Creating a Comprehensive Documentary Database for Restaurants and Hotels near Ski Resorts in Lombardy: An Exploration and Census Project

Simone Massardi, Alessandro Bianchi, Giorgia Prina

July, 2023

Abstract

This report describes the project of creating a documentary database encompassing advertisements related to restaurants and hotels in proximity to ski resorts in Lombardy, Italy. The database was constructed by gathering data from various online sources using data acquisition techniques and subsequently undergoing a process of cleaning and integration to ensure a highly accurate and comprehensive database that can be used for exploratory investigations or simple census of activities in the area.

The data acquisition process involved web scraping techniques and querying specialized online databases. The collected data includes key information such as geographical location, contact details, descriptions, and reviews of the establishments. Afterwards, the data underwent a cleaning phase to remove redundant or inconsistent information and was standardized to ensure a uniform structure.

Following the cleaning phase, the data was integrated to create a consolidated database. Data integration involved matching records based on criteria such as the establishment's name and location. Data from different sources were combined to provide a comprehensive set of information on restaurants and hotels in the vicinity of ski resorts.

The resulting database represents an important resource for tourism operators, researchers, and winter sports enthusiasts interested in exploring the hospitality and dining options in Lombardy. Access to an accurate and comprehensive database enables in-depth analysis, trend identification, and data-driven decision making. Thus, the project has contributed to creating an informative platform that promotes a better understanding of the tourism offerings in the Lombardy ski resort area and can be used to support future planning efforts and resource optimization.

Contents

1. Introduction

In this report we will show the different phases of data management, with particular focus on the acquisition phase, data cleaning, integration and storage. The overall quality of the data is finally assessed and some queries on the final database are provided.

2. Data Acquisition

In this section we present the sources from which we acquired the data, we describe the acquisition process and we provide an overview of the obtained datasets.

2.1. Sources

We retrieved the data from online listings of restaurants and accomodations (Hotels, apartments, BnB) in the proximities of the ski resorts of Lombardy (Italy). The ski resorts data have been acquired from Open Data site https://www.dati.lombardia.it, which contains different types of Regione Lombardia APIs.

The data concerning restaurants come from the websites of TripAdvisor, The Fork and Google.

The data concerning hotels come from two different sources: the first source is the Open Data site, the same as ski resorts data, the second is the scraping from Booking website.

2.2. Acquisition techniques

Regarding the announcements related to TripAdvisor restaurants, they were obtained through web scraping by combining the use of Selenium and BeautifulSoup libraries.

The code allows for automated control of the Chrome browser, directing it to the TripAdvisor search page. Subsequently, the names of the municipalities related to the ski resorts are cyclically entered, imported from the reference dataset. To determine which municipalities to extract, we used a CSV file containing all the ski resorts in Lombardy and created a deduplicated list of the municipalities present. This search leads to the list of restaurants present in that municipality or in the immediate vicinity. For each restaurant, the code opens the corresponding announcement and, using the BeautifulSoup library, retrieves information such as the name, URL, customer ratings, price range, contact, and address. It should be noted that not all announcements are complete, so in order to prevent the code from being blocked during execution, a clause has been added to append an empty string if any of the references are not found in the page's source code. It is also evident at this stage that some restaurants appear in searches related to different municipalities, reasonably due to their proximity. This problem will be addressed and resolved during the dataset cleaning phase.

Regarding TheFork, the data was obtained using Python libraries such as BeautifulSoup, requests.get, time, random, and pandas. Firstly, a dictionary of dictionaries was created, indicating the HTML pages to be requested for each location of interest. Then, using the requests.get function, we requested the HTML of the selected pages. Finally, using BeautifulSoup, we transformed the HTML into text that could be parsed by Python. With a custom-built function, we extracted the values we considered important and stored them in a precreated pandas dataframe. The informations scraped are name, address, rating, price, number of comments.

As for the data retrieved from Google Chrome, we used a Google extension tool called "Instant Data Scraper". This automatic tool, once the table from which to gather information is indicated, collects the data automatically and stores it in a database that can be downloaded as a CSV file. However, the efficiency of this tool is balanced by the fact that it is unable to scrape data from unrelated HTML pages. Therefore, for each location, we had to download a different file and then merge them together for further processing.

As we said in the previous section the data concerning hotels come from two different sources: the Open Data site of Regione Lombardia, and the scraping from Booking website.

Also in this case, as for TripAdvisor and The Fork, we used python, in particular Selenium and BeautifulSoup libraries to perform the get request for the API and then the scraping for Booking.

The dataset of choice is the one with the accomodation facilities in Lombardy which is available through API on SODA API. We then decided to use this API to download the data. We relied on the documentation to understand which are the access credentials and the available endpoints. We then requested "username", "password", "key", "app token" and "secret token". As an endpoint we used name
_comune. We iterated on the same list of all the municipalities used for TripAdvisor. In particular, we iterated over the API's "url" by appending the exact name of each municipality to the endpoint. Specifically, we set the value of "params" to take on the next municipality name for each iteration of the list. To avoid issues with the request, we decided to print the request status. This preventive method allowed us to understand why the code might not work properly. Once this was done, we were able to download all the data related to the municipalities of interest and saved them in JSON file.

The second data source we used was web scraping on the Booking.com website. We chose Booking because it is the global leader in accommodation offerings for all types of travelers, and its review system and continuous monitoring of listings and accommodations have created a high level of trust among users. Booking provides its own API, which has recently been updated. However, it requires registration and specific requirements, such as providing the "company name" or "VAT number" for the business one is working with. Since the data from this API is valuable for businesses operating on Booking, they provide endpoints that were not of interest to us and did not allow for general downloads based on municipalities or areas. Therefore, we opted for web scraping.

As we said before, to perform the web scraping, we used the Selenium and BeautifulSoup libraries. Selenium allowed us to access the WebDriver, a tool that simulates the behavior of a real user within a browser. BeautifulSoup is a Python library that allowed us to read the data present in web pages written in HTML language by selecting the tags of interest. Specifically, we decided to save the name of the property, the Booking.com link, the address, the municipality, the overall rating of the property, the number of reviews, and the ratings divided into categories such as staff, services, cleanliness, comfort, value for money, location, and free Wi-Fi.

Similar to the API, we created a loop to modify the URL queue so that it precisely searched for the municipalities in the list. Additionally, we used the information provided by Booking regarding the number of properties found for each municipality to select exactly that number of properties for scraping.

During this process, we noticed that sometimes the program produced an error related to the tags. Further analysis helped us understand that the driver occasionally opened two different pages with different HTML structures. We handled these exceptions by adding new tags, and finally, we saved the result in a CSV file.

2.3. Datasets

The dataset downloaded from Regione Lombardia website consists of 256 rows and 43 variables (last update 2018).

The dataset obtained through scraping from the TripAdvisor website consists of 1003 announcements, each with six attributes: name, URL, rating, price range, address, and contact.

The dataset obtained through scraping from the The Fork website consists of 956 announcements, each with five attributes.

The dataset obtained through scraping from Google consists of 993 announcements, each with seven attributes: name, type, rating, number of reviews, address, city.

The dataset got from API contains 3529 rows, 21 variables, and it is updated every 15 days. So, the latest update is 26/06/2023.

The dataset obtained through scraping from Booking contains 1881 rows, 13 variables.

3. Data Cleaning

The cleaning of the TripAdvisor restaurant dataset involves the following stages:

- **Removal of duplicates**: Since each announcement is unique within the website, it is sufficient to search for identical records within the dataset to identify and remove duplicates. It is not necessary at this stage to establish different criteria, such as similarity-based criteria. At the end of this operation, 104 duplicate announcements are identified.

- **Cleaning the "address" field**: The presence of various address formats makes it complicated to convert each one to a uniform format for all records in the dataset. The standard format adhered to by most announcements is typically "Street - house number - ZIP code - city - country". To perform an initial filtering, the "address" column is divided into sub-columns "street," "ZIP code," and "city" using a regular expression to locate the ZIP code. 66 "abnormal" addresses are excluded, which either lack the ZIP code or refer to Swiss locations where the ZIP code is a 4-digit number instead of 5. The records related to Swiss locations are removed as they are not relevant to the project's search query. To standardize the remaining records, which are in the "Street - City" format, the presence of one of the correctly identified cities is searched for in the string using a regular expression. Missing ZIP codes are later added by inserting the existing ZIP codes for the corresponding cities.

- **Cleaning the "contact" field**: This field should contain the phone numbers of the establishments. However, some records instead contain the restaurant's cuisine type. This happens when the contact number is not present in the announcement, and the cuisine type is mistakenly included during scraping. To address this issue, non-numeric values are removed from the column.

- **Cleaning the "rating" field**: The "rating" field contains customer ratings on a scale of 0 to 5 with increments of 0.5. Some announcements have a rating of -1, which, upon direct inspection, is automatically assigned in the source code and indicates the absence of ratings. These values are replaced with null values. Additionally, the rating format is converted from a string to a float to enable numerical evaluations on this field.

- **Cleaning the "price" field**: This field contains the price range of the restaurant expressed in $ symbols. To facilitate data interpretation, the symbols are converted into categories following TripAdvisor's official guidelines, where $ represents "inexpensive," $$-$$$ represents "moderate," and $$$$ represents "expensive cuisine."

For the cleaning of TheFork data, we started by using regular expressions to split the "locations" column into address, ZIP code, and city. Then, another regular expression was applied to clean the various columns from misplaced symbols and commas. The ZIP code column was cleaned, taking advantage of the fact that ZIP codes in Italy are standardized to five digits. Additionally, a new "rating" column was created to standardize the rating format to match that of TripAdvisor and Google.

Subsequently, a search for duplicates within the dataset was performed. The address and restaurant name were used as equality indices since, being homogeneous data taken from the same website, these two indices were sufficient to eliminate all duplicates.

The cleaning of the Booking structures dataset involves the following stages:

- **Removal of duplicates**: It was sufficient to search for rows that were exactly identical to consider them as duplicates. Duplicates often occurred due to the different extraction speed of information from tags and page loading. This means that sometimes the scraping process would copy the information of the same property twice. In terms of efficiency, we deemed it wiser not to increase the time sleep of the program as it would have required much more time to complete the scraping process, and performing a quick duplicate cleaning after saving the data was preferred.

- **Cleaning the "address" field**: In this case, just like we did for TripAdvisor, we focused on cleaning the fields in the "address" column. This column served as the foreign key to merge the data we downloaded from the API with the data from Booking. We noticed that in all cells after the actual address, there were the municipality name, ZIP code, and country. Therefore, we created three new columns, country, ZIP code, and municipality, to store these values. We split the address column using a comma as the splitting term. Since commas were not exclusive to these four fields and could also be present within the address section (e.g., to separate the street name and house number), we started from the end of the string, counting these three occurrences.

  Furthermore, we observed that some addresses had the house number at the beginning of the string, some at the end or in the middle, while others did not have a house number. Therefore, we divided the dataset into two distinct parts: "rows_with_number," which included all addresses with the house number at the beginning of the string, and "rows_without_number," which included addresses without a house number at the beginning, indicating that it could be at the end, in the middle, or not present. In the first case, we obtained a dataset of 180 rows.

  In general, we noticed that the address consisted of a maximum of four words. Therefore, we counted the first four words of the string and moved the remaining words to a new column called 'info_address'. We removed some duplicates within the two columns. Specifically, we searched for specific words such as "2nd," "Int.," 'floor,' 'piano,' and 'scala' in the 'address' column to move them to 'info_address'. Since there were only 180 rows, we handled the exceptions manually while attempting to automate as much as possible.

  Once we cleaned the address from additional information such as floor number or municipality (already present in the municipality column), we moved the house number to the end of the string and removed any additional symbols and spaces. We performed a similar process for the other sub-dataset, "rows_without_number," but in this case, we used numerical values as the splitting method.

  In both cases, we aimed to standardize the addresses and chose to:

  1. We made sure to expand all abbreviations, for example, "G. Garibaldi" was expanded to "Giuseppe Garibaldi";
  2. We transformed all accents, whether acute, grave, or apostrophes, into grave accents;
  3. The state roads are transformed into "ss+number."
  4. The internal numbers within the address are converted to Roman numerals (to differentiate them from house numbers). For example, "via 25 aprile" becomes "via XXV aprile.";
  5. The addresses that contain both street names (or squares or districts) and localities have been split, and the localities have been moved to a new column called "località" (locality).

The other columns were already clean, so we did not have to perform any specific operations. We checked the data types of the variables and standardized them, converting all variables to string.

Regarding the API, data exploration showed that the data had already been standardized and organized uniformly, so we decided not to make any modifications.

4. Data Integration and enrichment

The first step we took was related to the localization of hotels and restaurants. We noticed that during the scraping process from various websites, our functions were saving locations, municipalities, and cities of different kinds, not only exactly those from the list we used as a sample.

To better understand this reasoning, let's provide an example: when searching for restaurants in Bormio, the results included restaurants located in Cepina, which is a fraction of Bormio, or in Premadio (in Valdidentro), which is a neighboring location to Bormio.

What we wanted to do was consider all the conducted searches as valid, but assign to each fraction or location the nearest municipality chosen from those present in the list.

To achieve this, we used a function that could calculate the geographical distances between two places. By iterating over each row of the different datasets, this function would assign the reference municipality (i.e., the closest one), while preserving the original information in the dataset and adding a column called "reference" that contains the nearest municipality.

This operation was performed on the TripAdvisor, The Fork, and Booking datasets, but not on Google since the Google search produced municipalities that exactly matched those in the list.

Regarding the restaurant datasets, first of all, we renamed some columns to make all the datasets consistent, meaning that they had the same name for the same variables.

Next, we merged all the datasets by stacking them on top of each other using the 'concat' method. We then performed a search for duplicates. In this case, duplicates might not only refer to exactly identical rows, but also to the same restaurants found on different websites. Therefore, the only way to determine if we had a duplicate was by comparing the "reference" column (nearest municipality) and the "titles" column (restaurant name). Only when the references and names were exactly the same, we considered it a duplicate.

The next step was to add a "localita" column to the merged and cleaned restaurant dataset. In this column, we inserted all the municipalities from the "comune" column of the ski resorts dataset. This operation had the same purpose as the previous one: to assign the nearest municipality to each restaurant, which was also present in the ski resorts list.

As a precaution, we made several saves after each of these operations.

Finally, we replaced all null values and empty cells with NaN. Since, during the storage phase, we wanted the attributes with null values not to be displayed, we decided to save a JSON file where these attributes were eliminated.

Regarding the hotels datasets we performed an enrichment between API dataset and Booking dataset. We set the API database as the main database, or the one to enrich.

The first precaution we took was to rename some columns to avoid misunderstandings during code writing and iteration on the two databases.

Next, we performed an inner join between the two datasets as they were of reasonable size from our perspective. The join was done on the "comune" column, or more precisely, we checked if the "API_Comune" and "Booking_Comune" columns corresponded.

We then had to handle three distinct cases:

1. Rows with the same address (288 rows).
2. Rows with different and non-null addresses (74889 rows).
3. Rows with a null address in at least one of the two columns (68686 rows).

In the first case, we performed a check on the rows because even though we selected only those with the same address in the API and Booking data, there could still be cases of homonymy or simple errors. The approach we decided to take was to consider valid all the rows with a single occurrence. For those with multiple occurrences, we further checked the denominazione_struttura and nome columns, i.e., we verified if there was any similarity between the two cells. If there was a similarity, we kept the row as valid; otherwise, we excluded it.

In the second case, we worked on rows with different addresses. Here again, we couldn't rule out the possibility of addresses being the same.

We considered several specific scenarios, such as:

- The address being the same but with words in a different order (e.g., "via conte camillo benso di cavour" vs. "via camillo benso conte di cavour").
- The address being the same but differing by a single word (e.g., "via giovanni battista vico 11" vs. "via gian battista vico 11").
- The address being the same but the house number not existing in one of the cells (either in API or Booking data).
- The address being the same but explicitly mentioned in full in one cell while not in the other (e.g., "via giosuè carducci" vs. "via carducci").

- The address being the same but the house number in one cell including an interior while not in the other (e.g., "vicolo portichetti 3" vs. "via portichetti 3a").

Since the main issues in this second dataset were related to house numbers, we used regular expressions (regex) to perform checks specifically on house numbers. In all cases where the house number existed and was the same for both cells, we also checked for some similarity with the "denominazione_struttura" and "nome" columns. In cases where the house number didn't exist, we performed the same check on the "denominazione_struttura" and "nome" columns.

Example of a similar but not identical occurrence:

Denominazione struttura: terre aromatiche
Nome: b&b terre aromatiche
API Indirizzo: via somprato
Booking Indirizzo: via somprato 2

We also noticed that in some cases, the API provides specific information about apartments managed by the same host in the same building. Therefore, the address could match except for the apartment number, as well as the "denominazione_struttura" and "nome" columns. In this case, we kept all the cases where the apartment number or specific apartment name was not mentioned, selecting the correct occurrence only when both Booking and API explicitly mentioned the apartment name or number. For example:

Case of an unspecified number:

Denominazione struttura: la rocca
Nome: casa vacanza la rocca bilocale
API Indirizzo: via vassalini 27
Booking Indirizzo: via vassalini 27c

Furthermore, in cases where there was a match between "denominazione_struttura" and "nome" but not between addresses, we decided to keep the row only if the match was perfect. This choice depends on the fact that we are certain the API is updated every 15 days.

Case of different addresses but the same name:

Denominazione struttura: casa sofia
Nome: casa sofia
API Indirizzo: via dosso della benedizione 2
Booking Indirizzo: via dosso della benedizione 21/d

In the third case, where we didn't have at least one address available, we had to compare the locality (if present) and the name of the structure. Even in this case, we used a similarity index, but it didn't provide any further matches.

Finally, we merged the datasets created from the rows that provided a match, keeping the row indices from the original dataset. We performed a merge combining the two datasets, precisely using these indices between the new dataset and the API dataset.

The last step was to combine the duplicate columns for comune, indirizzo, cap, and località. If both cells were non-null, we gave priority to the API cell. In the case of null values, we ensured that the non-null value was retained.

5. Data Storage and Queries

For the storage phase, we agreed that MongoDB was the best choice. We leveraged the flexibility of its schema to store different documents without the need for a fixed table structure. Indeed, after the acquisition, cleaning, and integration operations, the resulting datasets were in perfect form to be saved as JSON documents.

The storage process involved transforming the files from CSV to JSON documents. The CSV format was used only during the initial stages for convenience and ease of row searching.

We created the "project_data_man" database, which contains three collections: restaurants, hotels, and ski resorts. Each collection consists of the 44 municipalities from the ski resorts dataset. We performed three separate loops, each with the goal of populating the respective collection with restaurants, hotels, and ski resorts from each municipality.

5.1. Queries

The following queries were performed in order to show some of the possible information that can be extracted from this database:

1. The following code listing is equivalent to the query: "what is the best restaurant in Aprica and which are the types of ski lifts in the area?"

```
pipeline = [
{'$match': {'_id':
    'aprica'}},
{'$unwind':
    '$restaurants'},
{'$match':
    {'restaurants.data_source':
    'tripadvisor'}},
{'$sort':
    {'restaurants.rating':
    -1}},
    {'$group': {
        '_id': '$_id',
        'best_restaurant':
            {'$first':
            '$restaurants'},
        'comprensori':
        {'$addToSet':
        '$comprensori.tipo_impianto'}
    }},
    {'$project': {
        '_id': 0,
        'best_restaurant':
            1,
        'comprensori': 1
    }}
    ]
```

2. This query returns the best hotel for each city:

```
        pipeline = [
    {'$unwind': '$hotels'},
    {'$sort':
        {'hotels.rating':
        -1}},
    {'$group': {
        '_id': '$_id',
        'best_hotel':
            {'$first':
            '$hotels'}
    }}
    ]
```

3. This query returns all the ski lifts types in Bagolino:

```
        pipeline = [
    {"$match": {"_id":
        "bagolino"}},
    {"$unwind":
        "$comprensori"},
    {"$group": {
        "_id":
"$comprensori.tipo_impianto"
    }},
    {"$project": {
        "_id": 0,
        "tipi_impianto":
            "$_id"
    }}
    ]
```

6. Data Quality

A fundamental concept in data management and analysis is Data Quality. The assessment of Data Quality involves various dimensions and metrics that are used to measure and evaluate the level of data quality. In our project, we have decided to assess our datasets based on three dimensions:

6.1. Accuracy

This dimension focuses on the correctness and precision of the data. It examines the extent to which the data corresponds to the actual values or expected standards. In our case, we encountered several challenges specifically in managing this dimension, particularly in certain specific fields such as restaurant and hotel names and addresses. The data cleaning phase primarily aimed to address these issues. Specifically, we considered correct localization and accurate ratings to be crucial for our purposes. Users searching for a good restaurant or hotel should be able to find it easily and assess if other users have regarded it highly. By utilizing specific regular expressions, we established fundamental patterns and evaluated the number of values that did not conform to these patterns. Regarding restaurants, we were able to achieve an accuracy rate of 63%. This result was calculated by taking the ratio of addresses that matched the patterns we proposed to the total number of observations in that column. The same process was applied to the hotel dataset, which obtained a slightly lower value of 50%. Having three distinct sources, we have noticed how scraping on Booking.com has produced more localized results compared to the area we provided as reference. Conversely, TripAdvisor covers a moderate range (approximately 25 km within our reference) and The Fork provides even broader coverage in the search (approximately 35 km within our reference). Thanks to these different ranges of action, we assume to have covered a good number of restaurants, taking into account that many may not have a subscription on these websites. Furthermore, we have conducted a thorough evaluation of the accuracy of the rating columns across all datasets. Unlike the addresses, which pose considerable challenges in terms of uniformity, we found that the accuracy levels in this case are considerably higher. Across all cleaned datasets, the accuracy surpasses 97%, affirming that all ratings consist of numeric values within a range of 1 to 5.

6.2. Completeness

This dimension evaluates the presence of all the required or expected data. It assesses whether any missing or incomplete data exists within the dataset. To assess completeness, we calculated the number of null values in the different datasets after the cleaning phase, before the merge, and after the merge, as the merge also helped fill in some gaps, in other words, perform enrichment. Regarding restaurants, particularly scraping from Google initially provided no null values, just like with scraping on The Fork. On the other hand, TripAdvisor provided a dataset with 187 null values out of a total of 7072, which is 2.6% of the total. After the Data Integration phase, the number of null values increased to 22%. We expected this increase as we knew that for many establishments, we would not find information on all 3 platforms, but maybe only on one or two at most. The same reasoning was applied for hotels. In that case, the API had 18467 null values out of 74109, which is 25% of null values in the entire dataset. Booking scraping provided 2456 null values out of 13068, which is 19% of the total. Just like with restaurants, the number of null values increased during the Data Integration phase, reaching 52% of the total. These numbers may seem far from a perfect result. However, not all variables carried the same weight within the datasets; some were only present to provide additional information. The ones most relevant to us, namely the names, addresses, and ratings, showed a very low rate of null values.

6.3. Consistency

Consistency refers to the coherence and conformity of the data across different sources, tables, or fields. It examines the uniformity and reliability of the data structure, format, and relationships. During the cleaning phase, we also managed this aspect. In the case of restaurants, we had three distinct datasets mostly with similar entries, but only a few could be enriched with each other. For instance, we could have created a unique rating or an average, but we chose not to do so in order to maintain the integrity and accuracy of the ratings from different sites, allowing users to compare them. The variables that could be enriched were the ones we directly used for merging, such as address, postal code, city, and name. The same reasoning applied to the two hotel datasets, which had many different entries compared to the restaurant datasets. Moreover, we performed duplicate removal on all datasets during the cleaning phase. By duplicate, we mean an entire duplicated row. Considering that scraping may provide duplicates due to time constraints and reading of the HTML page, it was probable to occur. In the restaurant dataset found through Google, we found 58 duplicates out of 993 rows. In the restaurant dataset found through The Fork, we found 719 duplicates out of 956 rows. In the restaurant dataset found through TripAdvisor, we found 104 duplicates out of 1003 rows. In the hotel dataset found through Booking, we found 11 duplicates out of 764 rows.

7. Conclusions

In conclusion, we can confidently state that we have successfully adhered to the various stages of the data lifecycle and created a consistent database aligned with the initial proposal.

This project has demonstrated that the data cleaning phase, particularly when starting from raw data, is often the longest and most complex. Identifying inconsistencies, errors, and the true nature of missing values can prove challenging. Thus, one of the most demanding challenges we encountered was striking a balance between data consistency and accuracy, while ensuring the efficiency of an automated system.

An important limitation is that the database cannot be automatically updated. Having a non-dynamic database only allows a potential user to check the presence of structures such as restaurants and hotels, but it does not provide information on whether they are available or open at a given moment. Therefore, if we were to think about an improvement, it would concern this aspect. In addition, if we were to consider another improvement for this project, one area to focus on would undoubtedly be enrichment. For instance, we could utilize TripAdvisor as a secondary data source for hotels, similar to what we did for restaurants. The decision to primarily use Booking was based on its comprehensive and reputable platform, well-known among users, hoteliers, and individuals. Finally, we could use the Google API to provide a visualization of the structures on the map, as we already have datasets with geographic location.

We believe that this database serves as a solid foundation for developing an app catering to ski enthusiasts who wish to have easy access to information about their favorite destinations.

By expanding the database and incorporating additional sources, along with the potential for user-generated content, we can create a valuable tool that offers comprehensive and up-to-date information for ski enthusiasts. Such an app would provide a convenient resource for individuals seeking relevant details about ski resorts, hotels, restaurants, and more.