OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

Pub Crawl Recommendation System

Applied Data Science with Python

Date: 19.11.2025

Student: Daniel Sowada
Matr. Nr.: 3398704

Contents

1 Introduction

The goal of this project is to develop a *Pub Crawl Recommendation Application* for the city centre of Regensburg (Altstadt). Users provide their current location, the number k of bars they want to visit, and optional preferences. The preferences depend on the quality of data input from the API's and is not certain yet. There will be some evaluation criteria consisting of the distance from your location to the bar, the opening hours, food choices or bar type (e.g., sports bar, student bar, dance bar, karaoke bar, jazz bar). The system generates an optimised route that visits k suitable bars in an walking distance, depending on the recommendation scores.

This document explains how I intend to solve the problem in terms of the **algorithm**, **tools**, **modules** and **data structures** used in the implementation.

2 Tools

To implement the project, I will primarily use the Python ecosystem. The main tools are:

- **Python 3.12.3** as the main programming language.
- **VS Code** with the Python
- **Jupyter Notebooks** for early experimentation and data exploration.
- **OpenStreetMap / Overpass API** to fetch bar locations and tags (e.g., `amenity=bar`, `pub`, `nightclub`).
- **Streamlit** to build a simple web-based user interface for interacting with the recommendation systemand displaying the result

These tools allow efficient data access, processing and visualisation.

3 Modules

The following Python modules are planned for use in the implementation:

- **requests** – to call the Overpass API and obtain raw JSON data describing bars in the Regensburg Altstadt.
- **json** – to parse the API responses into Python objects.
- **pandas** – to clean and organise the bar data in tabular form and to perform filtering and feature engineering.
- **geopy.distance** – to compute distances between latitude/longitude coordinates (e.g., between user and bars, or between different bars).
- **networkx** – to model the bars as a graph and to compute shortest or near-shortest paths for the pub crawl.

- **folium** – for displaying the ideal route from bar to bar

It is possible that the modules here will differ at the end of the project, because the implementation process did not finish. All changes will be added to the final assignment.

4 Data Structures

The project will mainly rely on the following data structures:

4.1 Pandas Dataframe

All bars fetched from OpenStreetMap will be stored in a `pandas.DataFrame` with columns such as:

Table 1: Structure of the bar Dataframe

Column	Type	Description
name	string	Venue name
lat, lon	float	Coordinates of the bar
tags	dict	OSM metadata (amenity, cuisine, music etc.)
opening_hours	string / parsed object	Extracted from tags
dist_user	float	Distance to user location (meters)
type_derived	string	Normalised bar category
score_total	float	Final overall recommendation score

This structure makes it easy to filter bars, compute scores and export to CSV if needed.

4.2 Dataframe of suitable bars

Intermediate results, such as the list of bars matching user preferences, will be stored in panda dataframes whit all the necessary informations before they are passed into the graph-based route computation.

5 Algorithm

This section presents the planned algorithmic workflow for the pub crawl recommendation system. Since the final implementation may evolve based on data availability and usability testing, the description reflects a flexible and structured design.

The algorithm section consists of four major phases: data acquisition, preference scoring, graph construction, and route optimisation.

5.1 Data Acquisition

The system will begin by retrieving venue information from the Overpass API for the Regensburg Altstadt. The obtained JSON data is normalised into a `pandas.DataFrame`, and relevant attributes such as geographical coordinates and opening hours are extracted.

Because API queries may fail due to rate limits or connection issues, the system will include a cached fallback dataset to ensure stable operation and reproducible results.

5.2 Preference Filtering and Scoring

To determine which bars best fit the users preferences, the system assigns a preference score to each venue. The scoring model follows a sum formulation, allowing flexible adjustment of the importance of each criterion based on usability considerations. This needs to be adjustedin the end.

For any bar b, the overall score is computed as:

$$\text{score}(b) = w_{\text{dist}} \cdot s_{\text{dist}}(b) + w_{\text{open}} \cdot s_{\text{open}}(b) + w_{\text{pref}} \cdot s_{\text{pref}}(b),$$

where each term consists of a weight w_i and a corresponding sub-score $s_i(b) \in [0, 1]$. The weights can later be tuned to reflect user feedback or empirical performance.

The following sub-scores are currently planned:

- **Distance Score:** Distances are normalised to the interval $[0, 1]$, such that closer bars receive higher values:
$$s_{\text{dist}}(b) = 1 - \min\left(1, \frac{d(b)}{d_{\max}}\right).$$
- **Opening-Hours Score:**
$$s_{\text{open}}(b) = \begin{cases} 1, & \text{if the bar is open at the planned visit time,} \\ 0.75, & \text{if the bar closes in the next 3 hours.} \\ 0.5, & \text{if the bar closes in the next 2 hours.} \\ 0.25, & \text{if the bar closes in the next 1 hour.} \\ 0, & \text{if it is closed,} \end{cases}$$
- **Optional Preference Score:** This term allows integrating additional user-defined preferences, such as bar type, atmosphere, or food availability. Since these attributes may not always be available in the data, this part of the scoring model is optional:
$$s_{\text{pref}}(b) = \begin{cases} 1, & \text{if the bar matches the user preference,} \\ 0.5, & \text{if information is missing,} \\ 0, & \text{if the bar does not match the preference.} \end{cases}$$

This additive scoring structure ensures high flexibility: the weights $w_{\text{dist}}, w_{\text{open}}, w_{\text{pref}}$ can be freely adjusted during development or testing to improve recommendation quality. The model for the preference score can also be enhanced.

5.3 Graph Construction

After selecting the top k bars, the system needs a way to organise their locations so that an efficient route can be determined. To achieve this, the bars and the users starting position are represented in a structure that captures how each location is connected to the others.

This representation stores which bars can be visited from which others and how far apart they are. It does not rely on any specific algorithm or library at this stage; instead, it provides a general foundation on which different route finding methods can be applied later. The main idea is simply to convert the geographic information of the bars into a form that allows systematic comparison of possible paths.

5.4 Route Optimisation

Once the bars are organised in this structure, the next step is to determine a sensible order in which they should be visited. Since visiting all possible permutations would be inefficient, especially as more bars are added, the system will rely on a simplified optimisation strategy.

Instead of computing the absolute mathematically optimal route, the goal is to find a route that is "good enough" for practical use. Possible approaches include following the nearest suitable bar at each step or using an approximation method that produces short routes. The exact choice of method may depend on how well different approaches perform during implementation.

The final output of this phase is a recommended sequence of bars that forms a efficient pub crawl route starting from the users current location.