Universitat Politècnica de Catalunya

Facultat d'Informàtica de Barcelona

Large-Scale Data Engineering for AI

Advanced Databases (BDA)

Autors:

Chen, Hao; **Chen, Pengcheng**; **Chen, Zhihao**; **Zhou, Zhiqian**

April 13, 2025

1 Introduction

In data science projects, building reliable data pipelines is crucial. A well-designed data engineering pipeline ensures high-quality data is delivered to stakeholders like data scientists and analysts. These pipelines are standardized across organizations for consistency, scalability, and reliability.

A typical data engineering pipeline is organized into distinct zones. These Zones are Landing Zone, Data is ingested in and organised in a Data Lake; Formatted Zone, Data is homogenized according to a canonical data model; Trusted Zone, The data quality is assessed and data cleaning processes are applied; and Exploitation Zone, exposes data ready to be consumed/analized either by advanced analytical pipelines or external tools. Once the data engineering pipeline has delivered clean, reliable data to the Exploitation Zone, it feeds into the data analysis pipeline, which focuses on activities such as data exploration, feature engineering, preprocessing, modeling and validation.

This project focuses on building an end-to-end data science pipeline that incorporates big data techniques, primarily using PySpark as the main data processing tool.

2 Data Sources

Before diving deeper into each zone, it is important to define the data sources involved. In this case, the pipeline will work with three data sources related to PlayStation videogames and they are available in Kaggle. The first source provides a general overview of each game, including attributes such as title, genre, release data, developer; then, the second source provides pricing details for the games across different currencies; and finally the third one is about which games have been purchased by each player.

We chose to use Kaggle for these datasets because, although the data is also available on a public webpage, obtaining it would require scraping and selecting each item individually. Fortunately, the dataset publisher has already compiled and organized this information on Kaggle, saving us considerable time and effort. Additionally, we anticipate that the publisher will continue to update the dataset over time, ensuring that we have access to the most recent data.

On the other hand, these datasets are easy to work with, as they all come from the same source (authorized by the professor), and there are no issues with entity identification since they all share a common key called "gameid", which greatly simplifies data integration.

3 Data Storing

The three DataFrames were saved to a local PostgreSQL instance temporarily mounted in Colab. This step simulates loading into a structured database within each Zone. Although this database is deleted at the end of the session, it allows us to test the integration process with PostgreSQL and ensures the data is in the appropriate format for subsequent queries.

PostgreSQL was chosen due to its suitability for real-world environments that require relational databases and complex SQL queries. Unlike DuckDB, which is designed for more lightweight analytics, PostgreSQL can be easily integrated with Spark and simulated as a production database.

Additionally, we set up a remote PostgreSQL instance hosted on Render, which is fully functional and publicly accessible. Its integration is demonstrated in the notebook titled ”postgreSQL”, where we show how Spark can connect to and interact with it using JDBC. However, this remote database wasn’t used for storing the formatted datasets due to storage limitations in the free plan (1GB capacity), which restricted data scalability and data volume.

Since the permanent database in Render was ultimately not used, but rather the temporary PostgreSQL instance mounted in memory in the Google Colab environment, we decided to save the transformed datasets in Parquet format on Google Drive, ensuring the data could be preserved and reused in future stages of the project without repeating the entire pipeline.

4 Landing Zone

To enter the Landing Zone, we defined a data collector that extracts data directly from Kaggle using its API. The retrieved data, in `.csv` format, is initially saved to a Google Drive folder.

However, `.csv` files are neither efficient nor reliable for big data or large-scale data processing. Therefore, we transformed the data into the **Parquet format**. The choice of Parquet is driven by its ability to handle both row-based and column-based storage, which optimizes data compression and speeds up query performance, especially for analytical tasks. This makes it much more suitable for large datasets compared to `.csv` files. Additionally, Parquet’s support for efficient schema evolution and its self-describing nature make it easier to manage and access during subsequent stages of data processing.

Once the data has been transformed into Parquet format, it is stored in Google Drive again. Although Drive is not an ideal data lake, it is sufficient for the current scope of the project. Alternatives like HDFS offer better scalability and efficiency; however, given the relatively small size of the Parquet files and the absence of data replication requirements, using HDFS would be excessive for this project.

4.1 Data Collector

Currently, our data comes from a Kaggle API, which does not offer a mechanism for incremental ingestion or allow access exclusively to new or updated data since the last run. Each time the API is queried, the full dataset is retrieved, with no distinction between previously ingested and new records.

In this context, implementing logic to identify and retain only new data (for example, by comparing with the previous version and merging changes) is neither practical nor efficient. Since the API already delivers all updated content, such an approach only adds complexity to the execution process without providing any real benefits. In practice, the result would be the same as simply using the latest full dataset provided by the API.

Therefore, we consider it more appropriate and functional to always work with the most recent version of the full dataset, ensuring consistency and simplicity in data collection.

On the other hand, the automation of the data collector can be achieved by configuring a *cron job*, which allows for the periodic execution of a Python script. This process is not explicitly demonstrated in this project, as it simply involves sending a command to the server system.

5 Formatted Zone

5.1 Data Formatting

In the Formatted Zone phase, the three main datasets (games, prices, and purchased_games) were processed to ensure data type consistency and, a clean and uniform structure. First, the type of each variable was corrected (for example, changing release_date from string to date). Furthermore, brackets and quotes were removed from the columns that originally contained lists of strings, leaving flat values more appropriate for storage and querying.

In addition, the purchased_games dataset was transformed to convert the library column—which contained lists of games by user—into a tabular format, to obtain the sales volume per game, as the primary focus of this project is on analyzing games, not individual gamers.

6 Trusted Zone

6.1 Data Quality

This project prioritizes data quality, focusing on three key aspects: completeness, freshness, and consistency. Completeness ensures that no essential data is missing, verifying that all records are correctly filled out. Freshness is assessed based on available date fields, assuming all records are created at the same time; if a dataset doesn't have its own date, such as "games_purchased," the freshness of the related dataset (games) is assumed. Finally, consistency is validated using a set of nine Denial Constraints, which act as rules that data must meet to be considered valid and consistent. This combination ensures that the data is complete, up-to-date, and structurally correct. We did not use the Accuracy to determine the quality of the data, because we don't know the real value of each field.

6.2 Denial Constraint

A denial constraint is indeed used to enforce logical integrity and business rules within the database. It's a mechanism to ensure that the data doesn't violate certain conditions or create inconsistent or invalid states in the database. The key point here is that it prevents specific combinations of data from being valid or allowed. Without these constraints, data might get inserted that contradicts the rules, leading to inconsistencies or errors that can affect reporting, decision-making and data integrity.

After a brief analysis of the general purpose and the general meaning of each feature, we implemented a total of 9 different Denial Constraints:

1. **Unique Gameid Across All Tables**: This rule is a **Key Dependent Rule** that ensures that each game has a unique gameid across all tables.

2. **Platform Domain Restriction**: This rule is a **Domain Rule** that restricts the values of the platform column to a specific set of values.

3. **No Duplicate Games with Same Title and Platform**: This is a **Conditional Functional Rule** that ensures that for a specific combination of title and platform, all other attributes like developer,

publisher, release_date, genre, and supported_language must be identical.

4. **Release Date Validity**: This is a **Cross-Column Predicate** that ensures that the release_date of a game is not earlier than the release_date of the platform on which it is released.
5. **Single Game ID for Same Game Across Platforms**: This is a **Key Dependent Rule** that ensures that for the same game, defined by the combination of title, developer, publisher, etc., but different platforms, there must only be one gameid.
6. **Release Date Before Current Date**: This is a **Cross-Column Rule** that ensures that the release_date of the game is always in the past (i.e., before the current date).
7. **Non-Negative Price**: This is a **Domain rule** that ensures that the price of the game is non-negative.
8. **Acquisition Date Validity**: This is a **Cross-Column Rule** ensures that the acquired_date (the date of acquiring the price) is before the current date.
9. **Non-Negative Sales Volume**: This is a **Domain Rule** ensures that the sales volume is non-negative, meaning no record can have a negative sales volume.

6.3 Assessment of Data Quality

The initial assessment of the datasets reveals that they are neither fully complete nor consistent. As a result, the next step in the project will focus on data cleaning to enhance these aspects of the data quality. The goal is to improve the completeness and consistency metrics until they reach a level of 1 (i.e., full completeness and consistency). Regarding freshness, this aspect is out of our direct control. It depends on the update frequency set by the data source author. However, we assume that our data collector operates continuously, ensuring that we have access to the most recent data possible at any given time.

6.4 Preprocessing

The game dataset underwent extensive cleaning to ensure game ID uniqueness, remove exact duplicates, and address inconsistencies such as future release dates, which were replaced with the current date. Textual values were normalized by converting them to lowercase and removing characters such as brackets in list columns. Additionally, multiple rows with the same title and platform were unified by combining attributes (such as genres or developers) and retaining the earliest release date. Any platform outside the set PS3, PS4, PS5, PS Vita was also categorized as NA and missings were imputed with “unknown”.

In the price dataset, future dates were corrected by replacing them with the current date, and all prices were validated to be non-negative. Any price value less than zero was replaced with missing values, which were subsequently imputed with zero as a simplification measure. This imputation allows the analysis to continue without introducing errors, enabling the application of more sophisticated techniques in subsequent stages.

For the purchased games dataset, instead of removing duplicates, the sales_volume values for each gameID were aggregated, assuming that each repetition indicates a new sale. All sales were verified to have non-negative values, and missing values were imputed using the mean of the rest of the dataset. This strategy allows sales information to be kept up-to-date and consistent.

After processing all three dataset, high data quality was guaranteed for all three, obtaining a score of 1.0 for both completeness and consistency.

7 Exploitation Zone

To integrate the three datasets (games, prices, purchased_games), a join was performed based on the common key gameid, thanks to the prior preprocessing that guaranteed its consistency. However, since some games with the same title and platform have different gameids, the sales_volume values were aggregated, grouping by (title, platform) after joining the sales with the original dataset. This aggregated information was then combined with the cleaned dataset of games also by (title, platform). Finally, approximately 500 records that did not have associated sales data were removed, as they represented a small fraction of the total. The result was a unified and coherent table, ready for analysis or modeling.

8 Feature Engineering

During the feature engineering phase, new variables were created and others refined to improve the quality of the dataset for analysis and predictive modeling. The supported_languages column was transformed into a numeric variable indicating the number of languages available per game, with missing values imputed using the discretized mean. The genres variable, which contained multiple values per row, was simplified by keeping only the main (most common) genre and grouping rare genres into an "others" category. For developers and publishers, rather than individually processing the numerous unique values, a new classification was created based on the number of games produced: high, medium, or low production. Additionally, price imputation was improved, replacing the previously simple method (price = 0) with a more accurate imputation using *KNNImputer*, which estimates missing values based on similar games. This more sophisticated approach allowed us to maintain the richness of the dataset and prepare it for more robust analysis.

9 Data Analysis

Three Data Analysis Pipelines were run. In the first, several models were tested to predict video game sales volume based on their attributes. Performance was limited: linear regression obtained an R^2 of 0.16 and Random Forest reached 0.3 in testing. The neural network showed severe overfitting, with high R^2 in training but low R^2 in testing. Regularization, cross-validation, and hyperparameter tuning are suggested to improve generalization.

In the second pipeline, PCA was applied to reduce the dimensionality of the dataset, explaining 40% of the variance in the first two components. Two clearly defined clusters and some outliers were observed. However, no correlation was found between these clusters and developer type, indicating that the clusters are due to other underlying factors.

In the last pipeline, the analysis by genre revealed that Action and Adventure games lead in total sales volume. When normalized by the number of games per genre, Shooter and Free-to-Play stood out, with high sales per title. This suggests that, although less common, these genres tend to have highly successful games.