length of this document fits the maximum of 5 pages (only taking into account the main text pages, excluding the cover, abstract, table of contents, conclusions and references) mentioned in the project statement to ensure that the explanation is concise. All the plots mentioned are not shown to avoid occupying too much space, but it's locations within the files provided for the delivery are provided.

Contents

1 Introduction

This project implements an end-to-end data engineering pipeline designed for scalability and robustness, addressing common pitfalls in ad-hoc data science workflows. The primary goal is to process, clean, and integrate diverse datasets related to the socio-economic landscape of Catalonia, ultimately enabling advanced analytical tasks.

We focus on combining demographic and economic indicators from Idescat (Catalan Statistics Institute) with rental price data and household income data for Catalan municipalities. These datasets were chosen for their potential to reveal relationships between local characteristics, economic conditions, and housing costs. The engineered pipeline transforms this raw data through standardized zones (Landing, Formatted, Trusted) using Apache Spark and Delta Lake, culminating in an integrated dataset within the Exploitation Zone ready for analysis.

Initial analytical objectives explored using this integrated data include:

- **Visualization:** Creating choropleth maps of Catalonia to visualize the geographic distribution of metrics like average rental price or income levels, potentially highlighting spatial patterns or correlations.
- **Predictive Modeling:** Developing a model to predict future rental prices at the municipal level based on historical price trends and relevant socio-economic indicators from the integrated dataset.

This structured approach ensures data quality, reproducibility, and provides a reliable foundation for these and potentially other future data analysis tasks.

2 Landing Zone

The Landing Zone serves as the initial ingestion point. Raw data from Idescat, rental price sources, and income datasets are collected via dedicated Python scripts and stored efficiently Parquet or JSON files on the local file system. Parquet is chosen for its optimized columnar storage and direct compatibility with downstream Spark processing, while keeping the landing zone simple and focused purely on data acquisition without transformations.

2.1 Idescat Dataset

The Idescat EMEX API [Idescat(2019)] (*El municipi en xifres*) was chosen as a primary data source for its rich offering of fundamental demographic and socio-economic indicators at the municipal level, crucial for analyzing regional housing and income patterns. This API provides data in a long format (one row per indicator/municipality/year), unlike conventional fixed-column datasets. Therefore, a specific subset of indicators relevant to our analysis and exhibiting high data availability (low missing values across municipalities) was selected. Key indicators ingested include total population, demographic breakdowns (gender, births, nationality), geographic references (surface area, density, coordinates), total unemployment, and counts of primary dwellings, among others, effectively constructing a tailored dataset from the available options.

Data collection utilizes a Python script employing parallel API calls (`ThreadPoolExecutor`) to efficiently fetch the selected indicators for all Catalan municipalities. The script extracts indicator values, reference years, geographic names (municipality, comarca), and source timestamps. Before saving, minimal preprocessing using Pandas structures the data and attempts basic type conversion for numeric and timestamp columns.

The aggregated raw data is stored as a single Parquet file (`idescat.parquet`) in the Landing Zone. This columnar format ensures storage efficiency and seamless compatibility with Apache Spark for subsequent processing in the Formatted Zone, while adhering to the Landing Zone principle of minimal transformation. The data remains in its original long format within this file.

This API only provides the most recent values available for each indicator, and each indicator has different update periods and dates. Thus, all values need to be fetched each time that the pipeline is executed to ensure that everything is up to date.

2.2 Lloguer Dataset

To capture rental market trends, data was sourced from the Generalitat de Catalunya's open data portal [Lloguer(2018)] via its API endpoint (`https://analisi.transparenciacatalunya.cat/resource/qww9-bvhh.json`). This dataset provides monthly average rent (`renda`), number of contracts (`habitatges`), and rent price brackets (`tram_preus`) aggregated by various territorial levels (municipality, district, etc.) and time periods (`periode`, `any`).

A dedicated Python script (`LloguerLandingZone`) handles the data collection. Due to the API's pagination, the script iteratively fetches data in chunks using the `requests` library, incorporating appropriate delays to respect API limits. The raw JSON responses from each page are aggregated into a Python list. This list of dictionaries is then efficiently converted into a Pandas DataFrame and saved directly to a single Parquet file (`lloguer.parquet`) using the PyArrow engine. Parquet was chosen over JSON for the landing storage due to its superior compression, columnar efficiency, and native integration with Apache Spark, aligning with the project's downstream processing requirements while keeping the landing process focused solely on acquisition.

2.3 RFDBC Dataset

The RFDBC dataset (*Renda Familiar Disponible Bruta Territorial*) [RFDBC(2024)] from Idescat provides annual estimates of gross disposable income per inhabitant for all Catalan municipalities with more than 5000 residents, covering the period from 2010 to 2021 (as of its latest update in April 2024). Its API endpoint [1] returns data in a complex and deeply nested JSON format. Consequently, the `RFDBCLandingZone` script uses the `requests` library to fetch this data and stores it in raw JSON format at its designated path in the Landing Zone.

We attempted to store the dataset as a Parquet file to ensure better integration with PySpark in the next zone (Formatted Zone). However, due to the nested JSON structure of the original data, saving it directly as Parquet would require transformations that conflict with the Landing Zone principle of minimal processing. Therefore, these transformations are deferred to the Formatted Zone, where the data will be normalized and restructured accordingly.

In this dataset we have not found a way of fetching only new data (excluding data previously fetched), so the request asks for all the available data to ensure that it gets de most recent values.

3 Formatted Zone

In the Formatted Zone, Apache Spark reads the raw Parquet files and applies predefined schemas to ensure consistent naming and correct data types (syntactic homogenization) for each dataset individually. The results are stored as separate Delta Lake tables. Delta Lake is utilized here, as required by the project guidelines allowing for robust storage options, because it builds upon Parquet while adding crucial database-like features such as ACID transactions and schema management, integrating seamlessly with the mandated Spark processing framework.

3.1 Idescat Dataset

Processing for the Idescat dataset begins by reading the `idescat.parquet` file from the Landing Zone using Apache Spark. An explicit read schema is defined to correctly interpret the data types as written by Pandas in the previous stage (e.g., handling potential Long types for timestamps). A key transformation performed in this zone is the normalization of the `municipality_id` using Spark functions to ensure a consistent 5-digit format, addressing potential variations (e.g., 6-digit codes with a check digit) found in the source API.

Subsequently, the data is rigorously conformed to the predefined target schema, defined using Spark's `StructType` (method `_define_target_schema`). This involves selecting the necessary columns and explicitly casting each one to its final, correct data type (e.g., 'reference_year' to Integer, 'municipality_value' to Double, and 'source_update_timestamp' correctly converted to Timestamp). The resulting structured DataFrame, which critically maintains the original long format (one row per indicator/municipality/year), is then saved as a Delta Lake table (located at `./data/formatted/idescat`), ready for quality checks in the Trusted Zone.

3.2 Lloguer Dataset

The `lloguer.parquet` file from the Landing Zone is read using Spark. Similar to the Idescat processing, a target schema (`LloguerFormattedZone._define_target_schema`) is defined with appropriate `StructType` definitions (e.g., any and habitatges as IntegerType, renda as DoubleType, geographic identifiers as StringType). Spark's schema inference on Parquet often reads numeric columns saved by Pandas without explicit typing as strings; therefore, an explicit casting step is applied using Spark SQL functions (`F.col().cast()`) to ensure each column conforms to the target schema's data type. No complex structural transformations occur here; the table retains its granularity (one row per territory/year/period/price bracket). The resulting DataFrame is saved as a Delta Lake table (`./data/formatted/lloguer`), benefiting from Delta's reliability features for the subsequent Trusted Zone operations. The optional `verbose` flag triggers an inspection of the output Delta table schema and sample data upon completion.

3.3 RFDBC Dataset

The JSON file from the Landing Zone is loaded by the `RFDBCFormattedZone` class, which transforms it into a flat list of records and converts the list into a PySpark DataFrame using a predefined schema. Precisely, this conversion is done inside the `_transform_data()` method using the dimensions and values provided in the metadata of the JSON file.

After this method, the columns are renamed and converted to lowercase in order to follow the same naming convention as the other datasets. Finally, the DataFrame is saved to the target path using the Delta Lake format due to its high performance with Apache Spark and its guarantee of data integrity and fault tolerance.

[1] `https://api.idescat.cat/taules/v2/rfdbc/13301/14148/mun/data`

4 Trusted Zone

The Trusted Zone focuses on improving data quality within each dataset; Spark processes apply validation rules, potentially expressed using concepts like denial constraints to identify unacceptable data combinations, handle inconsistencies or basic missing values, and remove duplicates from the Idescat, rental, and income Delta tables independently. This stage leverages Delta Lake's reliability for these cleaning operations before data integration.

4.1 Idescat Dataset

For the Idescat data, the process starts by reading the formatted Delta table (`./data/formatted/idescat`) using Spark. A series of data quality rules, conceptually acting as denial constraints, are then applied sequentially to filter out invalid records. These include:

- Ensuring key columns (`municipality_id`, `indicator_id`, `reference_year`) are not null.
- Validating that the `reference_year` falls within a plausible range (e.g., 2000-2025).
- Verifying indicator values based on their type; for instance, ensuring population counts (`f171`), births (`f187`), surface area (`f261`), and facility counts (`f191`, `f270`, etc.) are non-negative, and that geographical coordinates (`f328`, `f329`) fall within expected bounds for Catalonia.

Following these filtering steps, duplicate records are removed based on the logical key combination of `municipality_id`, `indicator_id`, and `reference_year` to ensure data uniqueness. The resulting cleaned and deduplicated DataFrame, still maintaining the long format, is then written as a new Delta Lake table to the Trusted Zone (`./data/trusted/idescat`).

4.2 Lloguer Dataset

The trusted processing for the Lloguer dataset begins by reading the formatted Delta table (`./data/formatted/lloguer`). The `LloguerTrustedZone` script applies several data quality filters implemented as Spark DataFrame operations, serving as practical denial constraints:

- **Null Checks:** Rows with null values in essential identifying columns (`ambit_territorial`, `codi_territorial`, `nom_territori`, `any`, `periode`, `tram_preus`) are removed, as these are necessary for meaningful records.
- **Year Validation:** Records where the `any` column falls outside a realistic range (e.g., 2005 to current year + 1) are filtered out.
- **Value Validation:** Rows where the number of contracts (`habitatges`) is negative, or where the average rent (`renda`) is present but non-positive, are removed as these represent impossible scenarios.

An examination step calculates the number of unique municipality-year combinations remaining after filtering to understand the effective annual granularity. A check is also performed to identify if duplicate rows exist based on the full key (including periode and tram_preus) but have differing habitatges or renda values; none were found in this dataset, validating the subsequent deduplication step. Finally, exact duplicate rows based on the key [`ambit_territorial`, `nom_territori`, `codi_territorial`, `any`, `periode`, `tram_preus`] are removed using `dropDuplicates`. The cleaned data, retaining the original schema, is saved to the trusted Delta table (`./data/trusted/lloguer`).

4.3 RFDBC Dataset

For this dataset, after the Delta table is read from the Formatted Zone, numerous transformations and data cleaning steps are performed inside the `RFDBCTrustedZone` class in order to improve the quality of the dataset.

First of all, the method `remove_columns()` removes some irrelevant columns like `concept_code` (always contains the same useless value) and `year_code` (it contains the exact same values as `year_label`, introducing unnecessary redundancy). Secondly, the method `modify_columns()` renames some columns to fit the naming convention of the project and then removes the last digit of the values of `municipality_id` to match the standard 5-digit administrative coding system. Finally, the `remove_rows()` removes the duplicates and applies some Denial Constraints (DC), such as ensuring that all years are in the range [2000, 2025], all values of gross income (`vale` column) are positive and all values of `municipality_id` have exactly 5 digits.

Once this is done, the cleaned dataset is stored inside the trusted directory using the same Delta Lake format.

5 Exploitation Zone

The Exploitation Zone is where the cleaned, trusted datasets are integrated and transformed into a format suitable for the specific downstream analysis pipelines. Recognizing that different analyses might require different granularities or features, the initial decision was to create a single, consolidated annual dataset as a versatile starting point for both visualization and prediction

baselines. This integration is performed using PySpark within the `ExploitationZone` class, leveraging Spark's distributed processing for efficient joins and aggregations, with the output stored as a final Delta Lake table.

The process involves several key steps:

1. **Loading Trusted Data:** The script loads the Lloguer, RFDBC, and Idescat Delta tables from the Trusted Zone.
2. **Processing Lloguer:** The trusted Lloguer data is filtered for municipal-level records (`ambit_territorial == 'Municipi'`). Crucially, because the original `renda` column had significant missingness, an imputation step is performed using the non-missing `tram_preus` (rent bracket) column. A representative value (typically the midpoint) is assigned to `renda` based on its corresponding bracket, significantly increasing data completeness for the rent metric. The data is then aggregated annually per municipality (`codi_territorial`, `any`), calculating the total number of contracts (`total_annual_contracts`) and the weighted average of the (imputed) monthly rent (`avg_monthly_rent_eur`), weighted by the number of contracts.
3. **Processing RFDBC (Salary):** The trusted RFDBC data is filtered to isolate the per capita income indicator (`PER_CAPITA_EUR`). Relevant columns (municipality_id, municipality_name, year, value) are selected, and the year is cast to Integer (renamed any) and value is renamed to `salary_per_capita_eur`.
4. **Processing Idescat Indicators:** To provide relevant context without introducing time-varying complexity for every indicator, the script identifies the latest available value for each selected indicator (`population`, `density_pop_km2`, `unemployment_total_avg`, etc.) for every municipality using a Spark Window function partitioned by municipality and indicator, ordered by year descending. These latest values (which might be from different reference years for different indicators/municipalities) are then pivoted to create one row per municipality with indicators as columns. This provides the most recent static context available for each municipality.
5. **Joining Data:** The annually aggregated Lloguer data and the annual Salary data are joined using a `full_outer` join on `municipality_id` and `any` to preserve all municipality-year combinations present in either dataset. The resulting table is then `left` joined with the pivoted Idescat indicators based only on `municipality_id`, adding the latest indicator values to each annual record. Column names are coalesced where necessary.
6. **Final Output:** The consolidated DataFrame, containing annual rent, salary, and the latest static indicators per municipality, is saved as a Delta Lake table (`./data/exploitation/municipal_annual`). An optional inspection step allows verification of the final schema and data.

This approach yields a single, comprehensive table suitable for analysis, with the understanding that the Idescat indicators represent the latest known static values rather than varying annually alongside rent and salary.

6 Data Analysis Pipeline 1: Visualizing Housing Affordability in Catalonia

To gain deeper insights into regional housing market dynamics, the first Data Analysis Pipeline was developed to visualize housing affordability throughout Catalonia. This approach was chosen because spatial context is critical for understanding variations in affordability, revealing geographic patterns and potential socio-economic correlations that tabular data alone cannot convey effectively. The pipeline leverages the unified dataset created in the Exploitation Zone, showcasing the value derived from the preceding data engineering stages by enabling such geographically-aware analysis.

The pipeline operates as follows:

1. **Data Consumption:** It begins by querying the primary integrated table within the Exploitation Zone.
2. **Data Selection & Filtering:** Relevant data for the all the years is extracted, specifically selecting the municipality identifier (e.g., `municipality_id`) and the chosen average rental price metric (`avg_monthly_rent_eur`) and the average salary per capita (`salary_per_capita`).
3. **Housing Affordability Calculation:** In order to calculate if a town for a given year is affordable in order to rent a house we calculated how much percentage of salary is spent in renting an average property.
4. **Geospatial Integration:** This filtered tabular data is then merged with a separate geospatial dataset containing the polygon boundaries for Catalan municipalities (shapefile). This critical step, performed using the `geopandas` library, links the statistical rental data to its corresponding geographic location.
5. **Map Generation:** Finally, a choropleth map is generated using visualization libraries such as `matplotlib` or `seaborn`. In this map, each municipality polygon is coloured according to its associated rental price level, allowing for visual interpretation of the geographic distribution.

The resulting map [2] visually represents the output of this analysis pipeline. Despite potential data gaps for some municipalities (areas appearing grey due to missing source data, as RFDBC only contains municipalities with more than 5000 inhabitants), a distinct concentration of higher rental prices (redder colours) is evident around the Barcelona metropolitan area, while more remote, mountainous regions generally exhibit lower rental prices (greener colours).

[2] Can be found in `./data/analysis/visualizer`

7 Data Analysis Pipeline 2: Predicting Annual Average Rent

The second analysis pipeline focuses on predicting future housing costs, specifically the average monthly rent per municipality for the next year (`avg_monthly_rent_eur`). This regression task leverages the consolidated dataset from the Exploitation Zone, aiming to build a baseline predictive model using historical trends and socio-economic indicators. LightGBM was chosen as the modeling algorithm due to its strong performance on tabular data, efficiency, and native handling of missing values and categorical features, making it suitable for a robust baseline.

The pipeline, encapsulated in the `DataAnalysisPipeline` class, follows standard machine learning steps:

1. **Data Loading:** The consolidated annual municipal data is loaded from the Exploitation Zone Delta table using Spark and then converted to a Pandas DataFrame for processing with Scikit-learn and LightGBM. This conversion is feasible given the dataset size after annual aggregation.
2. **Feature Engineering:** To capture temporal dynamics crucial for prediction, several features are engineered using Pandas:
   - **Lag Features:** Values from previous years (lag 1, 2, and 3) are created for the target variable (rent), salary, and contract counts. This allows the model to learn from recent historical values.
   - **Difference Features:** The change in rent between year T-1 and T-2 (`target_diff1`) is calculated to represent recent momentum. This was specifically calculated using past lags to avoid data leakage from the target year.
   - **Rolling Features:** A 3-year rolling mean and standard deviation of the lag-1 rent are calculated to provide smoothed trend and volatility information.

   Rows where the target or essential lag features (necessary for calculating differences) are null (primarily affecting the earliest years in the dataset) are dropped.
3. **Data Splitting:** A strict time-based split is performed. All data before the final year (2024) is designated for training/-validation, while data only from the final year (2024) constitutes the test set. This critically ensures that the model is evaluated on its ability to predict genuinely future data, preventing temporal data leakage.
4. **Preprocessing:**
   - **Categorical Handling:** The `comarca_name` column is converted to the Pandas 'category' dtype, allowing LightGBM to handle it efficiently using its internal algorithms.
   - **Scaling:** Numeric features (excluding the year column) are scaled using Scikit-learn's `StandardScaler`, fitted only on the training data to prevent test set information leakage, and then applied to both train and test sets. The fitted scaler is saved for potential future use.
   - **Missing Values:** No explicit imputation is performed for remaining missing values (e.g., in salary or lagged features); LightGBM's ability to handle NaNs internally is leveraged for simplicity and robustness in this baseline.
5. **Model Training:** An `LGBMRegressor` model is trained. To optimize the number of boosting rounds and prevent overfitting, early stopping is employed. The training data is further split temporally, using the penultimate year (2023) as a validation set to monitor performance during training. The model stops training when performance on the validation set ceases to improve for 50 consecutive rounds. Subsequently, a final model is retrained on the entire training dataset (up to 2023) using the optimal number of iterations identified by early stopping. This final model is saved using `joblib`.
6. **Evaluation & Visualization:** The trained model predicts rent for the test set (year 2024). Performance is evaluated using standard regression metrics: Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), R-squared (R^2), and Mean Absolute Percentage Error (MAPE). Diagnostic plots are generated and saved: Feature Importance, Prediction vs Actual, and Residuals vs Predicted.

7.1 Results

The baseline LightGBM model achieved the following performance on the test set (predicting 2024 rent): MAE=72.21€; RMSE=104.72€; MAPE: 11.61%; R^2=0.7174.

An R^2 of approximately 0.72 indicates that the model explains about 72% of the variance in the 2024 rental prices based on the historical data and indicators provided. The MAE suggests an average prediction error of around 72€. The feature importance plot [3] reveals that, even after correcting for leakage, the lag-1 rent (`avg_monthly_rent_eur_lag1`) remains a highly influential predictor, highlighting the strong persistence in rental prices. Other important features include salary, contract volume, geographic coordinates, and the year itself. The Prediction vs Actual plot [4] shows predictions generally following the ideal line but with notable scatter, particularly at higher rent values. The Residual plot [5] indicates that while residuals are mostly centered around zero, there might be a slight tendency for larger errors at higher predicted rents (heteroscedasticity), suggesting areas for potential model improvement. Overall, the baseline provides a reasonable predictive capability, heavily leveraging the previous year's rent, while also incorporating other socio-economic factors. Further improvements could involve more advanced feature engineering, hyperparameter tuning, or exploring alternative models.

[3] Can be found in `./data/analysis/model/feature_importance.png`
[4] Can be found in `./data/analysis/model/prediction_vs_actual.png`
[5] Can be found in `./data/analysis/model/residuals_vs_predicted.png`

8 Conclusions

This project developed a scalable and structured data pipeline for integrating and analyzing socio-economic and housing data in Catalonia. Apache Spark was used for its distributed processing capabilities, allowing efficient handling of large, multi-source datasets. Parquet was selected in the Landing Zone for its optimized columnar storage, which reduces disk usage and accelerates data loading in Spark.

Delta Lake was used in the Formatted, Trusted, and Exploitation Zones to add transactional reliability and schema enforcement on top of Parquet, ensuring data consistency during transformations and quality checks.

The use of a zone-based architecture (Landing $\rightarrow$ Formatted $\rightarrow$ Trusted $\rightarrow$ Exploitation) ensured clean separation between ingestion, formatting, validation, and integration stages. This structure supported a clear and reproducible workflow, enabling reliable downstream analysis such as geospatial visualizations and rental price prediction.

The chosen technologies and design decisions provided the necessary performance and robustness to process complex regional datasets while maintaining clarity and data quality throughout the pipeline.

9 References

[Lloguer(2018)] Lloguer 2018 : *Preu mitjà del lloguer d'habitatges per municipi.* 2018. – URL `https://analisi.transparenciacatalunya.cat/Habitatge/Preu-mitj-del-lloguer-d-habitatges-per-municipi/qww9-bvhh/about_data`

[Idescat(2019)] Idescat 2019 : *Idescat. Àrea de desenvolupadors API. El municipi en xifres.* Mar 2019. – URL `https://www.idescat.cat/dev/api/emex/`

[RFDBC(2024)] RFDBC 2024 : *Renda Familiar Disponible Bruta Territorial (RFDBC).* Apr 2024. – URL `https://www.idescat.cat/pub/?id=rfdbc`