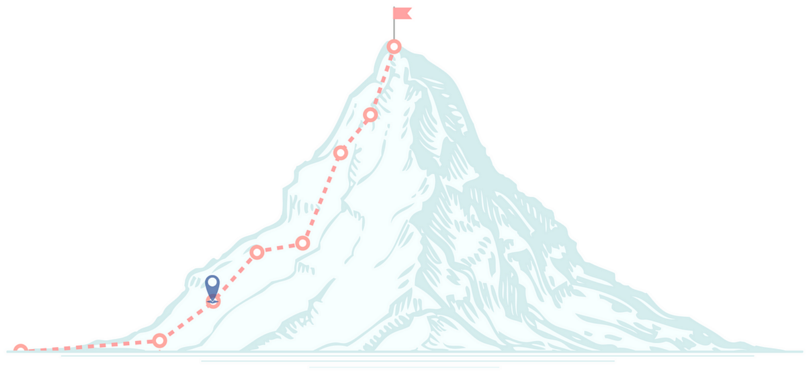
Universitat Politècnica de Catalunya

Bachelor's Degree in Data Science and Engineering

Visual Analysis of Datasets of Mountain Tracks

Progress Report



Student:

Xavier Rosinach Capell

Advisor:

Pere-Pau Vázquez Alcocer

Co-Advisor:

Oscar Argudo Medrano

1 Introduction

The following document aims to provide a structured overview of the project's progress, which is focused on developing an interactive application based on hiking routes shared by users on *Wikiloc*. By analyzing this data set, the objective is to offer a useful tool to hikers to explore, analyze, and better understand mountain trails in Catalonia.

This report outlines the progress achieved so far, details the key milestones set, and provides information on the current status of each one. It also identifies some potential deviations from the planned schedule.

1.1 Project Goals

Before continuing with the document, it is essential to recall the objectives of the project to ensure that the tasks carried out so far align with the objectives of the project.

The main objective of the project is to create an interactive application where users can view, through visualizations, information about mountain trails in three of the most popular hiking areas in Catalonia: *Canigó*, *Matagalls*, and *Vall Ferrera*. These visualizations aim to help users understand aspects such as the seasonality of the tracks, the most frequented routes, and the most famous points of interest.

This objective arises from the lack of a tool that brings together all these routes in a unified way. The web application from which the data has been sourced (*Wikiloc*) only displays routes individually, with their corresponding information. In contrast, the final application will integrate all route data so that users can gain more specific insights into all the trails in each area.

Since the work carried out so far has focused on data processing, the next section will mention that the data have already been processed and are now ready to be used to create the intended visualizations.

2 Work Plan

2.1 Tasks and Milestones

As mentioned in the initial document, the project consists of two main tasks. The first involves processing the initial data, where the original tracks are transformed to obtain data that are sufficiently useful for creating future visualizations. The second part focuses on the design and creation of these visualizations.

Regarding the first part, it is now complete (read Data Processing). For each area, all tracks have been processed efficiently and without errors. The current data are considered accurate and ready to be used for visualizations. However, it is possible that in a future this process may be a little modified due to changes in the visualizations.

The student is currently in the process of creating visualizations (read Data Visualization). Some of the information to be displayed have been planned and designed and the supervisors have given some comments about it. There has been a slight delay in the start of this second phase. This is due to unexpected errors that occurred during the data processing stage.

2.2 Meeting and Communication Plan

The communication plan between the student and the project supervisors has remained the same since the beginning. The student shares information and progress through a Discord chat, where the supervisors respond to all the student's questions and suggest possible improvements.

In addition, face-to-face meetings have been held so that the student can discuss the progress made and create a discussion between them and the supervisors. This combination of online chat and in-person communication ensures effective project development, promoting both collaboration and independent problem solving.

3 Data Processing

The following section will provide a reminder of the input data structure and explain how it has been processed to generate the data set that will later be used to create visualizations.

As a reminder, the project's data consists of hiking route trackings extracted from the *Wikiloc* application. These are organized into three folders, each corresponding to a highly frequented hiking area in Catalonia: *Canigó*, with 9,890 initial track files; *Matagalls*, with 9,932 initial track files; and *Vall Ferrera*, with 14,401 initial track files.

Each route is stored in *JSON* format and contains two main parts. The first contains route information, such as the user, URL, date, difficulty level, etc. The second is the route coordinates, which include longitude, latitude, elevation, and timestamp. In addition, the data may include points of interest that users have added to the route.

The data processing for this project was based on transforming the input *JSON* files. Check this table to see the rows contained in these files. For each of these files, relevant information about the track was saved, all points of interest were stored along with their corresponding information, and a map matching process was carried out in order to identify the paths recorded in the *Open Street Map* network that the user had followed. The entire process is documented in the appendix; however, in this section, we will explain the most important points to take into account.

All tracks in the same area are initially stored in a compressed *ZIP* folder. Therefore, we initially have *canigo.zip*, *matagalls.zip*, and *vallferrera.zip*. It is important to note that the process described in the following sections will be carried out separately for each area.

3.1 Data Preprocessing

As already mentioned, the tracks are initially found in a compressed folder. We will need to start our code by unzipping these folders. In addition, we will create other folders to store the output generated by our code.

Thus, the first part of the code (data preprocessing) focuses on organizing the data directory and ensuring that the necessary directories and data frames are in place to begin executing the map matching process. These folders will be created if they do not already exist, and the data frames will either be created or read from existing files. Check this schema to see how the data directories are structured.

In addition, an example directory has been added to GitHub containing a set of 10 tracks from the *Canigó* area that we will use to show the tables as examples.

For further information on these processes, read Data Preprocessing. You will be able to check how each directory and data frame has been created and why it is used.

Once the structure of the data directories has been defined, we can begin the map matching process. In this step, we take all the *JSON* files from the `/Data/Input-Data` folder and output the transformed version of them them into the `/Data/Output-Data/FMM-Output` directory.

3.2 Fast Map Matching

The next part of the processing is based on performing a map matching. That is, from the coordinates we have for our tracks, we align them to the paths corresponding to the *Open Street Map* network.

The library used to carry out this process is called *Fast Map Matching* (`fmm`). It was installed following the tutorial provided on the same official link. It is run using *Cygwin*, as it is not compatible with the *Windows* environment.

This library requires as input the track coordinates in *Well Known Text*[1] format (assuming they are ordered), and the predefined model of the area, which we will have created using the *Open Street Map* path network. This model needs to be configured. To do so, three parameters are considered:

- `k`: corresponds to the number of candidates the library is allowed to search for each point. The higher the value, the longer the process takes, but the more accurate the matching becomes. We tested values 2, 3, and 4, and kept the first match returned by the code. For almost all tracks, a value of 2 was sufficient.
- `radius`: the search radius for paths (in degrees). Since the tracks normally follow defined paths, we use the value 0.001 (approximately 100 meters).
- `gps_error`: the GPS error margin allowed for the track (in degrees). As before, we use a constant value of 0.001 (approximately 10 meters).

Therefore, for each track, we applied the model up to a maximum of three times (depending on whether a matching track was found with lower values of `k`).

The output returned by the library corresponds to an object that contains the coordinate points of the *Open Street Map* paths that were matched. It also provides additional information such as the *Open Street Map* edges identifiers from which each coordinate point comes, speed, error, among others. We only retain the information about the edges and the matched coordinates.

We save this information in a *CSV* file in the folder `/Data/Output-Data/FMM-Output`, using the track identifier as the file name. Additionally, we store the configuration used (track identifier and values of `k`, `radius`, and *gps_error*) in the data frame `output_files.csv`; in case an error occurred during the process for a specific file, it is noted in `discard_files.csv`.

For more information of this process, the code explanation can be found in the FMM Processing section. There, it is detailed how, for each area, all the files have been transformed.

[1]Well Known Text (WKT) is a text markup language for representing vector geometry objects, such as points, lines, and polygons.

3.3 Postprocessing

The third and final part of the processing is based on transforming both the initial data and the output obtained from the map matching to create a database with useful information for the visualizations we will create in the future.

As we already know, we have on one hand the *JSON* files with information about each track (user, title, difficulty, ...). In these same files, there are also the (non-transformed) coordinates and the points of interest of the route. We also have a data frame for each track with the transformed coordinates along with the edges. Additionally, we have stored information about the network (files in *SHAPEFILE* format for the edges or nodes).

With these files, in this part we obtain, on one hand, a file for each track in *CSV* format saved in the folder `/Data/Output-Data/Cleaned-Output` with the track identifier as the name. This data frame contains an identifier for each coordinate point, an identifier for the edge, coordinates, elevation, among others. This can be seen in the following table.

We will also obtain the data frames `cleaned_out.csv` with metadata about each track, `edges.csv` with the edges the tracks have passed through and each track that passed through them, and `waypoints.csv` with all the information about the points of interest in the area.

Now, with these three data frames for each area, we can begin to think about the design of the visualizations we want to display to the user. For more information about the postprocessing part, read the following section.

4 Data Visualization

The data visualization component is the most important part of the project. This is where clear and concise graphics must be designed to allow users to obtain meaningful information about the trails in the mountainous regions of Catalonia.

We will define two types of visualizations. On one hand, those that use geographic data—maps will be colored based on different criteria. On the other hand, there will be visualizations that do not display geographic data, but instead use other types of information to reveal patterns in the routes of each area. We will also consider a distinction between visualizations that represent all tracks in a given area and those that focus on individual tracks. The main goal is to provide the user with a generalized overview of all routes; however, users may also want to focus on a specific trail.

So far, no visualization has been finalized for inclusion in the final product. Nevertheless, this section will present some of the proposals the student has developed to potentially include in the final output, and some comments of the supervisors. These visualizations have been created using the `Altair` and `Folium` libraries, although the final ones may end up being developed with a different library.

4.1 Non-Spatial Visualizations

4.1.1 Individual Tracks

In order to create visualizations in this part, we will need the data of a single track stored in the directory `/Data/Output-Data/Cleaned-Tracks`, with this format. We will use the columns that record the elevation, distance, kilometer, time, elevation difference, and pace. Two visualizations have been created.

The first visualization corresponds to a line chart that shows the elevation profile. On the X axis we find the distance of the track, and on the Y axis the elevation of the coordinates. In this way we can see the ascents and descents of the route taken by the user. In addition, we will also add the possibility of showing the highest and lowest elevation points, so that the user can get an idea of how high they will climb. We will also add a vertical line with which the user can travel the track in a way that will show them in a banner the distance traveled so far, the elevation, and the time, among others.

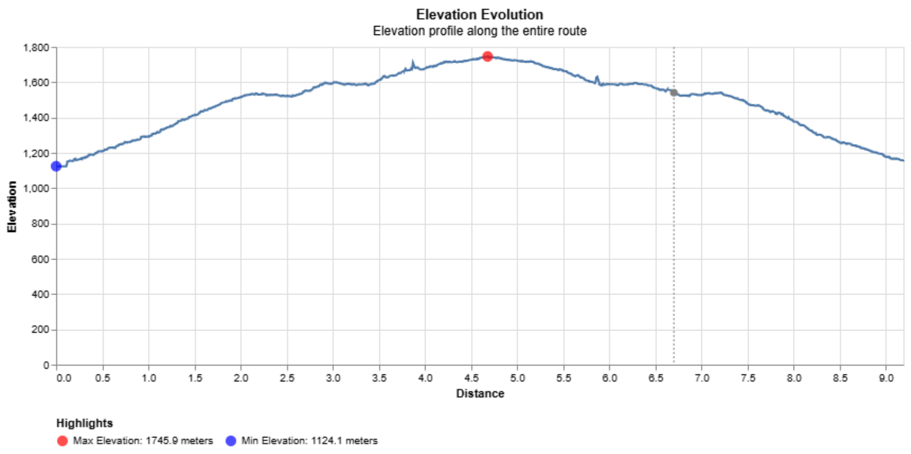


Figure 1: Elevation profile of the track .

A question that has been asked about this visualization is whether it could be implemented for all tracks at once. In order to be able to see if users start their routes going uphill, downhill, or flat. In the future, we will investigate whether this can be done.

The second visualization corresponds to a bar chart that contains the average pace for each kilometer of the track. On the X axis we have the kilometers, and on the Y axis, a bar rises according to the average pace for that section. We will also add the track's elevation profile to this graph; this way we can see if the user goes slower on climbs, and if he goes faster on descents.

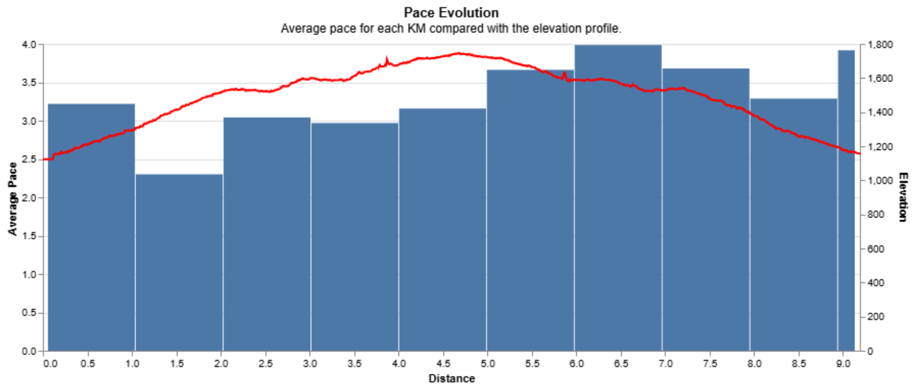


Figure 2: Average pace per KM of the track .

These two graphs have been effectively validated. However, we must bear in mind that they do not answer any efficient questions. They only serve to allow the user to learn more about a specific route.

4.1.2 All Tracks

This second part of visualizations of non-spatial data corresponds on creating tools to allow the final user see information about all the set of tracks of a given zone. We will use the data frame `cleaned_out.csv`. We will use information such as the distance, elevation gain, seasonality (date, month, weekday, ...), and others.

Some of the first visualizations designed are those that show us temporal trends. For each time division we have (year, month, day of the week, and season), a bar chart has been created to show the total number of tracks recorded during that period.

Regarding the years, we can clearly see a growing trend over time, due to the increasing popularity of the application. We can also observe a decline during the year of the COVID-19 pandemic. For the months and seasons, we can clearly see that in summer and autumn people tend to go to the mountains more often. The same happens with the division by days of the week, where, as expected, more routes have been uploaded during the weekends.

We can see the four visualizations together below:

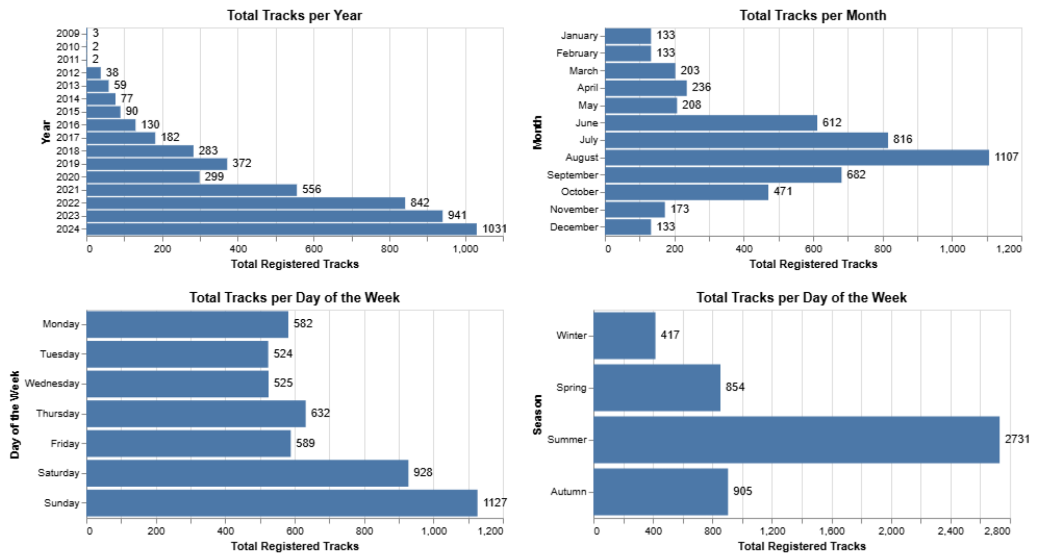


Figure 3: Time divisions bar charts.

We are also interested in seeing the distribution of routes based on quantitative data. Therefore, we will use histograms to display the total number of tracks according to distance, elevation gain, elevation loss, total time, and average pace. These are the designs:

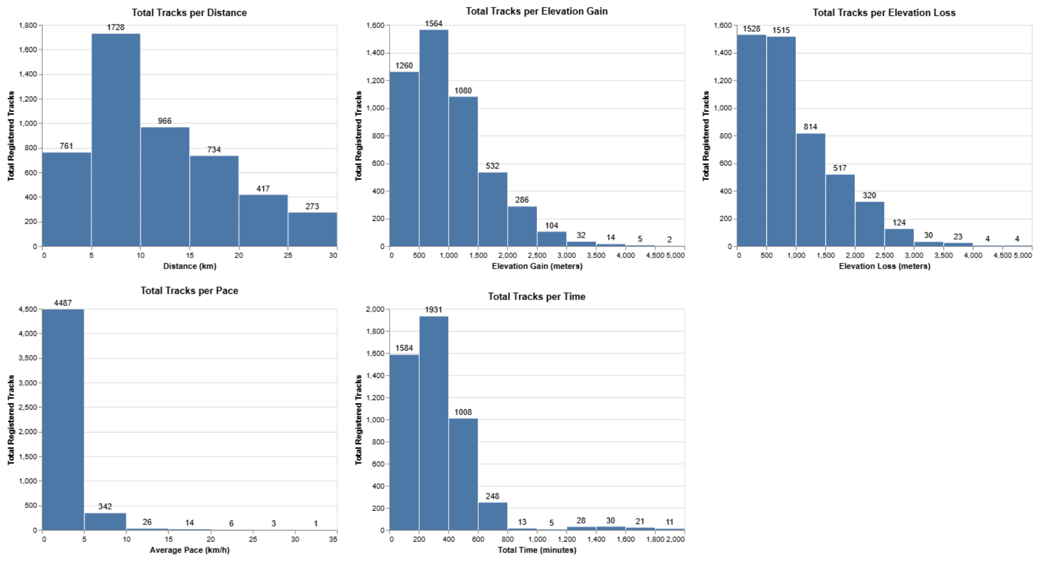


Figure 4: Quantitative data histograms.

With these visualizations, the user will be able to see the trends of the area, and we will be able to draw conclusions based on the distance of the most frequent tracks.

To conclude this section on visualizations, a heat map has also been designed to compare two quantitative variables from the five previously mentioned (distance, elevation gain, elevation loss, total time, and average pace). The cells of the heat map will be colored according to the total number of tracks recorded with that information. This visualization will be interactive; the user will be able to choose the two axes based on the comparison they wish to make. In this case, we will display the comparison between distance and elevation gain:

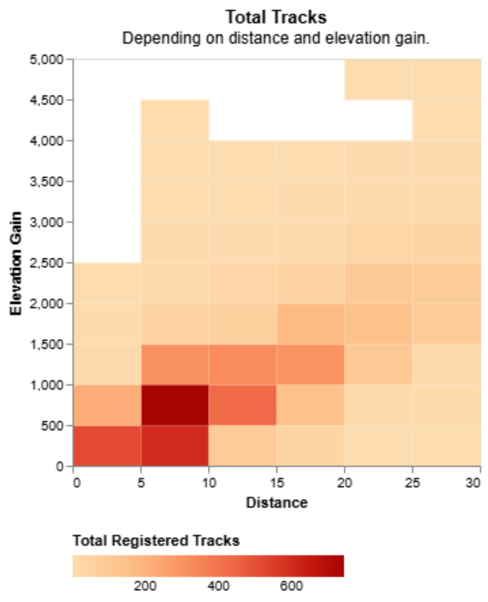


Figure 5: Total tracks heatmap (distance and elevation gain).

4.2 Spatial Visualizations

4.2.1 Individual Tracks

The following visualizations use geographical data, along with other types of data, to draw maps so that the user can learn about different routes. We will start by showing visualization proposals for an individual track. These will be displayed in case the user wants information exclusively about a single track. They will be shown alongside those created previously.

The main idea is to display the path the user has followed throughout the track using a line. To provide more information, we will color this line based on certain criteria: path popularity, speed, elevation, etc. In the following example, we can see the path colored according to the popularity of the trails. We can observe that at the beginning of the track, more people have passed through that path. This could be because there is a starting point for the route nearby, such as a parking area.

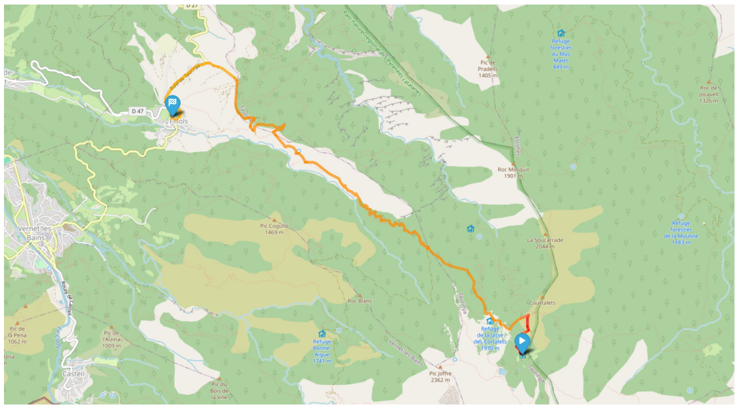


Figure 6: Path of the track 301306.

4.2.2 All Tracks

When showing geographic information about the routes, we can do it in two ways. On one hand, we can display a graph similar to the previous one, but showing more than one track. However, this type of graph will only display a filtered version of the routes. In the following graph, the routes that start within a 100-meter radius of the start of track 301306 (shown in the previous graph) have been filtered. This time, however, the routes have been colored according to their total distance.

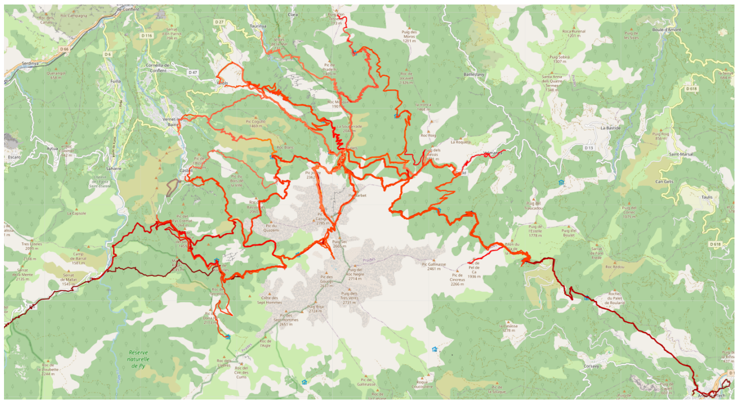


Figure 7: Filtered zone tracks.

This graph shows very useful information for the user: given a starting point, which are the most trafficked tracks in the area (in the example, they are colored by distance, but it could also be done by popularity). However, showing all the tracks may confuse the user. That is why the following visualization has also been designed.

Let's remember that we had the data frame `/Data/Output-Data/Dataframes/output_files.csv`, which contained geographical information about the network axes, along with the total number of tracks that passed through each axis. We will use this information to draw those axes where tracks have passed, and they will be colored according to the total number of tracks that have passed through them:

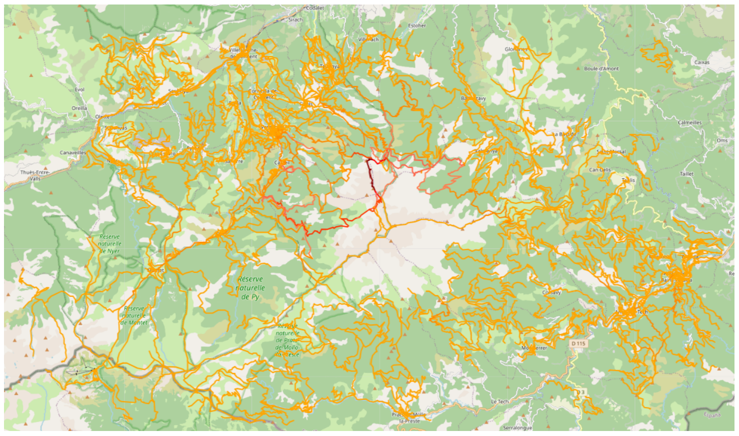


Figure 8: Most popular paths from *El Canigó*.

We can observe, with this graph, which are the most popular and trafficked paths. We could also add other functionalities to this graph, such as filtering (by route start or end, distances, elevation, etc.). Additionally, we could add an interaction where, when the user clicks on any axis, it shows information about the routes that have passed through that axis.

Next, we display the same graph as before, but enlarged so that we can observe that the paths closer to the summit of *El Canigó* have many more routes:

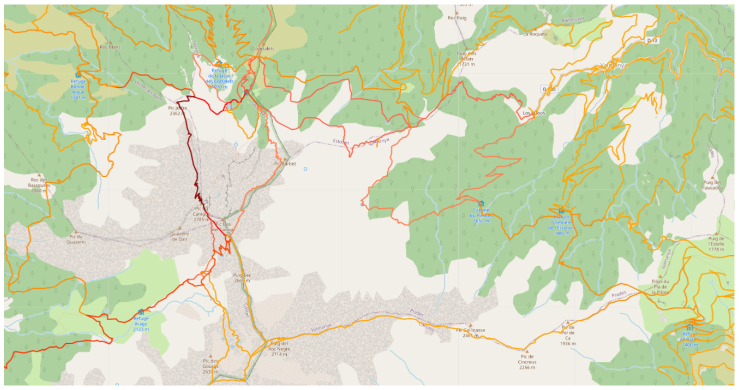


Figure 9: Most popular paths from *El Canigó* (zoomed).

5 Appendix

5.1 Data Preprocessing

This is the first part of the processing of the tracks for each area. In this part, we focus solely on setting up the directory and dataframe structure so that all the information can be stored there later on. In the explanation, we will mention the different functions of the code, along with the names of directories and dataframes (files in *CSV* format). The entire code can be found at the following link.

1. We begin by calling the `extract_zip()` function, which reads the initial *ZIP* file of the area and transfers all its contents to a temporary folder. Afterwards, all the *JSON* files from this folder — that is, those corresponding to routes — are moved to a folder named `Data/Input-Data`. This folder is only created if the folder with the extracted *JSON* files does not already exist.

2. With the `create_osm_network()` function, we create the *Open Street Map* path network. To do this, a polygon and a graph are created using the `osmnx` library (based on coordinates that define each area). For each area, a file in *SHAPEFILE* format will be created for both the nodes and the path edges. All this data is saved in the `/Data/OSM-Data` folder. The function only runs if this folder has not been created yet.

3. The next function we call is `output_structure()`, which will create the entire structure of folders and dataframes needed for the output data. Among these are:

   - The `/Data/Output-Data` folder, which contains all the other folders.
   - The `/Data/Output-Data/Dataframes` folder, where all the dataframes mentioned below will be stored. It is created if it does not already exist.
   - The `/Data/Output-Data/FMM-Output` folder, which will hold all the output track data obtained using the algorithm from the second part of the processing. It is created if it does not already exist.
   - The `/Data/Output-Data/Cleaned-Tracks` folder, which will contain all the output track files after applying the algorithm from the third part of the processing. It is created if it does not already exist.
   - The `/Data/Output-Data/Logs` folder, containing files that log each execution.
   - The dataframe `/Data/Output-Data/Dataframes/discard_files.csv`, which keeps a record of the identifiers of discarded tracks and the type of error that led to their dismissal. This is either created empty or read if it already exists.
   - The dataframe `/Data/Output-Data/Dataframes/output_files.csv`, which contains the information used for `FMM` processing (track identifier and parameters for the algorithm — `k`, `radius`, and `gps_error`). This is either created empty or read if it already exists.

4. Next, we use the `create_edges_df()` function, which takes as input the data extracted from *Open Street Map* network and the path to the `/Data/Output-Data/Dataframes` folder in order to create the `edges.csv` data frame as follows:

   - We read the *SHAPEFILE* containing the edges using the `geopandas` library. From this file, we use the columns `u`, `v`, and `geometry` (start and end edge identifiers, and geometry).
   - Since some edges are bidirectional — that is, they have the same geometry and identifiers but in reverse — we reorder these identifiers from smaller (`u`) to larger (`v`), so we keep only one row for each duplicated edge.
   - We add a column named `id` containing a unique numeric identifier for each edge, a column `total_tracks` (initialized to 0), which will later be updated to reflect the number of routes passing through that edge, `list_tracks` (initialized as an empty list), which will be filled with the tracks passing through the edge, and `weight`, which will be the normalized version (from 1 to 10) of the `total_tracks` column.

5. With the `create_cleaned_output_df()` function, we create the `cleaned_out.csv` dataframe (inside `/Data/Output-Data/Dataframes`), which will be filled during the third part of the processing with relevant route information (user, date, weather, etc.).

6. Finally, we call the `create_waypoints_df()` function, which creates the `waypoints.csv` dataframe (inside `/Data/Output-Data/Dataframes`), containing information about the points of interest that the routes pass through.

With these steps, we will have created all the folders and some input data needed. With all this information, we can start running the Map Matching algorithm defined in the second part of the code.

5.2 FMM Processing

The second part of the data processing is based on applying the map matching algorithm to find the *Open Street Map* paths that each processed route has followed. The Fast Map Matching library is used, which, for each input coordinate, returns an output point corresponding to a location recorded on the path network (created with `osmnx`). The algorithm uses three input parameters:

- The number of candidates `k`: this corresponds to the number of initial matching point options the library finds. In the end, it selects the one with the least error.
- The search radius `radius`: given an input point, it looks within this radius for all specified candidates (in degrees).
- The GPS error `gps_error`: corresponds to the margin of error allowed for the GPS (in degrees).

Next, we will explain the code used in this second part of the processing, along with the `fmm` input parameters used. It is important to note that at the beginning of this process, the function calling the preprocessing part has already been executed, so all the data frames and directory paths will be defined. The code can be found at the following link.

1. Once the main function from the preprocessing part has been called, we create the network and the graph of paths that we will need as input for `fmm`. We also create (or read) the `UBODT.txt` file (which is also necessary). Finally, with the network, the graph, and the file, we can create the required model.
2. We obtain a list of tracks that have already been processed (either found in the output data frame or in the discarded files data frame). With this list, we check, before processing a file, whether it has already been processed. From there, we enter a processing loop for each file found in the input folder.
3. We use the `extract_information()` function to obtain, using the path of the route's *JSON* file, the data frames of coordinates and waypoints, as well as the activity type (used later).
4. If the activity type is *"Senderisme"* (hiking), we proceed with the processing. Otherwise, we discard the file (error type 6). This is done to homogenize the output database, as we are interested in studying a specific user profile.
5. We use the `discard_coordinates()` function to determine whether the input file should be processed or not. This function:
    - Verifies that all coordinates are within the bounds of the defined area. Otherwise, we discard the file (error type 1).
    - Ensures that there are at least 100 coordinates in the data frame. If not, we discard the file (error type 2).
    - Checks that no two consecutive coordinates are more than 300 meters apart. If they are, we discard the file (error type 3).
    - Verifies that the total distance of the route exceeds 1000 meters. Otherwise, we discard the file (error type 4).

6. Now, if the previous function allows us to proceed with the file, we perform the map matching process using the `matching_track()` function, which takes the coordinate data frame and the defined model as input and returns the result (in `fmm` format), the parameters used, and any error that occurred.
   - The parameters used to call `FMM` will initially be: `k=2`, `radius=0.001` (approximately 100 meters), and `gps_error=0.001` (approximately 100 meters).
   - We convert the coordinates in the input data frame into a suitable format and apply the `match_wkt()` function from the library.
   - If a result is found, we return it. If not, we increase the value of `k` (up to `k=4`). We also set a timeout of 60 seconds for the function to run.
   - If we are unable to find a matching track, we return that the file must be discarded (error type 5).
7. If the track has been successfully processed, we save the waypoints to the `waypoints.csv` data frame, shared across all routes in a zone. We use the `waypoints_df_cleaning()` library to clean the input data frame and keep only relevant information.
8. With the `save_fmm_output()` function, we use the result file from the map matching process and save it in *CSV* format to the `/Data/Output-Data/FMM-Output` folder. For each matched point, we store the coordinates and also the edge it falls on (`u` and `v`, as in the edges data frame — these values are also reordered, as before).
9. Finally, we save the data to the `output_files.csv` data frame, or the discarded files data frame `discard_files.csv`.

Once this second process has been executed, we will have filled the `/Data/Output-Data/FMM-Output` folder with *CSV* files for all tracks that were successfully processed. We have also filled the `waypoints.csv`, `output_files.csv`, and `discard_files.csv` data frames. With all of this data, we can now proceed to the third part of the processing.

5.3 Data Postprocessing

The third and final part of the data processing is focused on cleaning the data extracted in the second part of the general process. It is based on two procedures: the first processes each output track from `FMM` and cleans it by extracting relevant information; the second revisits all these files and completes the edges data frame. It is necessary to check the output tracks twice, as some files may contain errors in the first step and need to be discarded. The entire process can be found in the following link.

As previously mentioned, the first sub-process cleans each track extracted earlier:

1. Initially, we read the original *JSON* file and obtain information such as `url`, `user`, `date` (which is converted to the correct format using `convert_date()`), and `difficulty`.
2. We use the `clean_coordinates_df()` function to clean the coordinates data frame. This function takes as input the initial coordinates data frame, the output from `FMM`, and the edges data frame. Check this example of output data frame.
   - From the initial data frame, we take the coordinates, elevation, and timestamp. We add new columns such as `dist` (accumulated distance as coordinates progress—calculated using `geodesic`), `km` (containing the kilometer segment in which each coordinate lies). We also compute the accumulated time in minutes (`time` column), corresponding to the timestamp differences (in milliseconds); and the `pace` column, in kilometers per hour.
   - Using the output from `FMM` (which contains coordinates and edge points), we perform a merge with the edges data frame to obtain the identifier of the traversed edge.
   - The *clean_single_edges()* function is applied to the merged data frame. This prevents isolated edges (which may have been visited only once due to library errors). We insert the previous edge identifier.
   - We concatenate this output data frame with the initial one and create an `id` column that enumerates each coordinate point.
   - We return, along with the data frame, the total distance, elevation gain and loss, total time, average pace, and the first and last coordinates (information that we will add to the shared dataframe).
3. Once the data frame is processed, we check that the timestamps are not all the same. If they are, we discard the track with error type 6.
4. We apply a filtering step to further homogenize our data. We keep only those tracks with a distance between 3 and 30 kilometers and with elevation gain/loss below 5000 meters. If the track passes this filter, we save it in `cleaned_out.csv`; otherwise, we discard it (error type 6).
5. If the file is not discarded, we add the extracted extra information—total distance, total time, etc.—to the shared data frame `cleaned_out.csv`.

Once the routes are processed, we enter the `cleaned_out.csv` data frame to transform it as desired. This step is performed afterward since some files may have been discarded earlier.

1. We convert the date into `pandas datetime` format and extract variants such as year, month (in letters), day of the week, and season.
2. Using the `obtain_weather_dataframe()` function, we obtain a data frame with minimum and maximum temperature, and weather condition, from the earliest to the latest route date in the specified area. We use the `requests` library to access an API that returns this information.
3. We merge the two data frames to obtain weather data for each date.
4. Finally, we save the data frame.

At this point, all routes have been successfully processed. The only remaining step is to process the `edges.csv` data frame. We do this as follows:

1. For each route in the processed data frame, we identify all the edges that were traversed and store them in a list.
2. For each edge in this list, we add the track identifier to the edges data frame and increment the counter.
3. We trim the edges data frame so that it includes only those edges that were traversed at least once.
4. We compute the `weight` column, which is a normalization of the previous count on a scale from 1 to 10.

5.4 Input Data

5.4.1 Data Folder Structure

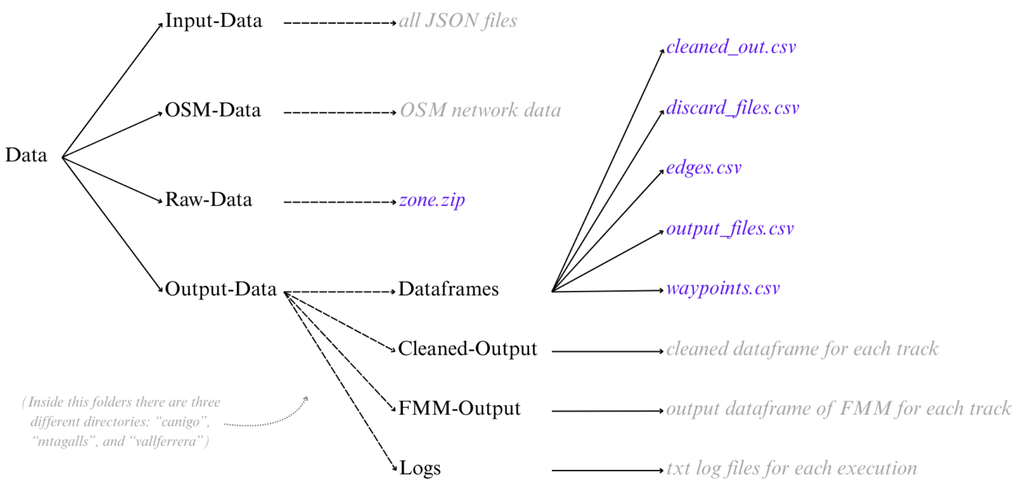

Data
Input-Data
all JSON files
OSM-Data
OSM network data
Raw-Data
zone.zip
Output-Data
Dataframes
cleaned_out.csv
discard_files.csv
edges.csv
output_files.csv
waypoints.csv
Cleaned-Output
cleaned dataframe for each track
FMM-Output
output dataframe of FMM for each track
Logs
txt log files for each execution
(Inside this folders there are three different directories: "canigo", "mtagalls", and "vallferrera")

Figure 10: Data directory structure.

5.4.2 Route Metadata

Column Name	**Meaning**	**Example**
`url`	Link to the *Wikiloc* web page with the track.	Link
`user`	User numeric identifier of the creator of the route.	*822421*
`track`	Track numeric identifier.	*187570034*
`title`	Title of the track.	*Coll de les Basses*
`date-up`	Date of registration of the track in catalan.	*7 d'octubre de 2024*
`date-track`	Month and year of registration of the track in catalan.	*d'octubre 2024*
`activity_type`	Identifier of the type of activity.	*1*
`activity_name`	Name of the type of activity.	*Senderisme*
`difficulty`	Difficulty of the track (inserted by the user).	*Moderat*
`waypoints`	See Points of Interest Metadata.	-
`coordinates`	See Coordinates Metadata.	-

Table 1: Example of route metadata fields from *Wikiloc* (Example track 187570034).

5.4.3 Points of Interest Metadata

Column Name	Meaning	Example
`elevation`	Elevation of the coordinate point in meters.	*1790*
`i18nTxtElevation`	Elevation as string.	*altitud 1790 m*
`id`	Numerical identifier for the point of interest.	*187570035*
`lat`	Coordinate latitude in degrees.	*42.439554*
`lon`	Coordinate longitude in degrees.	*2.423168*
`name`	Name that the user has inserted.	*null*
`photos_id`	Numerical identifier of the picture (for each picture inserted - maybe *None*).	*116128730*
`photos_prettyUrl`	Pretty URL of the picture (for each picture inserted - maybe *None*).	*/rutes-senderisme/...*
`photos_url`	URL of the picture (for each picture inserted - maybe *None*).	*Link*
`pictogramId`	Identifier of the type of point of interest.	*36*
`pictogramName`	Type of point of interest.	*Foto*
`prettyUrl`	Pretty URL.	*/rutes-senderisme/...*

Table 2: Example of waypoint metadata fields (Example track 187570034).

5.4.4 Coordinates Metadata

Column Name	Meaning	Example
`longitude`	Coordinate longitude in degrees.	*2.423061*
`latitude`	Coordinate latitude in degrees.	*42.43952*
`elevation`	Elevation of the coordinate point in meters.	*1790.7*
`timestamp`	Unix timestamp. The difference between two consecutive timestamps is the elapsed time in milliseconds.	*-524173384*

Table 3: Example of coordinate metadata fields (Example track 187570034).

5.5 Output Data

5.5.1 FMM Output Dataframe Columns

Column Name	Meaning	Example
`track_id`	Track numeric identifier.	*187589120*
`k`	Number of candidates parameter used in the `FMM` algorithm.	*2*
`radius`	Search radius parameter used in the `FMM` algorithm.	*0.001*
`gps_error`	GPS error parameter used in the `FMM` algorithm.	*0.001*

Table 4: Example of FMM output dataframe columns (Example).

5.5.2 Discarded Files Dataframe Columns

Column Name	Meaning	Example
`track_id`	Track numeric identifier.	*187570034*
`zone`	Zone where the track has been recorded.	*canigo*
`error_type`	Type of error (from 1 to 6).	*5*

Table 5: Example of discarded files data frame columns (Example).

5.5.3 Edges Data Frame Columns

Column Name	Meaning	Example
`id`	Edge numeric identifier.	*392*
`u`	Numeric identifier of the source node of the edge.	*36872472*
`v`	Numeric identifier of the target node of the edge.	*36872475*
`geometry`	Geometry line of the edge (as `LINESTRING()`).	*LINESTRING(...)*
`total_tracks`	Counter of the total tracks that pass through that edge.	*1*
`list_tracks`	List with all the tracks identifiers that pass through that edge.	*['187697056']*
`weight`	Normalized number from 1 to 10 of the `total_tracks` column.	*1*

Table 6: Example of edges data frame columns (Example).

5.5.4 Cleaned Output Data Frame Columns

Column Name	Meaning	Example
`track_id`	Track numeric identifier.	*187591820*
`url`	Link to the *Wikiloc* web page with the track.	Link
`title`	Title of the track.	*Pic de Très Estelles*
`user`	User numeric identifier of the creator of the route.	*2094809*
`difficulty`	Difficulty of the track (inserted by the user)	*Fàcil*
`distance`	Total distance of the track (in km).	*7.72*
`elev_gain`	Total elevation gain of the track (in meters).	*486.9*
`elev_loss`	Total elevation loss of the track (in meters).	*490.3*
`date`	Date of registration of the track (format `dd/mm/YYYY`).	*2024-10-08*
`year`	Year of registration of the track.	*2024*
`month`	Month of registration of the track (as string).	*October*
`weekday`	Day of the week of registration of the track (as string).	*Tuesday*
`season`	Season of registration of the track.	*Autumn*
`min_temp`	Minimum temperature in the zone on the date when the track was registered (in Celsius scale).	*1.8*
`max_temp`	Maximum temperature in the zone on the date when the track was registered (in Celsius scale).	*8.8*
`weather_condition`	Weather condition in the zone on the date when the track was registered (as string).	*Rain*
`total_time`	Total time of the track (in minutes).	*134.52*
`avg_pace`	Average pace during all the track (in meters per second).	*4.15*
`first_lat`	Latitude of the first coordinate of the track (in degrees).	*42.481365*
`fist_lon`	Longitude of the first coordinate of the track (in degrees).	*2.314325*
`last_lat`	Latitude of the last coordinate of the track (in degrees).	*42.481378*
`last_lon`	Longitude of the last coordinate of the track (in degrees).	*2.31444*

Table 7: Example of cleaned output data frame columns (Example).

5.5.5 Waypoints Data Frame Columns

Column Name	Meaning	Example
`waypoint_id`	Waypoint numeric identifier.	*187591821*
`track_id`	Track numeric identifier where the waypoint has been registered.	*187591820*
`lat`	Coordinate latitude of the waypoint (in degrees).	*42.481254*
`lon`	Coordinate longitude of the waypoint (in degrees).	*2.314285*
`elev`	Elevation of the waypoint.	*1758*
`type`	Type of point of interest.	*Collada*
`url`	URL of the first photo (if any pictures).	Link

Table 8: Example of waypoints data frame columns (Example).

5.5.6 Cleaned Track Data Frame Columns

Column Name	Meaning	Example
`id`	Coordinate point numeric identifier.	*1*
`edge_id`	Numeric identifier of the edge where the cordinate point pass through (found with `FMM`).	*11074*
`lon`	Longitude of the coordinate point (in degrees).	*2.314325*
`lat`	Latitude of the coordinate point (in degrees).	*42.481365*
`elev`	Elevation (in meters).	*1760.4*
`dist`	Total distance until that coordinate point (in km).	*0.0*
`km`	Actual kilometer of the coordinate point (from 0 to total kilometers of the track).	*0*
`time`	Total time until that coordinate point (in minutes).	*0.0*
`elev_diff`	Elevation difference between that coordinate point and the previous one (in meters).	*0.0*
`pace`	Pace on that coordinate point (in meters per second). Distance between the two coordinates divided by the elapsed time.	*0.0*
`clean_lon`	Coordinate longitude found with `FMM` (in degrees).	*2.314328*
`clean_lat`	Coordinate latitude found with `FMM` (in degrees).	*42.481289*

Table 9: Example of a cleaned track data frame columns (Example track 187591820).