Universitat Politècnica de Catalunya

Facultat d'Informàtica de Barcelona

PROJECT I

Bases de Dades Avançades

Llum Fuster Palà
Asal Mehrabi
Marc Ucelayeta

17/03/2024

Índex

1 Introduction

This project will consist into building an end-to-end data science pipeline that takes raw data from at least three different sources and turns it into actionable insights. We will be going through different phases to get our main objective. Finally we will implement two Data Analysis pipelines to show the usefulness of our data engineering job.

1.1 Main idea

Our main idea is to build an application for travelers. We want to give a platfform where we can put together all the necessary information for a traveler such as their flights information, the weather and the accomodation.

1.2 Tools

For this project, we are using the Python programming language, which we are most familiar with and which offers extensive tools for data management and data analysis. To manage our databases, we will utilize PySpark, as it allows us to parallelize the functions applied to the data efficiently. As our execution environment, we will use Google Colab, which facilitates the use of PySpark, especially on Windows systems where Java installation is required and can lead to complications.Additionally, we are making updates and saving the databases in separate directories within a GitHub repository to maintain control over every file.

You can find all the code in our github repository bda_viatges

2 Landing Zone

The initial phase of our project focuses on identifying and acquiring data that aligns with our core objectives. We aim to source data from three distinct areas: accommodation, weather information, and flight details.

2.1 Data Sources

Accomodation

In our search for accommodation data, we encountered challenges in finding an API that did not require a subscription model or credit card details for access. Consequently, we opted for a database from Kaggle: Arib&b: Airbnb Prices & TripAdvisor Ratings (Econometrics). This database presents us with a unique challenge as it is segmented into various datasets by city and distinguishes between weekday and weekend data. This adds an extra layer of complexity to our data integration and analysis efforts.

Weather Information

For meteorological data, our exploration involved searching for accessible APIs. Eventually, we discovered an API that does not require an API key for access: Open Meteo. Despite its request limitations, this API sufficiently meets our academic project's needs.

Flight Data

For flight information, we first picked one database (Aviation Data) but ended up changing our minds after a lab class. We discovered an API (FlightRadarAPI) that lets us access flight data, which seemed like a better fit for our project. Originally, we were going down one path with our project, but this new flight data API takes us in a different direction. We'll explain both our first idea and the new one, showing why we decided to switch and how it affects our project.

2.2 Consolidated Idea

Now that we have all the data sources, we can stablish which is our idea in a more clear way. As we have said, we have been through a process of consolidating our project with two main ideas. The first one apeared we had the airbnb database, the weather API and the first flights data source. As the database mainly covered airport flows (number of flights, people arriving and leaving...) we had the idea of recomending destinations depending on the tourism interests. For example, for a person who went to explore a lot of different cultures and meet lot of new people we should recomend a crowded destination.

However, our concept cahnged significantly following a laboratory class, which underscored the importance of how we acquire data for our academic purpose. We found a new data source from Flight Radar, which gives us different information than what we had before. This new information made us think of a new direction for our project.

Our revised project aims to use real-time flight arrival information to estimate the best time for a driver to pick up clients from the airport and take them to their accommodation. We can also suggest Airbnb places for them to stay at before they arrive. In simple terms, our project has turned into a travel service that not only finds people places to stay but also helps with their airport transportation, making their travel experience easier.

2.3 Data Collectors

We have constructed three data collectors for each of the data sources. All of them will be put in the datalake in a parquet format.

Airbnb

When constructing this dataset, we initially had various files in CSV format downloaded from Kaggle. Each file was associated with a specific city and differentiated data collected during weekends and weekdays, reflected in both the file name and the data itself. The goal was to merge all these files into a single dataset to simplify further analysis and processing.

To achieve this, we underwent a process to unify the CSV files. During this process, two significant variables were added to each data record to effectively preserve and differentiate the original information:

- A **City** column was introduced to identify the data's origin, indicating which city each record belonged to. Since each initial file was associated with a specific city (like Barcelona, Amsterdam, etc.), this column was filled with the relevant city name for each dataset before merging. This ensured that, despite combining data from different sources into one dataset, the origin of the data remained clear and accessible.
- A **Day of the Week** column was added to denote whether the data pertained to a weekday or a weekend. This distinction was vital as data patterns could significantly vary between workdays and non-workdays, thus influencing subsequent analyses. By maintaining this differentiation, it facilitated more nuanced and accurate insights.

After this preparation and unification process, the dataset was saved in Parquet format. This decision was made to take advantage of Parquet's efficient data compression and encoding schemes, making the dataset more manageable and performant for processing and analysis tasks that followed.

OpenMeteo

In constructing our data collector for weather information, we made several key decisions to ensure the relevance and comprehensiveness of the data collected. The first step involved identifying the cities for which we had accommodation data. This alignment is crucial for merging the weather data with accommodation options later in our pipeline.

To achieve a detailed spatial representation of each city, we took the central coordinate of each city and generated a cloud of 400 points around this central point. We then visually plotted these points

to verify their coverage and representativeness across the entire city area. This approach ensures that our weather data reflects varying meteorological conditions within different parts of the city, rather than relying on a single, potentially unrepresentative data point.

To gather comprehensive weather data from the OpenMeteo API, we collected several parameters on an hourly basis, forecasting the next 16 days. These parameters include temperature, to gauge whether the conditions will be warm or cool; humidity levels; precipitation, to predict rainfall and its potential impact on vacation experiences; and sky condition, to determine if there will be sunshine despite favorable temperatures.

This detailed, multi-faceted approach to collecting weather data ensures that our application can provide travelers with an extense understanding of expected weather conditions, ultimately helping them in making informed decisions about their Airbnb stay.

FlightRadar

For the FlightRadar data, we implemented a collector that interacts directly with the FlightRadar API. This API provides real-time flight data, which includes arrivals, departures, flight statuses, and even live aircraft positions.

The first step was to fetch a list of all airports via a single API call. Initially, we wanted to collect information on all these airports, but we got an error about API call limits. Therefore, we decided to filter the list of airports, focusing only on those whose names include any of the cities from our Airbnb data. This method, however, risked including airports from non-European cities with the same name as our targeted cities, a simplification we accepted due to the API's usage limits.

Next, we processed the names of the selected airports to extract key information: their codes, latitude, longitude, and associated city, creating an initial JSON file. This intermediary file was particularly useful as we reached our API call limit for the day.

Once we were able to make calls again, we retrieved information on the first 100 arriving flights for these airports, resulting in another JSON file. This file was added with an 'arrivals' key, listing all the incoming flights.

The final step involved storing this data in a Parquet file. Our first attempt did not meet our format expectations, prompting us to employ a map-reduce-like structure to experiment with key-value pairs. Finally, we saved the data in a Parquet file with meaningfully named columns, successfully compiling the required information for our data lake.

3 Formatted Zone

The transition from the Landing Zone to the Formatted Zone marks a crucial phase in our data pipeline. In this stage, we change our focus from raw data collection to structuring and storing this data in a relational database management system (RDBMS) that facilitates analysis and insight generation. For our project, we have chosen DuckDB.

3.1 DuckDB

We have used DuckDB as our RDBMS because of its simplicity in handling workloads directly from Python. We don't even need to host it anywhere like PosgreSQL, it's just a file we can store locally. It is also useful due to its compatibility with pyspark and also because we were told that we would be able to handle our json data with this tool.

4 Trusted Zone

In the Trusted Zone, we enhance data reliability and usability through meticulous cleaning and transformation processes. This section details the techniques we used to preprocess our datasets, preparing

them for integration and analysis in subsequent phases of our pipeline. Emphasis is placed on structuring data in a manner that aligns with the requirements of our data analysis pipelines, ensuring that our data is optimally prepared for analysis.

The Trusted Zone is, again, a duckDB database which contains three tables, one for each data source. We have created it in parts, populating the tables one by one when the notebook is executed.

4.1 Airbnb

In this part, we have modified various columns and variables in order to get the best results. Firstly, we have analysed the missing values and outliers and once this step was done, we procceeded to create new variables such as cleaniness rating as a categorical variable. We also tried to do an clustering for grouping two main variables which were the distance from the metro and the distance from the center and we divided that into 7 clusters.

4.2 OpenMeteo

For the meteorological database, various considerations have been taken into account. The first approach applied to this database is the study of missing values. These missing values appeared only in the `precipitation_probability` column (the probability of rain on that specific day, time, and place). Initially, it was considered that if the total precipitation on that day, time, and place is 0, the probability of precipitation is also 0. However, as there were still missing values, these were imputed by averaging the precipitation probability for that day and place.

The next step involved changing the `weather_code` variable. This variable was numeric, but it represented the state of the sky. This change of representation was done using a conversion table in the documentation of the OpenMeteo API[1]:

WMO Weather Code (WW)	**Description**
0	Clear sky
1, 2, 3	Mainly clear, partly cloudy, and overcast
45, 48	Fog and depositing rime fog
51, 53, 55	Drizzle: Light, moderate, and dense intensity
56, 57	Freezing Drizzle: Light and dense intensity
61, 63, 65	Rain: Slight, moderate and heavy intensity
66, 67	Freezing Rain: Light and heavy intensity
71, 73, 75	Snow fall: Slight, moderate, and heavy intensity
77	Snow grains
80, 81, 82	Rain showers: Slight, moderate, and violent
85, 86	Snow showers slight and heavy
95	Thunderstorm: Slight or moderate
96, 99	Thunderstorm with slight and heavy hail

Taula 1: WMO Weather Interpretation Codes

The next thing we have done, as we will not use the weather for each hour of the day, but rather the general weather that occurred on a specific day, we will group the rows by day and location.

The rows have been grouped by averaging, except for precipitation, which has been grouped by aggregating. For the new variable created with the description of the sky at each hour (weather_description), we will create a dictionary where we count the occurrences of each sky state to subsequently, through a set of rules, describe the global sky state of the day.

Some of the states have been grouped due to their low representativeness, and others have been grouped by averaging, for example, if it rained a little and a lot during the day, we say it rained moderately, maintaining the coherence of the variable. Here are some examples:

[1] https://open-meteo.com/en/docs

- "Despejado, Nublado"→ "Parcialmente nublado": If the sky has been clear and cloudy, we say it has been partly clear throughout the day.
- "Parcialmente nublado, Nublado, Lloviznas"→ "Parcialmente nublado, Lloviznas": If it has been partly cloudy, cloudy, and drizzled a bit, we say it has been partly cloudy and drizzled a bit.
- "Parcialmente nublado, Nublado, Lluvias intensas"→ "Nublado, Lluvias intensas": Contrary to the previous example, if it has been partly cloudy, cloudy, and rained heavily, we say it has been cloudy and rained heavily. That is, we adjust the clouds to the amount of precipitation.
- "Poco nublado, Parcialmente nublado, Nublado"→ "Parcialmente nublado": We leave the sky in an intermediate state.

4.3 FlightRadar

The processing of the flight data in the Trusted Zone involves several key steps to ensure that the data is clean, reliable, and suitable for integration with other datasets such as Airbnb and weather data. The transformation and cleaning processes aim to prepare the data for robust analytics, focusing on the prediction of flight delays which can significantly enhance the travel planning experience.

4.3.1 Json variable Handling

When we collected the data we encountered a significant challenge with one of the columns (*arrivals*), which was formatted as a list of JSONs containing detailed information about multiple flights.

We attempted to address this issue by performing an explode operation, which converted each flight's JSON into separate rows. We reached to do this; however, we were unable to access specific JSON fields like `arrival.flight.identification.number.default`, as these attempts returned nulls. The root of the issue was that the JSON fields were still being interpreted as StringType rather than the necessary StructType.

To resolve this, we returned to the data collection phase, implementing a revised approach directly within the DataCollector. We redefined the schema to accurately represent the nested structure of our JSON data using StructType, which allowed for more granular access and manipulation of the data fields. The schema included various nested fields under categories such as identification, time, owner, airport details, and flight status, ensuring that each aspect of the flight data was appropriately structured and accessible.

The revised code in the DataCollector included a comprehensive schema definition using PySpark's StructType to define a hierarchy that matches the JSON structure. This structure includes several nested fields for detailed attributes like flight number, estimated and actual arrival and departure times, airport gate, terminal information, and more. With this final scheme we were able to obtain different columns for flights informativve variables and get a new .parquet for the datalake (LandingZone).

As we have an organized code, we just needed to run DataFormatting script with the updated datalake and keep going on the quality pipeline. We know that this part should have been done in the formatted pipeline, but it was not easy because when we saved the data into .parquet format it was loosing all the structure information. We considered the idea of saving the json file in the datalake and then make all this work in the formatting pipeline, but we were told to save .parquet in the datalake and not anything else.

Handling Missing Data

The dataset initially contained numerous missing values, particularly in the arrival and departure time columns. These were largely due to the real-time nature of flight data, where many flights might not have yet departed or landed at the time of data retrieval. This characteristic makes the dataset less suitable for dynamic flight tracking analysis, which would be more effectively handled directly via the API. Consequently, for the purposes of our static data analysis, we focus on those flights that have information about real departures at least.

Delay Calculation

To identify flight delays, we created a new column, delay_in_seconds, initially set to NULL. We then calculated the delay as the difference between the scheduled and actual times of departure or arrival, whichever was applicable and available. The idea behind the code is to first check the departure time and then the arrival time. Of course, the arrival time will define better if the flight has delay because the delay can occur during the flight. Also we will first check the estimation time and then the real time if it's recorded. Finally we have a variable which defines the time of delay or advanced and we derive another variable has_delay, a boolean indicating if the flight has a delay or not.

Data Reduction

Given our focus on connecting flight information to accommodation data, we refined our dataset by removing specific time details and retaining only the date of arrival. This reduction was guided by the objective to sync flight dates with accommodation availability dates, simplifying the integration process. However, we retained the scheduled departure time as this information could be vital in predicting delays, which are often influenced by the time of day. Also, we are keeping the date to match the weather that day.

Categorization of Time of Day

We further categorized the time of day into morning, afternoon, evening, and night based on the scheduled departure time. This categorization is instrumental in analyzing patterns of delays throughout the day, because usually there are differences in delays depending on the time of the day.

Column Streamlining

Our next step involved streamlining the dataset by removing columns that were not essential for our analysis. This included various time columns that were redundant after delay calculations, as well as specific gate and terminal information that did not directly contribute to our primary analysis goals.

Handling Null Values

Post-cleanup, we assessed the dataset for any remaining null values and replaced them with 'Unknown' where applicable. This step ensured that our dataset did not contain any missing entries that could skew the results or complicate further data integration tasks.

5 Exploitation Zone

In this area, we will set up the necessary databases that will receive input from the following area. Therefore, these databases will be clean and structured to perform the corresponding `select` queries.

The different databases will be joined based on geographic coordinates. However, these are not exact because in the FlightRadar database, the coordinates represent airports; in the Airbnb database, they represent apartments; and in the OpenMeteo database, they represent points evenly distributed across the city. Due to this fact, we will join the rows by the minimum Euclidean distance.

The methodology we have followed is to only create a new table when it is necessary and useful, as creating many tables within the database can become inefficient and computationally heavy. Therefore, only in the first case, where we have a database of airports and their nearest geographic coordinate is really useful since it saves us from adding the nearest coordinates to all the rows in the flight table.

In the second case, as we want the nearest geographic coordinates from the meteorological database for each Airbnb row, we directly add this information to the original Airbnb table.

6 Data Analysis Pipelines

In the Data Analysis Pipeline section of our project, we outline the process we followed for advanced data analysis. After obtaining and refining three datasets, we focused on creating additional variables needed for analysis. Once all datasets were properly prepared and preprocessed, our goal is to merge them and assess their predictive capabilities using various AI methods.

With this objective in mind, we developed two pipelines: Pipeline 1 and Pipeline 2. Pipeline 1 is designed to predict whether a flight will be delayed or not based on weather conditions at the destination.

6.1 Pipeline 1

In this section, we will discuss the first prediction model we developed.

For this pipeline, our goal was to predict whether a flight would be delayed based on weather conditions. To achieve this, we initially explored processing the data using Spark and DuckDB, but eventually, we migrated to Python for better flexibility and control.

We utilized both weather and flight databases to prepare a joined dataset. Once the dataset was ready, we experimented with various machine learning models using Python. After trying different algorithms including logistic regression, decision trees, and Random Forest, we settled on the decision tree model with applying grid search, which achieved an accuracy of approximately 0.60:

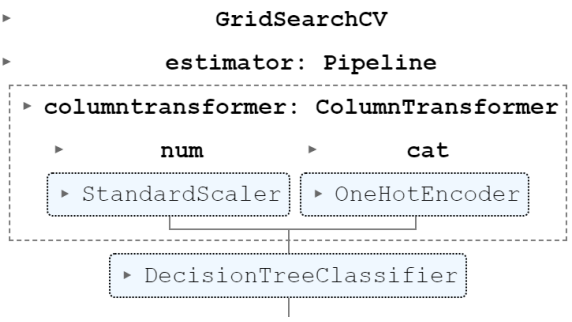

Figura 1: Grid Search

```
Best parameters found:  {'decisiontreeclassifier__criterion': 'entropy',
     'decisiontreeclassifier__max_depth': 9}
Best score found:  0.6003101334294987
Test score:  0.5972222222222222
              precision    recall  f1-score   support

           0       0.63      0.52      0.57       257
           1       0.58      0.68      0.62       247

    accuracy                           0.60       504
   macro avg       0.60      0.60      0.60       504
weighted avg       0.60      0.60      0.59       504
```

6.2 Pipeline 2

For Pipeline 2, we've developed a chatbot using the Airbnb and Weather datasets. The chatbot interacts with users, asking about their preferences such as weather conditions (e.g., rain, sunshine), desired location proximity to the city, and other criteria. Based on the user's input, the chatbot provides tailored accommodation recommendations. It considers factors like price, suggesting options with the lowest apartment prices and meeting the user's preferences. In summary, we've created an interactive chatbot that engages users, asks about their preferences, and recommends suitable accommodation options based on the Airbnb and Weather datasets.

Demo

```
------------------------------------------------

---------------------CHATBOT---------------------

--------We will find your ideal apartment--------

------------------------------------------------
```

Are you traveling on Weekdays or Weekends? Weekends How many people are you? (2 - 6): 3

What is the maximum price you want (34.8€ - 18545.5€): 4000

What types of rooms are you interested in? (Entire home/apt, Private room, Shared room): Private room

Which temperature you want? (8.6º - 15.1º): 13

How important is it for you that it doesn't rain? (Important, Not important): Important

The apartment 41384 with coordinates 38.7625 lat and -9.124 lon, in the city Lisbon, has the sky: Nublado

The apartment 43519 with coordinates 51.4527 lat and -0.0478 lon, in the city London, has the sky: Nublado

The apartment 10142 with coordinates 51.4696 lat and -0.0478 lon, in the city London, has the sky: Nublado

The apartment 33908 with coordinates 51.4443 lat and -0.0478 lon, in the city London, has the sky: Nublado

The apartment 4848 with coordinates 51.4864 lat and -0.0478 lon, in the city London, has the sky: Nublado

The apartment 10133 with coordinates 51.4274 lat and -0.0562 lon, in the city London, has the sky: Nublado

The apartment 19112 with coordinates 51.4948 lat and -0.0478 lon, in the city London, has the sky: Nublado

The apartment 23732 with coordinates 51.5201 lat and -0.0478 lon, in the city London, has the sky: Nublado

The apartment 50803 with coordinates 38.7288 lat and -9.1324 lon, in the city Lisbon, has the sky: Nublado

The apartment 32697 with coordinates 38.7372 lat and -9.1324 lon, in the city Lisbon, has the sky: Parcialmente nublado

7 Conclusion

In conclusion, we have conducted a comprehensive data analysis that allowed us to categorize the time of day, optimize the database structure, and handle null values effectively. Additionally, we have developed two data analysis pipelines: the first one to predict flight delays based on weather conditions, and the second one to create an interactive chatbot that recommends Airbnb accommodations according to user preferences and weather conditions. These analyses and tools provide us with a detailed and useful insight to understand and address issues related to flight delays and accommodation options.

Using PySpark, and DuckDB, we efficiently manage and process our data to ensure its reliability and usability. Our analysis pipelines, designed to predict flight delays and recommend accommodations, provide valuable insights for travelers, enhancing their overall travel planning experience.

Through careful data collection, cleaning, and analysis, we've created a robust framework to provide travelers with real-time information and personalized recommendations. Ultimately, our goal is to make travel planning easier, more efficient, and more enjoyable for users, ensuring they have the best possible travel experience.