Università degli Studi di Milano Bicocca

Dipartimento di Informatica

Corso di Laurea Magistrale - Data Science

A Comparative Analysis of Weather Forecast Websites

Borruso William Joseph
Galli Luca
Ronchi Davide

Academic Year 2022-2023

Contents

Abstract

Our *Data Management* project focuses on the collection, preparation and analysis of hourly weather forecast data from three of the main weather websites for all the provinces of the Lombardy region. In particular, hourly weather forecasts for the next day were obtained daily from each selected website through web scraping. The collected data was subjected to rigorous cleaning and preprocessing to ensure consistency, reliability and usability. At the end of the collection period, we also collected a posteriori measurements. All the obtained data were aggregated and stored in an SQL relational database.

The project aims to demonstrate the significance of effective data management and preprocessing in facilitating meaningful insights from raw data, and sets the research question of discovering which website performed better during the collection period.

Chapter 1

Introduction

During the month of August 2023, from the 9th to the 22nd, we collected daily weather forecast from three websites, for a collection period of 2 weeks. We chose three of the main websites for weather predictions used in Italy, namely *ilMeteo.it*, *Meteo.it* and *3bMeteo.com*.

To collect and store the data we wrote a web scraping script able to collect the hourly forecast of the following day. Web scraping is a technique used to extract information and data from websites. It involves automating the process of fetching and parsing web pages to extract relevant data for various purposes, such as analysis, research, or storage, when no APIs or downloadable data are available. We chose to collect the data of the twelve Lombardy provinces: Bergamo, Brescia, Como, Cremona, Lecco, Lodi, Mantova, Milano, Monza and Brianza, Pavia, Sondrio and Varese.

The project is divided into five main parts: data collection, data preparation, data augmentation, data quality inspection, querying of the collected data and comparison with historical data.

By choosing this topic we were able to perform three of the main data acquisition techniques (web scraping, data download and the use of APIs) on structurally different sources, and structure a relational database from the data, thereby enabling comprehensive analysis and production of meaningful insights for our research objectives.

1.1 Software and Technologies

During our work on this project, two main technologies were used: Python and PostgreSQL 15.

Python is a high-level, simple and readable programming language. The many libraries and frameworks it offers allow it to be widely used in various fields, among which web scraping and data manipulation tasks.

In particular, we list the main libraries we used:

- **Selenium:** *Selenium* is a popular open-source framework for automating web browsers. It allows for interaction with web pages and to perform various tasks such as clicking buttons and navigating between pages.
  - **Chrome WebDriver:** the *Chrome WebDriver* is a part of Selenium and acts as a bridge between the Python code and the Google Chrome web browser.
- **Psycopg2:** *Psycopg2* is a popular Python library for interacting with PostgreSQL databases. This library provides an interface for connecting to PostgreSQL databases, executing SQL queries, and managing database connections.
- **Requests:** *Requests* is a popular Python library used for making HTTP requests, including GET requests. It simplifies the process of sending HTTP requests and handling the responses.
- **Other libraries:** we also used other libraries for specific tasks and data manipulation, such as *Time*, *Datetime*, *Re* and *Pandas*.

PostgreSQL is an open-source relational database management system (RDBMS) known for its robustness, extensibility, and compliance with SQL standards. As of when we developed our project, we used version 15.

Chapter 2

Data Collection

We wrote our web scraping scripts in Python, utilizing libraries such as Selenium and Pandas. The Selenium library is a popular open-source framework primarily used for automating web browser interactions, which came in useful for our purposes of scraping iFrames, dynamic tables, and simulating user clicks. The Pandas library is a widely-used open-source Python library that provides powerful data manipulation and analysis tools, so we used it when we created the DataFrames for the collected data and performed data cleaning.

2.1 Environment Setup

The first task we did in our project was to setup our environment. We started by importing the libraries and setting up Selenium, in particular we installed the Chrome driver and used it with Selenium.

2.2 Scraping

2.2.1 meteo.it

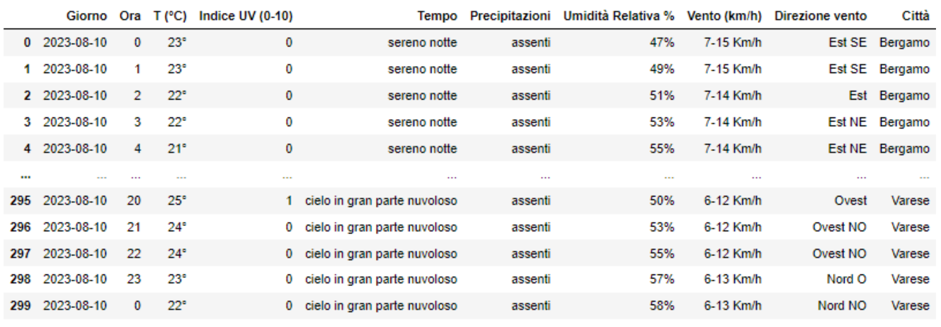
Our script starts by scraping weather data from the provided URLs, extracts relevant information for each city's forecast, and stores the extracted data in a list. We then process the scraped weather data by converting data types, filtering rows based on indices, setting a multi-level index, concatenating the data with previously saved data (if available), and saving the resulting DataFrame to a pickle file.

The process can be summarized as follows:

- There are two lists: *url_meteoit* contains URLs for different cities' weather forecasts, and *province_lombardia* contains the names of the respective cities.
- The script iterates through each URL:
  - The URL, corresponding city name and the next day's date are extracted for the current iteration.
  - The web driver opens the URL in a Chrome browser window. Weather data for the selected day and city are shown.
  - The script locates specific web elements containing tabular weather data such as time, temperature, weather description (extracted from weather icons' *alt* attributes), wind speed, and city name. In this case the table is shown as a unordered bulleted list *ul*, and columns are organized in three main horizontal *div* containers.
  - In each loop iteration we extract a table row, append date and city name, and store it in a list of rows (*table_row*) for the current city. Empty cells are replaced by verbose placeholders.
  - The *table_row* elements are iteratively appended to the *table_new* list, which will contain, for every city, all the data for that day by the end of the loop.
- After processing all URLs, the web driver is closed.

Finally, we process the scraped weather data:

- We create a DataFrame with the collected weather information.



	Giorno	Ora	T (°C)	Indice UV (0-10)	Tempo	Precipitazioni	Umidità Relativa %	Vento (km/h)	Direzione vento	Città
0	2023-08-10	0	23°	0	sereno notte	assenti	47%	7-15 Km/h	Est SE	Bergamo
1	2023-08-10	1	23°	0	sereno notte	assenti	49%	7-15 Km/h	Est SE	Bergamo
2	2023-08-10	2	22°	0	sereno notte	assenti	51%	7-14 Km/h	Est	Bergamo
3	2023-08-10	3	22°	0	sereno notte	assenti	53%	7-14 Km/h	Est NE	Bergamo
4	2023-08-10	4	21°	0	sereno notte	assenti	55%	7-14 Km/h	Est NE	Bergamo
...	...	...	...	...	...	...	...	...	...	...
295	2023-08-10	20	25°	1	cielo in gran parte nuvoloso	assenti	50%	6-12 Km/h	Ovest	Varese
296	2023-08-10	21	24°	0	cielo in gran parte nuvoloso	assenti	53%	6-12 Km/h	Ovest NO	Varese
297	2023-08-10	22	24°	0	cielo in gran parte nuvoloso	assenti	55%	6-12 Km/h	Ovest NO	Varese
298	2023-08-10	23	23°	0	cielo in gran parte nuvoloso	assenti	57%	6-13 Km/h	Nord O	Varese
299	2023-08-10	0	22°	0	cielo in gran parte nuvoloso	assenti	58%	6-13 Km/h	Nord NO	Varese

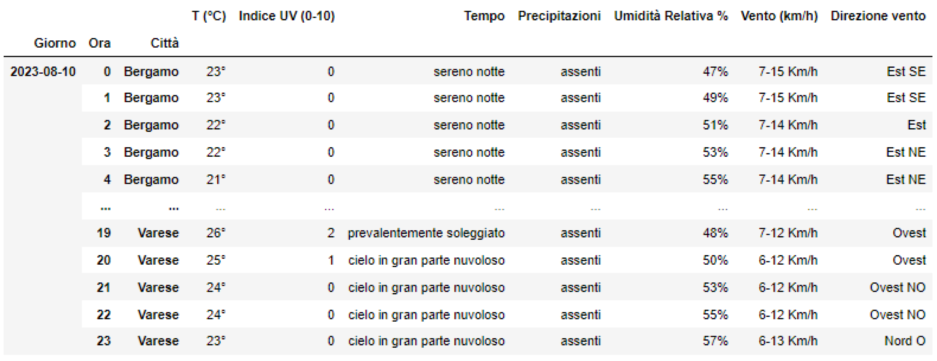
- We filter out not needed rows, such as inconsistent rows and repeated rows (e.g. 00:00 forecast shown twice)

- We set a multi-level index for the DataFrame, which includes date, time and selected city.

- We concatenate the DataFrame with the previous days aggregated DataFrame (if it already exists) from a pickle file.

- Finally, we save the updated DataFrame locally as a pickle file.

From *meteo.it*, our DataFrame is composed by the columns '*Giorno*' (Day), '*Ora*' (Time), '*Città*' (City) that are used as indexes, and the information collected regards temperature (gathered in °C), UV index (represented on a scale from 0 to 10), the weather conditions, the amount (in mm) of rainfall, the percentage of relative humidity, the intensity (represented in km/h) and the direction of the wind.



			T (°C)	Indice UV (0-10)	Tempo	Precipitazioni	Umidità Relativa %	Vento (km/h)	Direzione vento
Giorno	Ora	Città							
2023-08-10	0	Bergamo	23°	0	sereno notte	assenti	47%	7-15 Km/h	Est SE
	1	Bergamo	23°	0	sereno notte	assenti	49%	7-15 Km/h	Est SE
	2	Bergamo	22°	0	sereno notte	assenti	51%	7-14 Km/h	Est
	3	Bergamo	22°	0	sereno notte	assenti	53%	7-14 Km/h	Est NE
	4	Bergamo	21°	0	sereno notte	assenti	55%	7-14 Km/h	Est NE
	...	...	...	...	...	...	...	...	...
	19	Varese	26°	2	prevalentemente soleggiato	assenti	48%	7-12 Km/h	Ovest
	20	Varese	25°	1	cielo in gran parte nuvoloso	assenti	50%	6-12 Km/h	Ovest
	21	Varese	24°	0	cielo in gran parte nuvoloso	assenti	53%	6-12 Km/h	Ovest NO
	22	Varese	24°	0	cielo in gran parte nuvoloso	assenti	55%	6-12 Km/h	Ovest NO
	23	Varese	23°	0	cielo in gran parte nuvoloso	assenti	57%	6-13 Km/h	Nord O

2.2.2 iLMeteo

Our script now uses a different website as a source for weather forecasts (*ilmeteo.it*). The script still iterates through the provided URLs, extracts weather information from the web pages using Selenium, and organizes the collected data into a structured format.

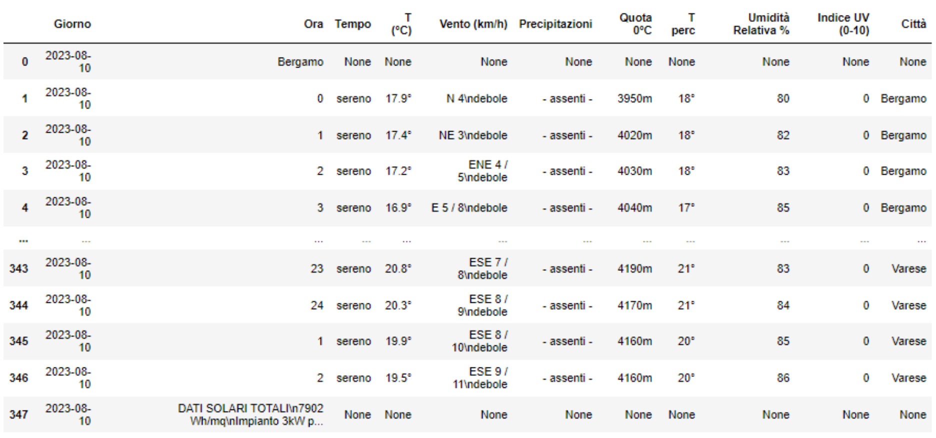
Again, the scripts starts by defining two lists:

- *url_ilmeteo*: the list of URLs corresponding to the iFrames containing weather forecasts from the specific website and date.

- *province_lombardia*: the same list of city names.

The table_new list is initialized again. This list will hold the scraped weather data. We also set up a web driver using Chrome from the Selenium library. The code then iterates through each URL:

- The URL, corresponding city name and the next day's date are extracted for the current iteration.
- The web driver opens the URL.
- The script locates a specific HTML element (*table* tag) with the class name *"datatable"*, and finds all the rows within the table.
- For each table row, it creates an empty list (*table_row*) to hold the data for that row and appends the specific city and date to it. It then finds all the table columns within the row and extracts the text from each column.
- The extracted data is appended to the *table_row* list, along with the city name.
- The table_row list is appended to the *table_new* list at every loop iteration, forming a list of lists containing weather data for different cities and times.
- Finally, after processing all URLs, the web driver is closed.



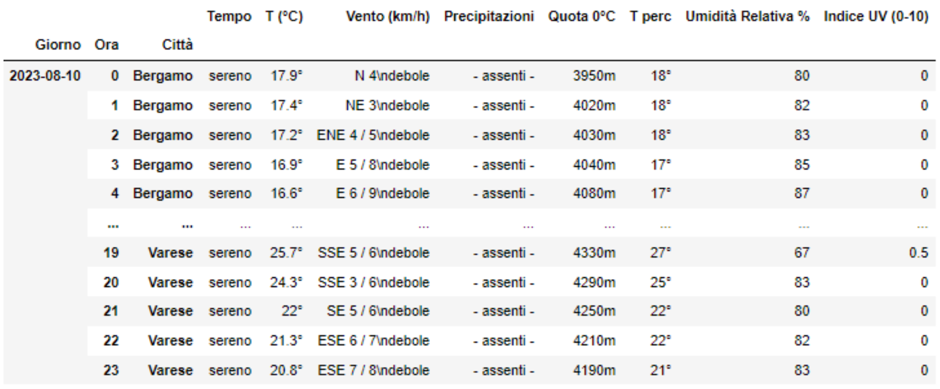
	Giorno	Ora	Tempo	T (°C)	Vento (km/h)	Precipitazioni	Quota 0°C	T perc	Umidità Relativa %	Indice UV (0-10)	Città
0	2023-08-10	Bergamo	None	None	None	None	None	None	None	None	None
1	2023-08-10	0	sereno	17.9°	N 4\ndebole	- assenti -	3950m	18°	80	0	Bergamo
2	2023-08-10	1	sereno	17.4°	NE 3\ndebole	- assenti -	4020m	18°	82	0	Bergamo
3	2023-08-10	2	sereno	17.2°	ENE 4 / 5\ndebole	- assenti -	4030m	18°	83	0	Bergamo
4	2023-08-10	3	sereno	16.9°	E 5 / 8\ndebole	- assenti -	4040m	17°	85	0	Bergamo
...	...	...	...	...	...	...	...	...	...	...	...
343	2023-08-10	23	sereno	20.8°	ESE 7 / 8\ndebole	- assenti -	4190m	21°	83	0	Varese
344	2023-08-10	24	sereno	20.3°	ESE 8 / 9\ndebole	- assenti -	4170m	21°	84	0	Varese
345	2023-08-10	1	sereno	19.9°	ESE 8 / 10\ndebole	- assenti -	4160m	20°	85	0	Varese
346	2023-08-10	2	sereno	19.5°	ESE 9 / 11\ndebole	- assenti -	4160m	20°	86	0	Varese
347	2023-08-10	DATI SOLARI TOTALI\n7902 Wh/mq\nImpianto 3kW p...	None	None	None	None	None	None	None	None	None

Like the previous script, we process and start cleaning the scraped weather data by:

- Adjusting column names and reordering the columns.
- Removing specific rows based on a defined step value and indices.
- Rearranging indexes and setting the same multi-level index chosen for the previous website DataFrame.

- Attempting to concatenate the processed DataFrame with an existing one (if available) and saving the updated DataFrame as a pickle file.

From iLMeteo.it our DataFrame is initially composed by Day, Time and City which are again used as the indexes, and then we collect data regarding weather conditions, temperature and perceived temperature (both registered as °C), wind information, rainfall amounts, relative humidity, UV index and 0°C altitude.



			Tempo	T (°C)	Vento (km/h)	Precipitazioni	Quota 0°C	T perc	Umidità Relativa %	Indice UV (0-10)
Giorno	Ora	Città								
2023-08-10	0	Bergamo	sereno	17.9°	N 4\ndebole	- assenti -	3950m	18°	80	0
	1	Bergamo	sereno	17.4°	NE 3\ndebole	- assenti -	4020m	18°	82	0
	2	Bergamo	sereno	17.2°	ENE 4 / 5\ndebole	- assenti -	4030m	18°	83	0
	3	Bergamo	sereno	16.9°	E 5 / 8\ndebole	- assenti -	4040m	17°	85	0
	4	Bergamo	sereno	16.6°	E 6 / 9\ndebole	- assenti -	4080m	17°	87	0
	...	...	...	...	...	...	...	...	...	...
	19	Varese	sereno	25.7°	SSE 5 / 6\ndebole	- assenti -	4330m	27°	67	0.5
	20	Varese	sereno	24.3°	SSE 3 / 6\ndebole	- assenti -	4290m	25°	83	0
	21	Varese	sereno	22°	SE 5 / 6\ndebole	- assenti -	4250m	22°	80	0
	22	Varese	sereno	21.3°	ESE 6 / 7\ndebole	- assenti -	4210m	22°	82	0
	23	Varese	sereno	20.8°	ESE 7 / 8\ndebole	- assenti -	4190m	21°	83	0

2.2.3 3B Meteo

For our third website that we used to obtain weather forecast, we similarly developed a script that scrapes multiple web pages, extracts different weather attributes, and organizes the data into separate DataFrames for two different sets of information: columns were in fact separated in two sets and showing alternately when the button *Other data* was clicked.

As usual, we first define the two lists:

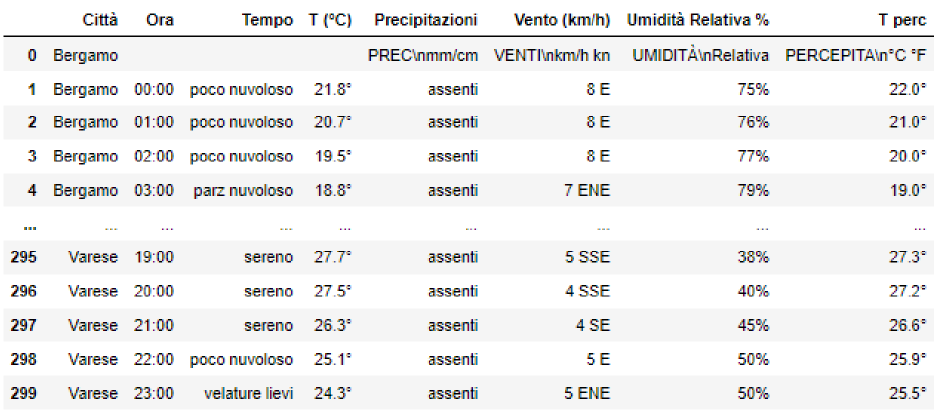
- *url_3b*: the list of URLs corresponding to detailed hourly weather forecasts for different cities.
- *province_lombardia*: the list of city names.

We then initialize a web driver using Chrome from the Selenium library, and start iterating through each URL:

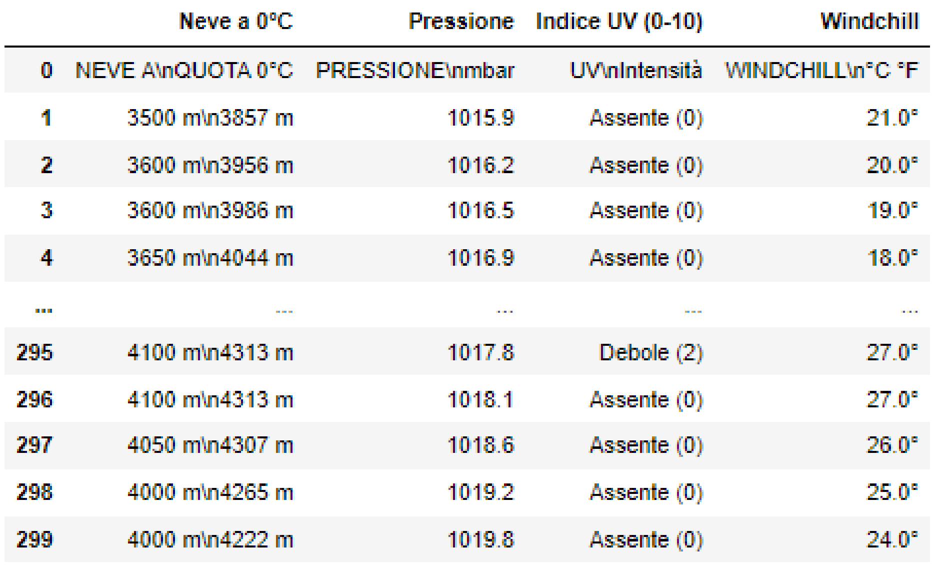
- The URL, corresponding city name and the next day's date are extracted for the current iteration.
- The web driver (*driver*) opens the URL.
- We handle any cookie banner that might be present on the page by accepting them through a simulated click.

- We locate a specific HTML element (table) with the class name *'table-previsioni-ora'*.
- We further find rows within the table and iterate through each row to extract weather details using different CSS selectors for different attributes.
- The extracted data is stored in dictionaries for two sets of weather attributes for each row. We also append specific date and city name for each row.

We then append the collected data for each city to two separate lists, where each list corresponds to one set of weather attributes (this separation was made by the website itself). After processing all URLs, the web driver is closed and finally the collected data is organized into two separate Pandas DataFrames.



	Città	Ora	Tempo	T (°C)	Precipitazioni	Vento (km/h)	Umidità Relativa %	T perc
0	Bergamo				PREC\nmm/cm	VENTI\nkm/h kn	UMIDITÀ\nRelativa	PERCEPITA\n°C °F
1	Bergamo	00:00	poco nuvoloso	21.8°	assenti	8 E	75%	22.0°
2	Bergamo	01:00	poco nuvoloso	20.7°	assenti	8 E	76%	21.0°
3	Bergamo	02:00	poco nuvoloso	19.5°	assenti	8 E	77%	20.0°
4	Bergamo	03:00	parz nuvoloso	18.8°	assenti	7 ENE	79%	19.0°
...	...	...	...	...	...	...	...	...
295	Varese	19:00	sereno	27.7°	assenti	5 SSE	38%	27.3°
296	Varese	20:00	sereno	27.5°	assenti	4 SSE	40%	27.2°
297	Varese	21:00	sereno	26.3°	assenti	4 SE	45%	26.6°
298	Varese	22:00	poco nuvoloso	25.1°	assenti	5 E	50%	25.9°
299	Varese	23:00	velature lievi	24.3°	assenti	5 ENE	50%	25.5°

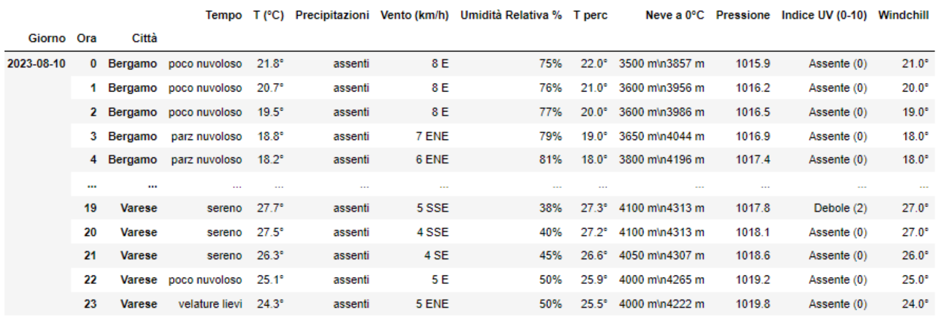


	Neve a 0°C	Pressione	Indice UV (0-10)	Windchill
0	NEVE A\nQUOTA 0°C	PRESSIONE\nmbar	UV\nIntensità	WINDCHILL\n°C °F
1	3500 m\n3857 m	1015.9	Assente (0)	21.0°
2	3600 m\n3956 m	1016.2	Assente (0)	20.0°
3	3600 m\n3986 m	1016.5	Assente (0)	19.0°
4	3650 m\n4044 m	1016.9	Assente (0)	18.0°
...	...	...	...	...
295	4100 m\n4313 m	1017.8	Debole (2)	27.0°
296	4100 m\n4313 m	1018.1	Assente (0)	27.0°
297	4050 m\n4307 m	1018.6	Assente (0)	26.0°
298	4000 m\n4265 m	1019.2	Assente (0)	25.0°
299	4000 m\n4222 m	1019.8	Assente (0)	24.0°

Finally, just like for the first two websites, an initial phase of processing and cleaning is performed:

- We filter out rows with undesired values in specific columns (e.g. additional headers, duplicates of 00:00, forecast of nightly hours beyond 00:00).
- We merge the two DataFrames horizontally, to obtain a single complete DataFrame.
- We convert the time values to integers and set a multi-level index (always Day, Time and City).
- We append the processed data to an existing dataset (if available) and save the updated DataFrame as a pickle file.

From the data scraped from 3B Meteo, our DataFrame is composed by Day, Time, City, which we use multi-level index. We then collect data regarding temperatures, rainfall, wind, humidity, snow, pressure, UV index and windchill.



			Tempo	T (°C)	Precipitazioni	Vento (km/h)	Umidità Relativa %	T perc	Neve a 0°C	Pressione	Indice UV (0-10)	Windchill
Giorno	**Ora**	**Città**										
2023-08-10	**0**	**Bergamo**	poco nuvoloso	21.8°	assenti	8 E	75%	22.0°	3500 m\n3857 m	1015.9	Assente (0)	21.0°
	1	**Bergamo**	poco nuvoloso	20.7°	assenti	8 E	76%	21.0°	3600 m\n3956 m	1016.2	Assente (0)	20.0°
	2	**Bergamo**	poco nuvoloso	19.5°	assenti	8 E	77%	20.0°	3600 m\n3986 m	1016.5	Assente (0)	19.0°
	3	**Bergamo**	parz nuvoloso	18.8°	assenti	7 ENE	79%	19.0°	3650 m\n4044 m	1016.9	Assente (0)	18.0°
	4	**Bergamo**	parz nuvoloso	18.2°	assenti	6 ENE	81%	18.0°	3800 m\n4196 m	1017.4	Assente (0)	18.0°
	...	**...**	...	...	...	...	...	...	...	...	...	...
	19	**Varese**	sereno	27.7°	assenti	5 SSE	38%	27.3°	4100 m\n4313 m	1017.8	Debole (2)	27.0°
	20	**Varese**	sereno	27.5°	assenti	4 SSE	40%	27.2°	4100 m\n4313 m	1018.1	Assente (0)	27.0°
	21	**Varese**	sereno	26.3°	assenti	4 SE	45%	26.6°	4050 m\n4307 m	1018.6	Assente (0)	26.0°
	22	**Varese**	poco nuvoloso	25.1°	assenti	5 E	50%	25.9°	4000 m\n4265 m	1019.2	Assente (0)	25.0°
	23	**Varese**	velature lievi	24.3°	assenti	5 ENE	50%	25.5°	4000 m\n4222 m	1019.8	Assente (0)	24.0°

2.3 Download

2.3.1 Historical data

Monthly historical weather data are provided by *ilMeteo* as *csv* files. These files contain the day-by-day a posteriori weather measurements for every city in Italy. Comparing forecast data with a posteriori measurements will allow us to fulfill our research question of understanding which website performed better in weather predictions. Note that, being the weather forecasts only provided in an hour-by-hour format, some calculations will be performed to estimate daily forecasts and allow comparison.

The specific *csv* files were downloaded from ilMeteo as separate CSVs for every province of Lombardy. We stored them locally and imported them into our notebook as DataFrames.

Once imported, they were all concatenated, to obtain a single dataframe with historical data for all selected cities.

2.4 API

2.4.1 Air Quality data

We decided to enrich our selected cities with valuable data, so we used an Application Programming Interface (API) to enrich them with data regarding air quality. The API we used was provided by *weatherbit.io*. This is useful since it allows us to understand associations between air quality and weather data and phenomenons.

To access *weatherbit.io* APIs, we registered to the website with a new account and started a 7-days free trial: after that, we were provided with an API key and an endpoint to use in order to retrieve data through HTTP GET requests.

To do that, we used Python's *requests* library to send GET requests to the *Weatherbit* API endpoint to gain information regarding air quality data at specified coordinates, and stored the API responses locally in a *.txt* file. The coordinates, provided by the website itself, indicated the Lombardy provinces we have been studying during this project, so Bergamo, Brescia, Como, Cremona, Lecco, Lodi, Mantova, Milano, Monza, Pavia, Sondrio, Varese.

Details about the collected data are discussed in Section 4.2.

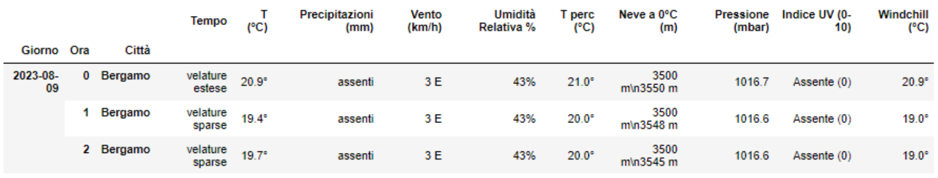
Chapter 3

Data Preparation

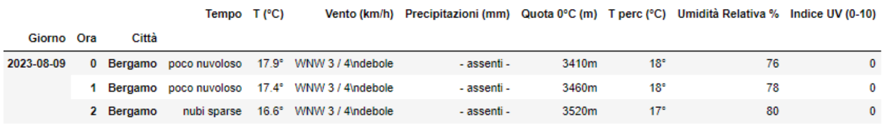
3.1 Data Cleaning

As data collected from the websites was not in the format we desired, we undertook a phase of feature engineering and data cleaning to improve our data quality and make columns across the three separate DataFrames consistent.

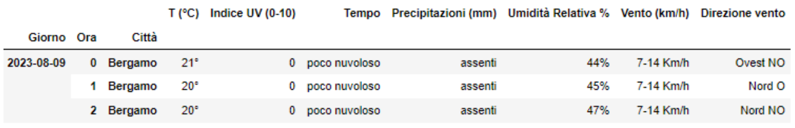
First of all, we simply renamed some of the DataFrames columns to ensure consistency, making it easier to combine and analyze them together later on.



			Tempo	T (°C)	Precipitazioni (mm)	Vento (km/h)	Umidità Relativa %	T perc (°C)	Neve a 0°C (m)	Pressione (mbar)	Indice UV (0-10)	Windchill (°C)
Giorno	**Ora**	**Città**										
2023-08-09	**0**	**Bergamo**	velature estese	20.9°	assenti	3 E	43%	21.0°	3500 m\n3550 m	1016.7	Assente (0)	20.9°
	1	**Bergamo**	velature sparse	19.4°	assenti	3 E	43%	20.0°	3500 m\n3548 m	1016.6	Assente (0)	19.0°
	2	**Bergamo**	velature sparse	19.7°	assenti	3 E	43%	20.0°	3500 m\n3545 m	1016.6	Assente (0)	19.0°



			Tempo	T (°C)	Vento (km/h)	Precipitazioni (mm)	Quota 0°C (m)	T perc (°C)	Umidità Relativa %	Indice UV (0-10)
Giorno	**Ora**	**Città**								
2023-08-09	**0**	**Bergamo**	poco nuvoloso	17.9°	WNW 3 / 4\ndebole	- assenti -	3410m	18°	76	0
	1	**Bergamo**	poco nuvoloso	17.4°	WNW 3 / 4\ndebole	- assenti -	3460m	18°	78	0
	2	**Bergamo**	nubi sparse	16.6°	WNW 3 / 4\ndebole	- assenti -	3520m	17°	80	0

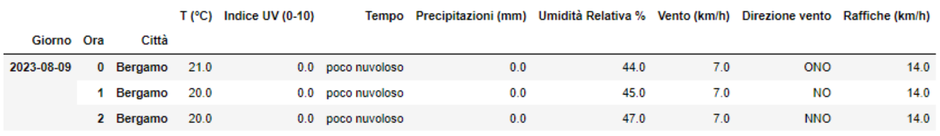


			T (°C)	Indice UV (0-10)	Tempo	Precipitazioni (mm)	Umidità Relativa %	Vento (km/h)	Direzione vento
Giorno	**Ora**	**Città**							
2023-08-09	**0**	**Bergamo**	21°	0	poco nuvoloso	assenti	44%	7-14 Km/h	Ovest NO
	1	**Bergamo**	20°	0	poco nuvoloso	assenti	45%	7-14 Km/h	Nord O
	2	**Bergamo**	20°	0	poco nuvoloso	assenti	47%	7-14 Km/h	Nord NO

3.1.1 meteo.it

In this phase we perform thorough data cleaning on the DataFrame obtained by the data collected from the *meteo.it* website. Initial cleaning has already been performed, consisting mainly of removing unnecessary rows and setting up the structure of our DataFrame. However, we now operate on types and columns. Among the operations we perform, we remove the units of measurement from the columns and adjust their data types, ensuring that format and measurement units are consistent within and across all tables. Overall, we deal with units, data types, symbols, text formatting and column rearrangements.

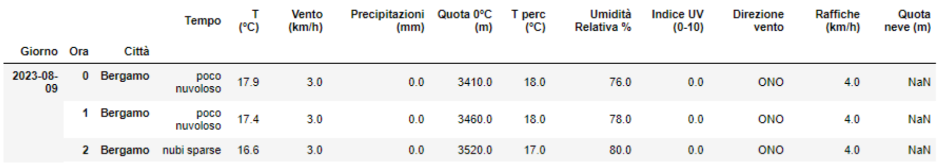
For example, Wind column was separated in Wind (*Vento*) and possible Wind Gusts (*Raffiche*), if present, by splitting over the "-" character, removing the unit of measurement and eventually replacing null values on the Raffiche column with a consistent placeholder [1].



			T (°C)	Indice UV (0-10)	Tempo	Precipitazioni (mm)	Umidità Relativa %	Vento (km/h)	Direzione vento	Raffiche (km/h)
Giorno	Ora	Città								
2023-08-09	0	Bergamo	21.0	0.0	poco nuvoloso	0.0	44.0	7.0	ONO	14.0
	1	Bergamo	20.0	0.0	poco nuvoloso	0.0	45.0	7.0	NO	14.0
	2	Bergamo	20.0	0.0	poco nuvoloso	0.0	47.0	7.0	NNO	14.0

3.1.2 iLMeteo

We then perform similar data cleaning on the data collected from the *iL-Meteo* website. We remove symbols from specific columns, adjust column formats and replace values and strings. By cleaning and formatting this second DataFrame in a standard way, we ensure consistency and accurate processing of the data.

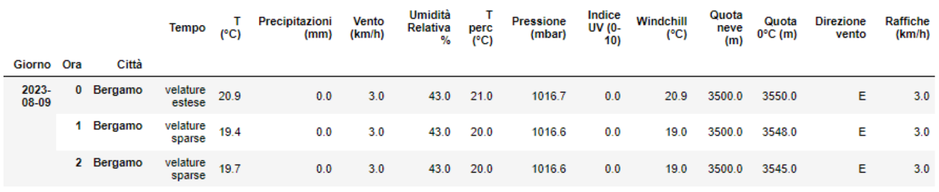


			Tempo	T (°C)	Vento (km/h)	Precipitazioni (mm)	Quota 0°C (m)	T perc (°C)	Umidità Relativa %	Indice UV (0-10)	Direzione vento	Raffiche (km/h)	Quota neve (m)
Giorno	Ora	Città											
2023-08-09	0	Bergamo	poco nuvoloso	17.9	3.0	0.0	3410.0	18.0	76.0	0.0	ONO	4.0	NaN
	1	Bergamo	poco nuvoloso	17.4	3.0	0.0	3460.0	18.0	78.0	0.0	ONO	4.0	NaN
	2	Bergamo	nubi sparse	16.6	3.0	0.0	3520.0	17.0	80.0	0.0	ONO	4.0	NaN

3.1.3 3B Meteo

Finally, we perform a series of data cleaning and type conversions on the DataFrame obtained by the data scraped from the *3B Meteo* website. For the third time, we handle units, symbols, and formatting issues in the DataFrame columns as needed to ensure consistency with the DataFrames collected from the other two websites.

[1] Other similar operations won't be discussed in detail in the report, but they can be consulted on the respective Python notebook.



			Tempo	T (°C)	Precipitazioni (mm)	Vento (km/h)	Umidità Relativa %	T perc (°C)	Pressione (mbar)	Indice UV (0-10)	Windchill (°C)	Quota neve (m)	Quota 0°C (m)	Direzione vento	Raffiche (km/h)
Giorno	Ora	Città													
2023-08-09	0	Bergamo	velature estese	20.9	0.0	3.0	43.0	21.0	1016.7	0.0	20.9	3500.0	3550.0	E	3.0
	1	Bergamo	velature sparse	19.4	0.0	3.0	43.0	20.0	1016.6	0.0	19.0	3500.0	3548.0	E	3.0
	2	Bergamo	velature sparse	19.7	0.0	3.0	43.0	20.0	1016.6	0.0	19.0	3500.0	3545.0	E	3.0

3.1.4 Historical data

Being historical data downloaded as .csv files, they are already in a tabular, cleaned and consistent format. However, we performed feature selection, similarly to the previous cases, and renamed the columns in a consistent way.

Also, we converted dates from *dd/mm/YYYY* format to *YYYY-mm-dd* format.

Finally, we observed that the daily historical data shows a list of all the actually happened phenomenon during the day, separated by a space. We decided to store them into an array, which becomes an empty array in case of a sunny or cloudy day. As introduced in Subsection 3.2.1, this will be treated as a *fenom[] array*.

3.2 Data Processing

3.2.1 Categorical Variables

We have a particular feature, which is *Tempo* (weather), that describes the weather of that hour in a colloquial way (e.g. quite sunny, cloudy with weak rainfalls, mostly cloudy, foggy...): we can have a wide variety of different descriptions, which are also different for every website (since they don't follow any standard), and we are not provided with the full list of possible strings. As it is, the values of this variable are not comparable.

Since this is an important piece of information, we decided to maintain it as a more general categorical variable: in the database, it will be treated as a *enum* variable that we called *fenom*, which can take the 4 values *'nebbia', 'neve', 'pioggia', 'temporale'* ('fog', 'snow', 'rain', 'thunderstorm').

To map every string obtained at the end of the data collection process to one of these 4 categories, we first listed all possible unique strings taken once for all three websites. After that, with the help of an external NLP AI tool, followed by our supervision, we created a mapping dictionary that maps all of the strings to one of the four classes (or, in case of a sunny or cloudy day, to the null placeholder *None*).

3.2.2 Feature Selection

As every DataFrame offers diverse categories of information, we decide to perform feature selection to select useful information present in every DataFrame that can be compared to historical data in the end phase of our project. In this phase, we not only perform feature selection, but also add a column to indicate the source website of each row of the dataset.

The columns we selected include '*Sito*', '*Città*', '*Giorno*', '*Ora*', '*T (°C)*', '*Precipitazioni (mm)*', '*Umidità Relativa %*', '*Vento (km/h)*', and '*Tempo*', and the index of each of the final DataFrames is set to a multi-index composed of '*Sito*', '*Città*', '*Giorno*' and '*Ora*' columns. We finally combine the three DataFrames together to have a final DataFrame composed by data from all three original sources, after having checked for duplicates.

Since the '*Tempo*' column presented different values across each website, we standardized all of them to ensure consistency.

This section was dedicated to standardizing, combining, and processing the weather data we collected from different sources (*iLMeteo*, *3B Meteo*, *meteo.it*) into a unified format.

Chapter 4

Data Augmentation

4.1 Data Aggregation

Our research questions can only be answered if all collected data are comparable between each other. In data preparation, especially data cleaning, we took care that data from different tables were consistent and comparable in format, type, measurement unit, language and so on. This will allow us to join their structures with database queries and compare data to extract information.

However, both the forecast data and historical data went through a phase of data aggregation, since we had multiple tables representing data of the same nature.

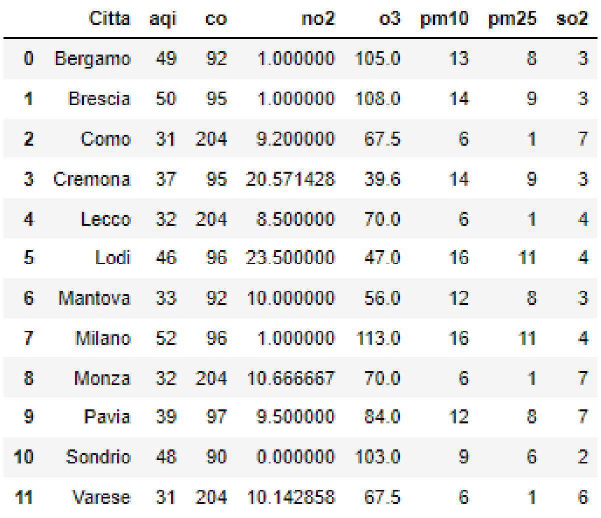
Starting from forecast data, we concatenated the three DataFrames representing two weeks of hourly weather from the three selected websites, making sure that primary and foreign keys constraints were not violated, and that we didn't produce useless redundancies, inconsistencies or data loss. We obtain a single forecast data table, where each row identifies univocal forecast metrics for a specific day, hour, city and website.

Regarding historical data, we also concatenated all the DataFrames that were separated per city and month of collection, avoiding integrity or quality losses. We obtain a single historical data table, where each row identifies univocal measure metrics for a specific day, city and website.

4.2 Data Enrichment

As already introduced, we used data provided by *weatherbit.io* APIs to enrich the cities under our study with data regarding air quality, to potentially understand associations between air quality and weather data and phenomenons.

To perform data enrichment, we read the air quality data from the *.txt* file where we previously stored the API's responses, parse it into a Python object, transform and normalize the data, turn them into a DataFrame, and perform data cleaning to retain relevant columns. The resulting DataFrame contains relevant air quality information for our various cities. The columns we deemed relevant ended up being: '*Citta*', '*aqi*', '*co*', '*no2*', '*o3*', '*pm10*', '*pm25*', '*so2*', referring, respectively, to Air Quality Index [US - EPA standard 0 - +500], Concentration of carbon monoxide (µg/m^3), Concentration of surface NO2 (µg/m^3), Concentration of surface O3 (µg/m^3), Concentration of particulate matter $<$ 10 microns (µg/m^3), Concentration of particulate matter $<$ 2.5 microns (µg/m^3), Concentration of surface SO2 (µg/m^3).



	Citta	aqi	co	no2	o3	pm10	pm25	so2
0	Bergamo	49	92	1.000000	105.0	13	8	3
1	Brescia	50	95	1.000000	108.0	14	9	3
2	Como	31	204	9.200000	67.5	6	1	7
3	Cremona	37	95	20.571428	39.6	14	9	3
4	Lecco	32	204	8.500000	70.0	6	1	4
5	Lodi	46	96	23.500000	47.0	16	11	4
6	Mantova	33	92	10.000000	56.0	12	8	3
7	Milano	52	96	1.000000	113.0	16	11	4
8	Monza	32	204	10.666667	70.0	6	1	7
9	Pavia	39	97	9.500000	84.0	12	8	7
10	Sondrio	48	90	0.000000	103.0	9	6	2
11	Varese	31	204	10.142858	67.5	6	1	6

Figure 4.1: *N.B.: For all the metrics, lower values correspond to cleaner and healthier air, and therefore a higher quality.*

We then connect to the same PostgreSQL database and insert the newly acquired data to the '*Citta*' table, also committing changes to the database to ensure data integrity. Finally, we insert the predefined website data into the '*Siti*' table, and ensure that duplicate entries are ignored.

Chapter 5

Data Quality Inspection

Since checking for data quality is a critical task in any data analysis process, we perform some inspections on the columns of the DataFrame.

We start with string variables, and our results show that strings are all correct and of equal number, and we have no null strings. '*Tempo*' is also correct after the "categorization". The majority of rows has no category, meaning there are no phenomenons (weather is sunny or cloudy). We've had no snow ("*Neve*") during the collection time, as expected in north-Italy in August. We also perform a check on the integer column '*Ora*', and we check the range of values of float columns ('*T (°C)*', '*Precipitazioni (mm)*' and '*Umidità Relativa %*'). All columns show realistic and plausible values (for example, our percentage values are between 0 and 100).

Our forecasts DataFrame presents no null values, indicating a good collection phase and an high level of data completeness.

Furthermore, we produced no duplicates at the end of the aggregation phase.

The historical DataFrame, instead, has some rows with missing values: specifically, 13 rows between Cremona, Brescia and Lodi take null values for all the columns, which were not stored or maybe not collected at all. For Cremona, we filled the 6 missing rows with values taken from a city very close to Cremona called Bonemerse: we assumed and observed that the predictions are very similar due to their proximity and similar altitude. We did the same for the 1 missing row for Brescia by using data from Montichiari, while we decided to discard Sondrio and Pavia for the complete lack of data of both Sondrio and nearby cities, and also Lodi, for which we found no data from nearby cities with similar altitude to complete the data.

Chapter 6

Database Creation

After data preparation and augmentation, we focused on building a relational database using PostgreSQL, taking care that it was consistent with the structure of the data, but at the same time efficient for the operations we are going to perform in the interrogation phase.

We started by building a conceptual model, specifically an ER model, which was then adjusted to be a logical model of a relational database, which is the structure we decided to be more fitting by observing our structured data.

Going from a conceptual to a logical model, we performed basic query optimization to select optimal primary keys, and to find out whether it was more efficient to introduce attribute redundancy or not, in order to have the most efficient performance on the main queries we expect to use to fulfill our research goals.

For instance, our main queries will consist in computing average measurements by grouping hourly data by date, and then compare the computed daily measurements with another entity containing the historical data, to see which website got closer to the correct measurements.

6.1 ER model

In Figure 6.1 we show our final ER model, which we translated to a SQL database. An Entity-Relationship model (ER model) is a visual representation used to design and describe the structure of a database. It's a high-level model that depicts the relationships between various entities and attributes associated with those entities.

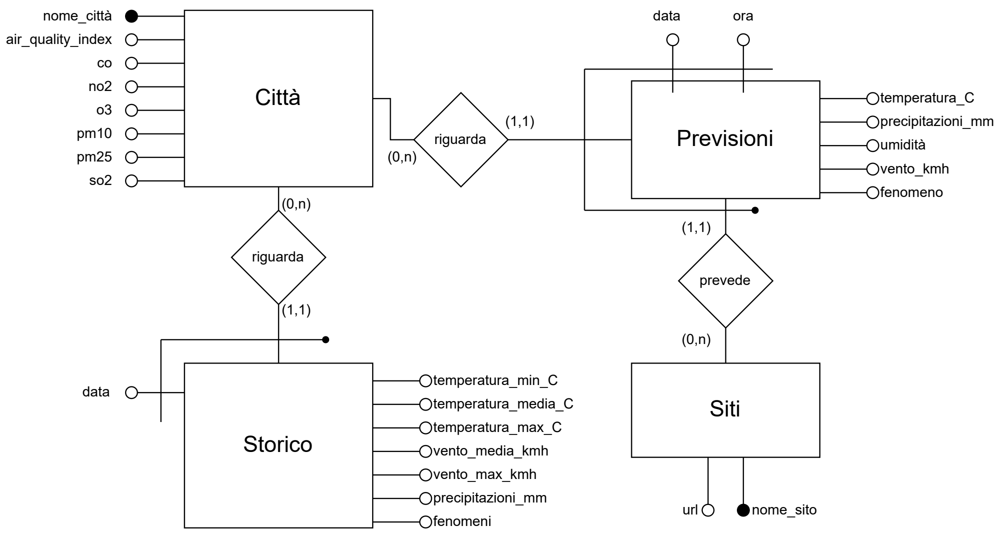


Figure 6.1: ER Model of the database

The Diagram shows entities inside rectangles, with branches indicating attributes: a filled black dot at the extreme of the branch means that attribute is a primary key for the entity. We also represented relationships with cardinalities.

Since we will mainly perform time-focused grouping queries, we decided to refer to cities and website names by using foreign keys to avoid redundancy and optimize performance for join operations. Date and time attributes, instead, were not placed on a separate entity or neither summarized by a serial ID, since it would be less clear and less efficient to perform *GROUP BY date* or *GROUP BY time* queries.

Almost all attributes are numerical (decimal), except for *date*, *nome_città* and *nome_sito*, which are strings, and *fenomeno* and *fenomeni*, which are, respectively, of type *fenom* and *fenom[]* (*fenom* array).

fenom is a custom ENUM data type we created, so that phenomenons could be treated as a categorical variable which can take the values *'nebbia'*, *'neve'*, *'pioggia'*, *'temporale'*.

6.2 Database initialization

In a separate notebook we perform a one-time database initialization, creating the structure translated by the represented ER model and inserting static data, which will remain the same before and after the collection, hence websites data in *Siti* entity and enriched cities data in *Citta* entity.

In our Python notebook, we do this by establishing a connection to a PostgreSQL database using the provided connection parameters. After that, a series of SQL statements are committed as a transaction (through *cursors*, in order to maintain database integrity) to create tables and define their schema in the connected database. We created the tables '*Citta*', '*Siti*', '*Previsioni*', '*Storico*'. After that, the same connection and cursors technique was used to insert data into the tables *Citta* and *Siti*, therefore initializing the database.

After performing DB connection, creation and initialization, a cursor was created to contain all the forecast data. Once all the data was inserted, cursor was committed: this ensured that data was loaded in a single transaction.

6.3 Data insertion

In our original notebook, we now start the insertion of the data into the '*Previsioni*' and the '*Storico*' tables, which are the dynamic data collected in the acquisition period (which was 2 weeks in our case).

We establish a connection to the PostgreSQL database and insert data from the DataFrame which combines the scraped data into the '*Previsioni*' table, and ensure that duplicate entries are ignored. All necessary precautions to ensure and maintain data integrity (through transactions) and error handling are always integrated in every insertion.

We then insert the historical data we downloaded into our database. After the preparation of the data, which made them consistent with the scraped data, we inserted it into the '*Storico*' table in our PostgreSQL database, always keeping in mind potential conflicts and inconsistencies that may occur.

Chapter 7

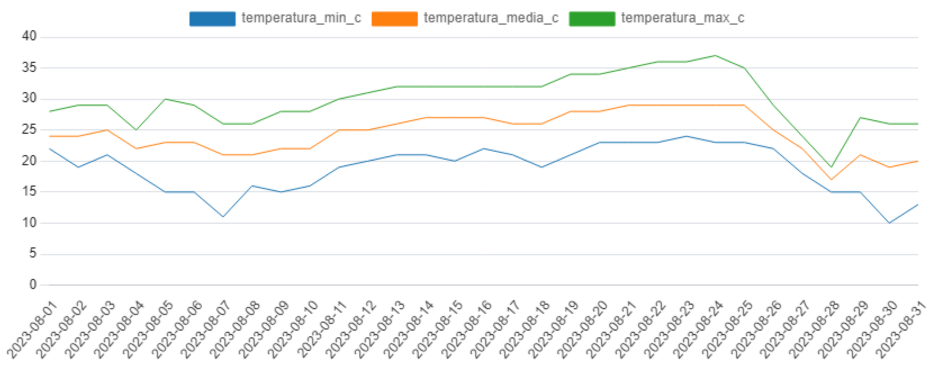
Data Interrogation

Now that we have collected, cleaned, and stored our data, we pass to the final step of our study: data interrogation. Data interrogation refers to the process of systematically querying, analyzing, and investigating a dataset to extract valuable insights and information. In our case, we created SQL queries to interrogate our collected data and showcase some potential insights that could be generated from this procedure.

We performed data interrogation through some data visualization charts to test if our dataset could be used in such a manner, and through SQL queries. We experimented initially with some simple queries, then proceeded with some more advanced analysis, aimed to answer our research question.

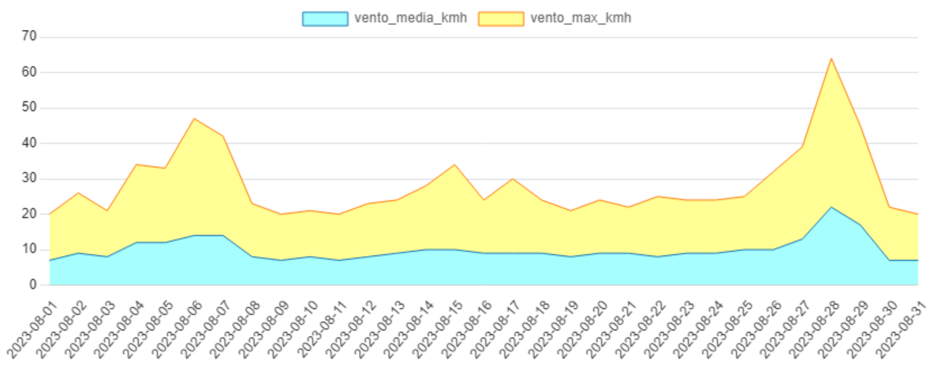
7.1 Data Visualization

1. Temperature in the city of Varese



From this line chart we can observe the course of the recorded minimum, maximum and mean temperatures in the city of Varese for each of the 14 days we collected data for. We can see that on the day 2023-08-07 the minimum temperature was the furthest from the mean, and on the day 2023-08-24 the maximum recorded temperature was furthest from the mean.

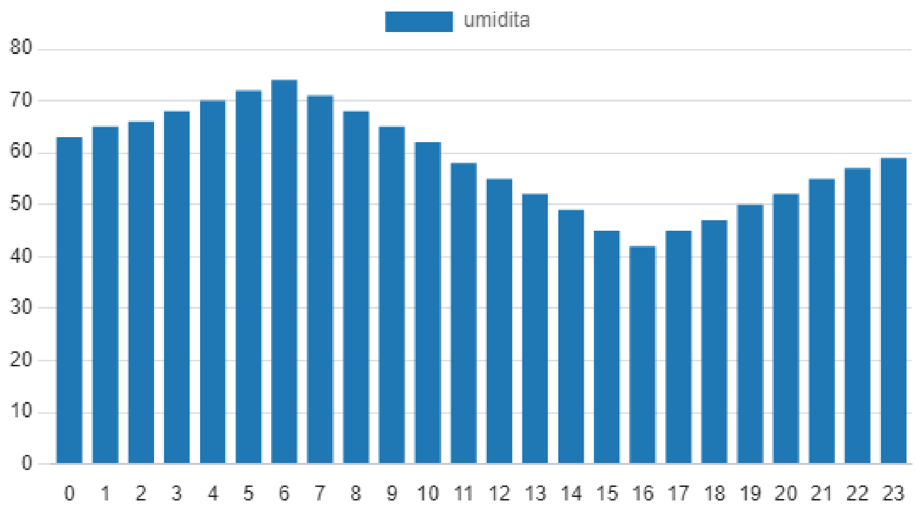
2. Wind speed in the city of Varese



From this area chart we can clearly see that on the day 2023-08-26 the maximum wind was registered in the city of Varese, recording a speed of over 60 km/h. This value was the one that most influenced the mean wind speed over the course of our two week collection period.

3. Predicted humidity per hour

For this bar chart, we selected the predicted hourly humidity percentage for Varese on the 11th of August 2023 from meteo.it.



We observe that the highest percentage of humidity in the selected day is at 6 am, with a percentage of 74%, while the lowest percentage of 42% can be found at 16 pm, which coincides with the hottest hour in the selected day.

7.2 Beginner Queries

1. Temperature range queries

Days with highest temperature range

This query retrieves data from the “storico” table, calculates the average temperature variation for all provinces for each date, and then presents the top 10 dates with the highest temperature variations in descending order.

```
SELECT data, round(avg(temperatura_max_c - temperatura_min_c), 1)
as escursione_c
    FROM storico
    GROUP BY data
    ORDER BY escursione_c DESC
    LIMIT 10
```

data	escursione_c
2023-08-25	12.4
2023-08-22	12.4
2023-08-06	12.2
2023-08-09	12.0
2023-08-18	12.0
2023-08-24	11.9
2023-08-21	11.9
2023-08-23	11.9
2023-08-15	11.8
2023-08-12	11.6

As we can see, the days with highest average temperature ranges are the 25th, 22nd, 6th and so on, with a maximum range of 12.4°C. The 25th was depicted forecasted by the new as an abnormally hot day, after which temperatures would drop sharply.

Note that the 25th, for example, is not included in the forecasts collection period. The highest temperature range within the collection period was still 12.4°C, collected on the 22nd.

Cities with highest temperature range

This query retrieves data from the "storico" table, calculates the average temperature variation for all dates for each province, and then presents the top cities with the highest temperature variations in descending order. Also, we joined the table with data obtained from the data enrichment phase to observe if air quality has some association with temperature ranges.

```
WITH escCitta AS (
    SELECT citta, round(avg(temperatura_max_c - temperatura_min_c), 1)
    as escursione_c
            FROM storico
            GROUP BY citta
)

SELECT citta, escursione_c, air_quality_index, co
    FROM escCitta e JOIN citta c ON e.citta=c.nome_citta
    ORDER BY escursione_c DESC
```

citta	escurs_c	aqi	co
Mantova	11.3	85	100
Milano	11.2	61	102
Monza	11.2	55	199
Como	11.1	53	199
Varese	11.1	53	199
Lecco	9.9	55	199
Bergamo	9.9	59	110
Cremona	9.3	71	95
Brescia	8.5	61	117

We notice that Mantova has the highest temperature range (11.3°C), followed by Milano, Monza and so on. The majority of the provinces with higher temperature range are on the west side of Lombardy. Altitude, proximity to water, and proximity to the Alps might be some factors that influence these values.

There seems to be no particular correlation between temperature range and both Air Quality Index and CO concentration. AQI appears higher when CO concentration is lower, which mostly seems to happen with eastern regions.

2. Days that recorded critical temperatures

This query is fetching information about days when the maximum temperature in a city reached 35° Celsius or higher.

```
SELECT citta, data, temperatura_max_c
        FROM storico
        WHERE temperatura_max_c >= 35
        ORDER BY temperatura_max_c DESC
        LIMIT 10
```

citta	data	t_max_c
Mantova	2023-08-25	38
Monza	2023-08-22	37
Varese	2023-08-24	37
Como	2023-08-24	37
Mantova	2023-08-24	37
Mantova	2023-08-23	37
Milano	2023-08-22	37
Milano	2023-08-23	37
Milano	2023-08-24	37
Monza	2023-08-23	37

We can see that the critical benchmark we set (35°C) was reached multiple times (here we only represent the top 10 of 47). The highest peak was 38°C and was reached by Mantova on the 25th, which also was the highest temperature range day as queried above.

3. Rainy days per city

This query retrieves data from the "storico" table and calculates the count of days with specific weather phenomena ("pioggia" and "temporale") for each city. It then presents the results, ordered by the cities with the highest count of days with precipitation.

```sql
SELECT citta, count(data) as ggPrec
FROM storico
WHERE Fenomeni && ARRAY['pioggia', 'temporale']::fenom[]
GROUP BY citta
ORDER BY ggPrec DESC
```

citta	ggprec
Como	10
Varese	10
Bergamo	9
Lecco	9
Mantova	7
Milano	7
Monza	7
Brescia	5
Cremona	5

We can see that the rainiest cities have been Como and Varese, both with 10 days of rainfalls, while the least rainy have been Cremona and Brescia with only 5.

4. Forecasted phenomena per website

This is the first more complex query, which involves multiple layers of sub-queries and aggregations. It calculates the sum of occurrences of various weather phenomena for each website and presents the results in a structured manner.

```sql
WITH hourfen AS (
        SELECT sito, citta, data, fenomeno, count(fenomeno)
                FROM previsioni
                GROUP BY sito, citta, data, fenomeno
), dayfen AS(
        SELECT sito, citta, data, CASE WHEN
        SUM(CASE WHEN "count" > 0 THEN 1 ELSE 0 END) > 0 THEN 1
        ELSE 0 END AS nPrec
                FROM hourfen
                GROUP BY sito, citta, data
)

SELECT sito, sum(nPrec) as nFenomeni
FROM dayfen
GROUP BY sito
ORDER BY nfenomeni DESC
```

sito	nFenomeni
3bMeteo	21
Meteo.it	19
IlMeteo	13

We can see that iLMeteo predicted 13 phenomena, 3B Meteo predicted 21, and finally, meteo.it predicted 19.

7.3 Research Question

Our main goal of our study was to compare the performance of the three websites we chose to work on by collecting daily weather predictions, and in the end compare the predictions with historical data. We chose three categories to check whether one website performs better than another for specific information. We decided to evaluate best temperature predictions, best wind predictions and best weather phenomena predictions.

7.3.1 Best Temperature Predictions

As introduced earlier, the first category we decided to focus on was to find which website performed better at predicting the temperature:

- we first grouped hourly forecasts to make daily estimates, by computing the *minimum*, *average* and *maximum* temperature for each day
- then joined historical daily measurements with computed daily forecasts
- computed temperature differences between historical and forecasts (*"errors"*);
- grouped by website by summing all errors to obtain a *"total error"*
- presented and ordered results.

```
SELECT sito,
       SUM(temperrordate) AS tempError
FROM   (SELECT sito,
               data,
               SUM(difftemp) AS tempErrorDate
        FROM   (SELECT sito,
                       citta,
                       data,
                       ( mintempdiff + maxtempdiff + mintempdiff )
                       AS diffTemp
                FROM   (SELECT sito,
                               s.citta,
                               s.data,
                               Abs(f.mintemp - s.temperatura_min_c)
                               AS minTempDiff,
                               Abs(f.maxtemp - s.temperatura_max_c)
                               AS maxTempDiff,
                               Abs(f.avgtemp - s.temperatura_media_c)
                               AS avgTempDiff
                        FROM   (SELECT sito,
                                       citta,
                                       data,
                                       Min(temperatura_c) AS minTemp,
                                       Round(Avg(temperatura_c), 1)
                                       AS avgTemp,
                                       Max(temperatura_c) AS maxTemp
                                FROM   previsioni
                                GROUP  BY sito,
                                          citta,
                                          data) AS f
```

```
                                     join storico AS s
                                       ON f.citta = s.citta
                                          AND f.data = s.data) AS c) AS i
        GROUP  BY sito,
                  data) AS o
GROUP  BY o.sito
ORDER BY temperror ASC
```

sito	temperror
Meteo.it	543.0
IlMeteo	559.8
3bMeteo	560.9

We can see that the best website for predicting the temperature is meteo.it, obtaining an error of 543. The website that performed worst in this category was 3B Meteo. As these observations are only the results of our experiment and study, this is not enough to make assumptions on the real accuracy of the websites.

7.3.2 Best Wind Predictions

For the second category we focused on the wind predictions. The steps were similar to the previous query, as we retrieved the wind speed differences; calculated daily and total wind speed errors; and presented and ordered the results. Overall, this query processes forecasted and historical wind speed data, calculates the differences between them, and computes the cumulative wind speed error for each site. It then presents the sites along with their corresponding total wind speed errors, ordered from lowest to highest wind speed error.

```
SELECT sito,
       SUM(winderrordate) AS windError
FROM   (SELECT sito,
               data,
               SUM(diffwind) AS windErrorDate
        FROM   (SELECT sito,
                       citta,
                       data,
                       ( avgwinddiff + maxwinddiff ) AS diffWind
                FROM   (SELECT sito,
                               s.citta,
                               s.data,
                               Abs(f.avgwind - s.vento_media_kmh)
```

```
                                AS avgWindDiff,
                                Abs(f.maxwind - s.vento_max_kmh)
                                AS maxWindDiff
                         FROM   (SELECT sito,
                                        citta,
                                        data,
                                        Round(Avg(vento_kmh), 1)
                                        AS avgWind,
                                        Max(vento_kmh) AS maxWind
                                 FROM   previsioni
                                 GROUP  BY sito,
                                           citta,
                                           data) AS f
                                join storico AS s
                                  ON f.citta = s.citta
                                     AND f.data = s.data) AS c) AS i
        GROUP  BY sito,
                  data) AS o
GROUP  BY o.sito
ORDER BY winderror ASC
```

sito	winderror
Meteo.it	828.8
3bMeteo	963.7
IlMeteo	1034.2

From the output we can see that the best website for predicting wind speeds is meteo.it, which gained an error of 829. The least performing website in this category was iLMeteo.

7.3.3 Best Weather Phenomena Predictions

For the third category we observed how well each website predicted weather phenomena. This query involves comparing the forecasted weather phenomena with historical weather phenomena. To evaluate the similarity between the predictions and the historical records, we created a scoring system. We first aggregated the hourly forecast data into daily forecast data, concatenating all the distinct daily phenomena into an array; combined the predicted and historical data; calculated our similarity score; and finally grouped by website, presented and ordered the results.

The similarity score was structured as follows:

- If only one between the predicted and historical phenomena arrays is empty or null, that day is assigned a score of 0

- If both arrays are not empty, we assign a score equal to the number of matching elements between the two arrays divided by the size of the biggest array between the predicted and historical one. In this way, we obtain a percentage of match between the arrays.
- If both the arrays are empty, the prediction is 100% correct, but we assign a score of 0.1 (instead of 1) to give more weight to predictions when weather phenomena occur, which in August are way less frequent than sunny days with no phenomena.

In the end, we sum the scores for every website and identify the one with the highest score as the most accurate one in predicting weather phenomena.

Overall, this query is evaluating how well forecasted weather phenomena match the historical data for each site, city, and day. Our scoring system indicates the degree of similarity, considering scenarios of missing or empty phenomena arrays.

```sql
WITH DailyForecast AS (
    SELECT
        sito,
        citta,
        data,
        ARRAY_REMOVE(ARRAY_AGG(DISTINCT fenomeno), NULL)
        AS fenomeni_previsti
    FROM
        Previsioni
    GROUP BY
        sito,
        citta,
        data
),
CombinedData AS (
    SELECT
        s.citta,
        s.data,
        s.fenomeni AS fenomeni_storici,
        p.fenomeni_previsti,
                p.sito
    FROM
        Storico s
    JOIN
        DailyForecast p ON s.citta = p.citta AND s.data = p.data
),
FinalData AS (
        SELECT
                c.sito,
                c.citta,
                c.data,
                c.fenomeni_storici,
                c.fenomeni_previsti,
                CASE
                        WHEN c.fenomeni_previsti IS NULL
                                OR c.fenomeni_storici IS NULL THEN 0
                        WHEN c.fenomeni_previsti = '{}'
                                AND c.fenomeni_storici = '{}' THEN 0.1
                        WHEN c.fenomeni_previsti = '{}'
                                OR c.fenomeni_storici = '{}' THEN 0
                        ELSE( CAST( COALESCE(
                                (SELECT COUNT(*)
                                 FROM UNNEST(c.fenomeni_previsti) f
                                 WHERE f = ANY(c.fenomeni_storici)), 0
                                        ) AS numeric )
                                 /
                                GREATEST(
                                        CAST(array_length(c.fenomeni_previsti,
                                                        1) AS numeric),
                                        CAST(array_length(c.fenomeni_storici,
                                                        1) AS numeric)
                                )
                        )
                END AS Punti
        FROM
                CombinedData c
        ORDER BY sito, citta, data
)

SELECT sito, ROUND(sum(punti), 1) as score
FROM FinalData
GROUP BY sito
ORDER BY score DESC
```

sito	score
3bMeteo	15.8
Meteo.it	15.4
IlMeteo	13.2

We can see from the output that the best website for predicting weather phenomena is 3B Meteo, while the website that performs worst is iLMeteo. Differently from the previous research questions, scores here are very close between each other.

Chapter 8

Conclusions

At the end of our project, after having gone through the phases of data collection, data preparation, data augmentation, data quality inspection, database creation and finally data interrogation, we can finally draw some conclusions.

Our aim was to collect weather predictions for two weeks in the month of August 2023. Our sources were three popular Italian websites, and we enriched the collected data with air quality measures. We focused on the twelve provinces of the Lombardy region in Northern Italy.

Our data acquisition methods were three:

- **Data Download:** Data download refers to the process of transferring digital information or files from one location, often a remote server on the internet, to your own computer or device. We obtained our historical data through this method.

- **Web Scraping:** Web scraping is a technique used to extract specific information from websites. It involves writing a computer program or script that visits web pages, simulates human browsing behavior, and then gathers the desired data from the page's HTML code. The weather predictions from the three websites we selected were collected using web scraping techniques.

- **API:** An API, or Application Programming Interface, is like a bridge that allows different software applications to communicate and interact with each other. It defines a set of rules, protocols, and tools that developers can use to build connections between their own programs and external services, libraries, or systems. The data regarding air quality was obtained through an API.

We recognise one of the main limitations of web scraping is that websites can change their structure and layout frequently. If a website's structure changes, the web scraping code may break, requiring constant maintenance and updates to adapt to these changes. Also, if we were to expand our range of cities too much, we could encounter performance impact issues, and different cities may offer different information to scrape.

Our research question consisted in comparing the performance of the three websites we selected. In other words, we wanted to see which website provided the best weather forecast predictions. We did so by comparing the data collected from the websites' predictions to the historical data. However, we decided to perform tests in three different categories: we found which website performed best in predicting the temperature; which website performed best in predicting wind speeds; which website performed best in predicting weather phenomena.

Through our study we found that meteo.it performed best in two categories: temperature and wind. Meteo.it recorded an error of 543.0 in the temperature category (against errors of 559.8 for iLMeteo, and 560.9 for 3B Meteo) and an error of 828.8 for the wind category (against errors of 963.7 and 1034.2 for 3B Meteo and iLMeteo, respectively). The best website for predicting phenomena turned out being 3B Meteo, recording a score of 15.8 (against the scores of 15.4 and 13.2 recorded by meteo.it and iLMeteo). We observed, also, that iLMeteo performed worst in two of the three categories (wind and phenomena).

Disclaimer: The accuracy metrics presented in this paper were created for querying purposes only. Also, the collection period was too restricted to be statistically significant. Therefore, the results should not be interpreted as assumptions about the real accuracies of the weather forecasts.

8.1 Future Developments

Future developments of our initial project could include additional functions to ease and expand the scope of our work. Among other things, some improvements could include:

- **Automatic Data Collection:** during our two week collection period, we manually executed our scripts daily at a specific time (16 pm). An automatization feature could ensure a daily collection without the need of a physical user present to execute the script. Alternatively, a semi-automatic set up could be a Telegram bot which through a command executes the scripts and returns the collected and cleaned data as and when a user desires.

- **Nation Wide Data Collection:** our work focused on the provinces of the Lombardy region in Italy. A nation wide collection phase could be possible by gathering the urls of any city a user desires. However, significant adjustments to the web scraping scripts will have to be performed, as not all cities for some websites offer the same information (for example, coastline cities offer information regarding sea level and other metrics that in a land locked city are irrelevant).

- **Long Term Data Collection:** to perform a more in depth study of the accuracy of the three websites we have considered, a long term study should be performed. Our study was limited to two weeks in August, but a long term study (potentially, even a year long study) will offer more insights regarding the accuracy of the three websites we considered, especially by covering all four seasons.

- **Additional Websites for Data Collection:** in our project we focused on three weather forecast websites. The study could be extended to additional websites, however custom web scraping scripts will have to be written for each website. Also, according to the information gathered from the new websites, some changes might have to be made at the feature selection or the queries.