Report 1: Data Management Backbone

Group 12 : Walter J. Troiani Vargas, Joan Acero Pousa

Table of contents

The project

The data management backbone was developed within the integrated environment of Google Drive and Google Colab, leveraging the collaborative and cloud-based features of these tools to streamline data operations. After finalising the development and ensuring the robustness of the data management processes, the operational components were seamlessly transferred to a GitHub repository for version control and easier accessibility.

- To access the Google Drive notebooks, please use the following link: ADSDB.
- For a comprehensive view of the GitHub repository, visit the repository.

For further information on installing and using the repository locally or using Docker, please refer to the README.md located inside the project directory, which contains an extensive installation and usage guide.

Context

This project focuses on analyzing Barcelona's housing market and socioeconomic landscape. Various datasets will be employed, including rental prices and quantities, gross and net incomes, and economic indicators such as the Gini coefficient and P80/P20 distribution indexes across different districts and neighbourhoods of the city.

At the heart of our analysis lies a question that resonates with many residents: “Is it just me, or are rental prices skyrocketing every year?”. Our goal is to uncover hidden trends and patterns within the city's economy and answer a range of critical questions, such as:

- How do housing prices in different neighbourhoods correlate with the incomes of their residents?
- What percentage of income do people allocate to housing and necessities to make ends meet?
- Are there significant disparities and wealth inequalities present in the city?
- Which neighbourhoods are experiencing trends such as gentrification or abandonment?
- What are the projections for housing prices in the coming years?
- Which districts are seeing exponential increases in rental prices?

To answer all of these questions and gather even more deep insights that are yet to be discovered, we relied on the official public state data sets, given that data of this kind given from particular/private data sources are unreliable and could even be manipulated with misinformation.

This project has involved up to 8 data sets, each containing two versions of 2 different years, 6 of them being from the first source, “Opendata”, and the rest belonging to “Portaldades”. Below, we provide a summary of the data sources and data sets:

OpenData: A new source of data provided by Barcelona's town hall from 2012, contains several open-source datasets available to everybody, ranging from economic data to population and city infrastructure information. The datasets we gathered are:

- **Renda Bruta:** This dataset provides information on the gross average income by census section across different neighbourhoods and districts. The versions used in this project are from 2020 and 2021, allowing for temporal comparisons and insights into income changes at the neighbourhood level.
- **Renda Disponible:** Focused on average disposable income by census section, this dataset mirrors the structure of Renda Bruta, covering 2020 and 2021. It highlights income available after taxes and the main living expenses.
- **Renda Neta:** This dataset provides average net income information at the census section level, as well as for 2020 and 2021. It reveals income levels after all necessary deductions, providing a clearer view of economic standing by neighbourhood.
- **Demografia:** This dataset includes demographic details by census section, covering factors like average age, gender distribution, and population percentages. It helps in understanding the population structure of each neighbourhood and district, which is essential for correlating with socio-economic data. Also, 2020 and 2021 versions were employed.
- **Index Gini:** A pre-computed Gini index for each census section, measuring income inequality within neighbourhoods. The Gini index supports a detailed analysis of wealth distribution disparities across the city's districts. Also, 2020 and 2021 versions were employed.
- **Distribució P80/P20:** This dataset contains the P80/P20 income ratio by census section, pre-computed to reflect income inequality. The P80/P20 ratio contrasts the income of the wealthiest 20% with the poorest 20%, highlighting wealth gaps. Also, 2020 and 2021 versions were employed.

Portaldades: This historical data source, also provided by the Barcelona Town Hall (Oficina Municipal de Dades), includes datasets with a more extended time context.

- **Lloguers Recompte:** This dataset aggregates rental data from 2000 to 2024, detailing the number of rental units by neighbourhood, district, and city level. For this project, we focus on neighbourhood and district data.
- **Lloguers Preus:** This dataset contains average rental prices by neighbourhood and district over the same timeframe as Lloguers Recompte. Like the rental count data, these tables provide both absolute values and an artificial incremental version for educational purposes.

Despite the lack of cohesion between some attributes like "District", standardisation can be performed to solve this problem. More importantly, it's clear that the data contains almost no null values, making this aspect of management more straightforward. These datasets can be joined by two attributes, mainly by time or space (District, Neighborhood...); however, they come in different levels of granularity.

Data Management Backbone

1. Data Ingestion

This initial section clarifies the origins of the datasets utilised in the project. We selected two primary data sources: Portal de Dades Barcelona and OpenData Barcelona. As mentioned in the previous section, we systematically downloaded multiple datasets from both sources for each year (2020 and 2021), each focusing on a single attribute. These individual datasets were then merged to create comprehensive datasets for each source and year, encompassing multiple attributes.

In the data ingestion code, we detail this process, illustrating how the datasets are transformed and refined. Ultimately, the final datasets are transferred to the temporal landing zone, where they will serve as the foundational datasets for our project.

This is not more than a previous step before starting the project in order to ensure that the data with which we will work is valuable enough. Therefore, while this preprocessing of the data is included in the deliverable, the project starts in the temporal landing zone, where we simulate that the initial raw data is found.

Going through the data ingestion preprocessing provides valuable data for the development of the whole project and ensures comprehension. However, a better practice, yet maybe less comprehensive, would be to upload all the individual CSV files, execute the entire pipeline up to the exploitation zone, and,º in the exploitation zone, perform the joins to prepare the data well for the analysts. Since we were allowed to choose the initial data, we decided to do it in the way mentioned in order to understand better the execution of the pipeline, which we think is the main focus of this project.

2. Landing Zone

The Landing Zone is structured into two subzones: the temporal and the persistent landing zones. Within the persistent landing zone directory, we have implemented a system for organising the raw files collected in the temporal landing zone into distinct directories based on their respective data sources. Each file is renamed according to the following convention: “data_source_year”, where the year expresses when the data in the dataset was collected.

We chose this naming convention to manage our datasets effectively, considering that, in our particular case, data is ingested annually. This way, no duplicate names can be created, and no redundant information is included in the file names.

An alternative approach could have involved using the naming convention “file_name_timestamp,” which would also avoid generating duplicate file names. However, our chosen implementation explicitly includes the data source in the file name, which provides immediate clarity regarding the origin of each dataset. This is particularly advantageous when extracting tables from the database in the formatted zone, as it allows one to quickly

understand the complete context of the data without needing to parse through the landing zone directories to identify the sources of individual files.

3. Formatted Zone

The formatted zone unifies the formats of all the system files, preventing us from having to manage data that is structured differently.

Following the recommendations of the project statement, we used DuckDB as an analytical database to represent our data in the relational model. Using it, we created a database and generated a table per data source version, since this is the safest method to protect against potential changes in future data ingested.

The main downside we detect in this zone is that, in a real case, unifying data in the relational model with a table per dataset version may restrict flexibility for handling non-relational data formats, add storage overhead, and complicate schema evolution. However, since we work in a simplified case and assume the relational model as the canonical form, this should be fine.

Moreover, we included a profiling notebook that deeply analyses the database data. Please refer to the "db_profiling.ipynb" notebook section "Main Insights" to see the conclusions we extracted from the deep profiling executed in the notebook section "3. Profiling of each table." There, we explain all the relevant attributes in the tables, analyse their distributions and correlations between them, and mention which ones are worth keeping.

4. Trusted Zone

In this section, we justify our transition from the formatted zone to the trusted zone. The trusted zone's core element is a database containing a single table per dataset.

We concatenated one dataset with the other to merge the two versions of each dataset. We chose this method of combining both datasets even though the year of ingestion that was written in the name of each table is lost since the tables themselves contain a column that indicates the year they belong to, and our analysis is yearly based. Therefore, we retain all valuable information.

Additionally, the trusted zone directory includes notebooks for profiling the database data and performing some data quality analyses.

The data profiling in the trusted zone database is found in the file "profiling_trustedDB.ipynb" inside the trusted zone directory. Please refer to the notebook's Section "Main Insights" to read the main conclusions extracted from the profiling of the data in the trusted zone database, which contains just one table per dataset.

The data quality analysis has been organised into multiple dedicated notebooks. For a comprehensive overview of the findings from each analysis procedure, please refer to the "Main Insights" in each notebook. This structured separation of methods for deduplication,

misspelling detection, outlier identification, and special character removal enables us to execute each data quality procedure independently, allowing for a focused evaluation of their impacts on our dataset.

This approach offers significant advantages, particularly in large-scale operations where the data quality checks encompass all cells of the dataset and may require considerable execution time. By running these procedures separately, we can optimise performance and manage execution time effectively.

Given the importance of these data quality procedures, it is essential to acknowledge a notable drawback: the need to navigate through multiple notebooks to ensure that all critical processes are completed.

On the other hand, it is essential to clarify that the outlier detection notebook is explicitly designed to identify potential imputation errors rather than to manage outliers for model improvement. The primary focus of this notebook is on detecting these anomalies, which may indicate data entry issues. Once an outlier is determined not to be an imputation error, it is not addressed within this notebook; instead, outlier handling is conducted later in the data management backbone pipeline.

In conclusion, the data quality analysis has demonstrated that the data in the database was inherently clean. However, implementing these methods is beneficial for verifying the cleanliness of the data and preventing potential integrity issues in the future.

5. Exploitation Zone

In this part, we explain the tables that are created in the exploitation zone database:

Firstly, we created a **GeographicalInformation** table that organises geographic details from the most specific to the broadest level, with the columns CodiDistricte, NomDistricte, CodiBarri, and NomBarri. By centralising geographic information in this table, we can remove these columns from other tables, keeping only the most specific geographic level (e.g., Neighbourhood code and SeccioCensal). This approach simplifies the structure of our other tables and still allows us to easily query data at broader geographic levels by joining them with GeographicalInformation.

With the creation of this table, we enhance the query efficiency. Organising geographic information in a dedicated GeographicalInformation table reduces redundancy, streamlines the structure of other tables, and optimises queries by joining only the relevant geographic details as needed.

The second and third tables are **OpenDades** and **PortalDades** tables, which are new versions of the same tables in the trusted zone. These tables do not contain columns that do not provide a considerable amount of information and do not have all the geographical data since we can join the tables with the **GeographicalInformation** to make queries about other geographic regions. Therefore, they are simpler and easier to understand and query, and so they are adequate for

the exploitation zone. With the creation of these tables, we seek data clarity and accessibility. By simplifying the OpenDades and PortalDades tables and removing non-essential columns and unneeded geographic data, the tables become more accessible to understand and query, making them better suited for direct analysis. Therefore, these tables are suitable for executing more specific queries, allowing simplified and more efficient filtering of the data.

However, a downside of simplifying the main tables is, for example, that the analysts will not be able to perform KPIs regarding the evolution of the prices in the four trimesters since we just provide the average price.

Finally, we created a primary table, **UnifiedData**, which consolidates all relevant information from both **OpenDades** and **PortalDades** by merging them on the Any and Barri columns. This unified table is well-suited for in-depth socioeconomic and demographic analysis, as it compiles the most valuable data gathered from the landing zone. Having passed through all processing zones, the data now reaches the exploitation zone in its most refined form, maximising its analytical utility and potential impact.

With this table, we achieve comprehensive data integration. The UnifiedData table consolidates the most relevant information from OpenDades and PortalDades, allowing analysts to work with a well-organized dataset that maximises analytical depth while maintaining the data's integrity. Therefore, this table allows more complex queries which involve many different data, such as KPIs.

With the mentioned tables, we achieved a balance between specific and efficient queries and more informative queries involving lots of data, allowing the analysts to choose between a range of different options before executing queries.

In the following notebook, we defined several KPIs together with their explanation, interpretation and computation. Please check the notebook to see which KPIs we extracted from the data.

Additionally, we have created a comprehensive profiling notebook that highlights the most insightful aspects of the OpenData, PortalDades, and UnifiedData tables. This analysis brings attention to critical patterns, distributions, and potential anomalies within these datasets, helping to establish a solid foundation for further exploration. We chose not to profile the GeographicalInformation table, as we determined it would add limited analytical value in this context. Note that profiling was done before the data quality analysis in order to analyse the data as it was without any modifications.

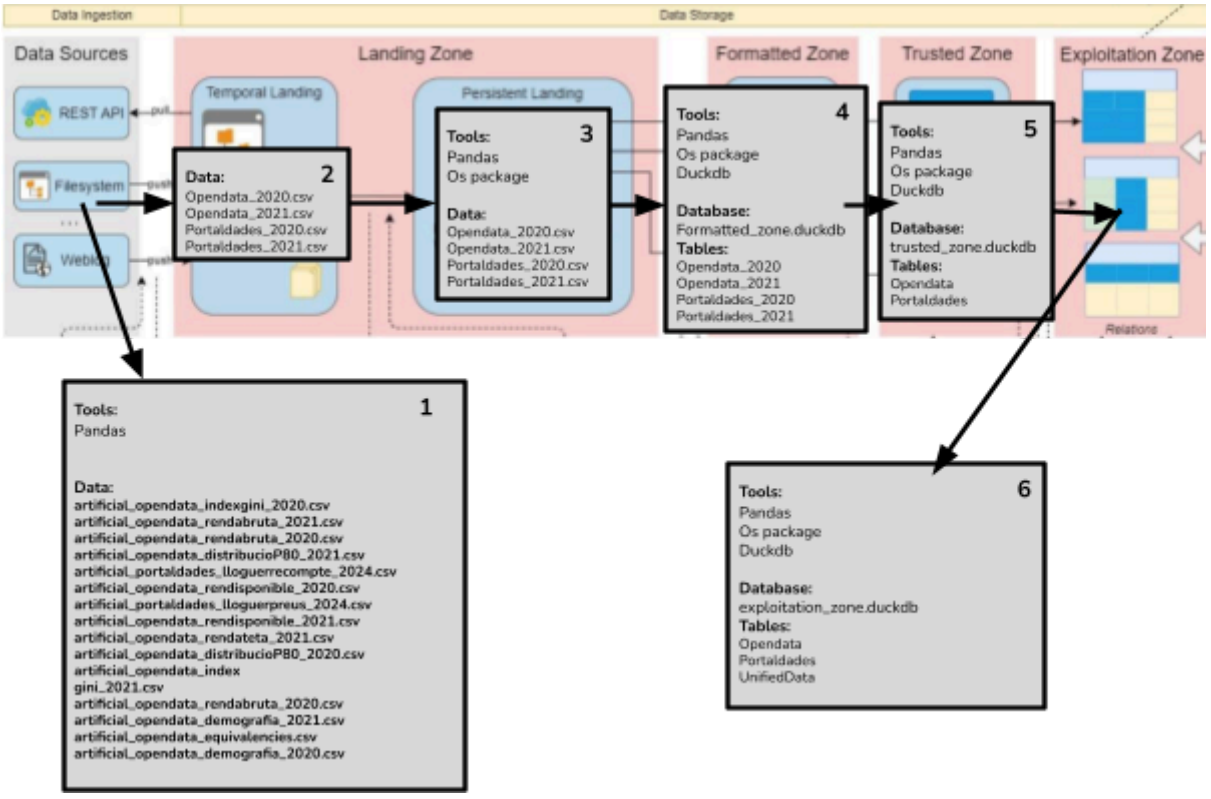
In the context of our data quality analysis, it is essential to note that the insights documented in the data quality notebooks within the trusted zone are also reflected in the exploitation zone. This overlap occurs because the tables created in the exploitation zone serve as simplified representations or combinations of those assessed for data quality in the trusted zone. Specifically, this behaviour is clearly observed in the OpenData, PortalDades, and GeographicInformation tables.

The UnifiedData table, which integrates relevant information from OpenData and PortalDades, illustrates an important aspect of data quality management. Although we can confirm that the individual datasets are clean- free from misspellings and imputation errors- the process of joining these datasets may introduce new challenges. Specifically, the join operation could lead to missing values in specific rows due to discrepancies in the critical fields used for merging.

To address these potential issues and ensure the integrity of our data, we have implemented a dedicated notebook focused on managing missing values in the UnifiedData table. Additionally, this notebook also handles outliers across all datasets since this was not addressed in the trusted zone data quality notebooks. Once this notebook is executed, the data is updated in the database and ready for analysis.

Summary

With all zones of the data management pipeline clearly justified, we now present a comprehensive summary of the pipeline's architecture. This includes an overview of each pipeline element, the tools utilised in each stage, and a detailed outline of the datasets and transformations specific to our pipeline. This summary captures how each zone integrates to support robust data handling, from initial ingestion to final analysis, ensuring data quality, consistency, and reproducibility throughout the process.



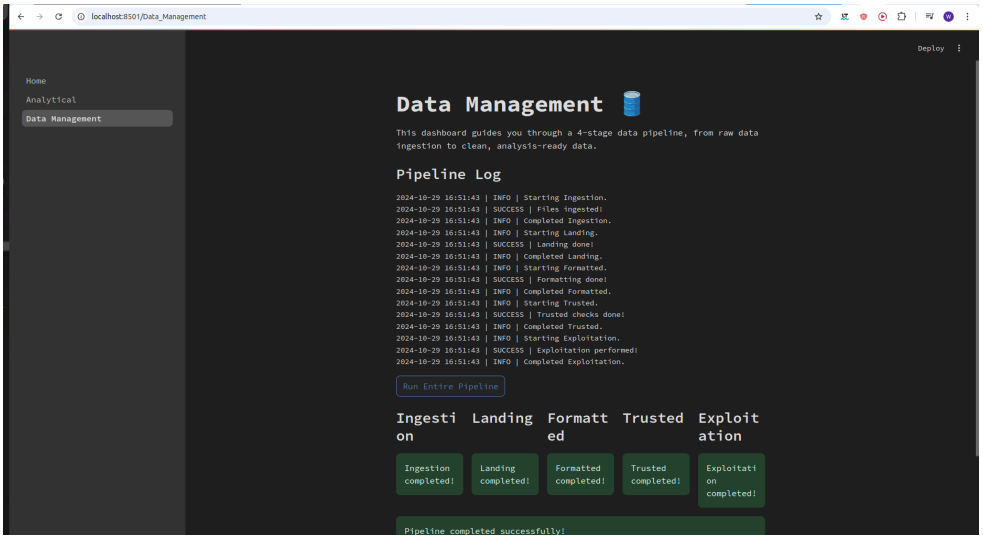
Operations

To prepare the operational environment for production, we heavily reused the development notebooks. In fact, nearly everything has changed except for the directory tree structure, paths (using pathlib package instead of os package to avoid bugs), and wrapper functions, which have been enhanced to improve the user experience. All zones have been encapsulated within wrapper driver functions that are called in the orchestrator code inside Data_Management.py, which serves as the driver for the data management backbone.

Next, we will discuss the technology stack used for this project. Firstly, we chose to implement the project using the Python ecosystem, given the versatility this language offers for data-related projects in both development (notebooks) and operational domains, especially compared to other popular choices like R or Julia. Furthermore, Python provides a wide array of off-the-shelf libraries such as Pandas, PyTorch, and NumPy.

Secondly, the database and management tool selected is DuckDB combined with Pandas. Although we considered using PySpark for its versatility in sharding, horizontal scalability, and extensive data management, we ultimately decided against it due to the relatively small size of our datasets. We opted for faster, in-memory frameworks like Pandas and DuckDB, as all datasets fit comfortably in memory. Utilising more robust frameworks like PySpark would have been overkill, introducing unnecessary overhead and complicating the memory hierarchy.

Thirdly, after evaluating various tools such as Streamlit, Django, Flask, Airflow, and Dagster, it became evident that Streamlit was the best choice for our user interface. We discarded Django and Flask due to the complexity of development and the higher costs associated with prototyping compared to the rapid development capabilities of Streamlit. While Airflow and Dagster offer comprehensive data pipeline interfaces with logging and a rich ecosystem, they are too specific to data management. This specificity could hinder our ability to scale in the future, especially when we need our user interface to support data analysis and dashboarding without relying on external tools like Tableau. Streamlit allows us to prototype generic and complex multipage web applications quickly.



Lastly, following the MLOps / DevOps methodologies religiously, we decided to implement the solution using Docker to guarantee portability between operating systems, computers, and software versions. Additionally, a simple resource monitoring script and some tests have been added inside the /monitoring directory.

Lastly, we would like to address a critical topic that has yet to be covered in the first part of the project: the implementation of a comprehensive CI/CD pipeline. Due to limited time constraints, this aspect was deemed unfeasible and ultimately set aside. However, integrating a CI/CD pipeline would significantly enhance the project by enabling essential actions such as testing (using PyTest), static code analysis (with SonarQube), linting, and advanced logging (via Prometheus/Grafana). This will be reconsidered in the second part of the project.

Another potential challenge in our operational environment is the absence of YAML configuration files, which limits the code's versatility and flexibility. However, given the project's current state, this issue is not immediately relevant.

Additionally, there is a notable limitation regarding the coverage of unit testing, primarily due to the complexities involved in testing data pipelines. It may only be worthwhile to conduct tests with critical aspects to monitor. This area will also be revisited in Part 2, as it will have a more significant impact when addressing models.

In conclusion, the added code is kept intentionally minimal to prioritise simplicity and flexibility, ensuring that the pipeline can be easily modified or adjusted with minimal effort. Additional scripts, such as those for monitoring and testing, act as optional enhancements that contribute to the resilience of the project but aren’t strictly essential for its core functionality. The inclusion of a Dockerfile, however, is crucial—it guarantees a secure, portable environment necessary for consistent and replicable performance across various platforms. In short, the code additions are fully justified, balancing the need for an orchestrated pipeline with UI support while avoiding any unnecessary complexity.