Should I get the yearly pass (Jahreskarte) for the Viennese public transit system?

Example project using JSON, python, web API, and SQLite

Roni Hogri, August 2023

Background

Many people in Vienna take advantage of the city's excellent public transit system. For frequent travelers it is often advantageous to purchase a yearly pass (Jahreskarte) rather than paying for a single-ride ticket each time. However, many people use the public transit system only occasionally, making it difficult to decide whether purchasing the Jahreskarte would be cost-effective. To address this issue, I developed a procedure utilizing Googles location history data (JSON files), python, a web API for reverse geocoding, and SQLite.

How it works

The python program 'annual_ticket_calculation_from_google_data.py' (henceforth: 'main program') extracts location history data collected from your cell phone by Google to an SQL database. This data is then used to determine whether a yearly pass would have been more cost-effective than purchasing single-ride tickets during the period in question.

Note: In order to protect sensitive information, the main program works on a redacted version of the Google data. This version does not include details such as device ID and place IDs, and the spatial resolution of geographical coordinates has been reduced. To produce such redacted files, use the 'json_redact.py' program (see step 3 below).

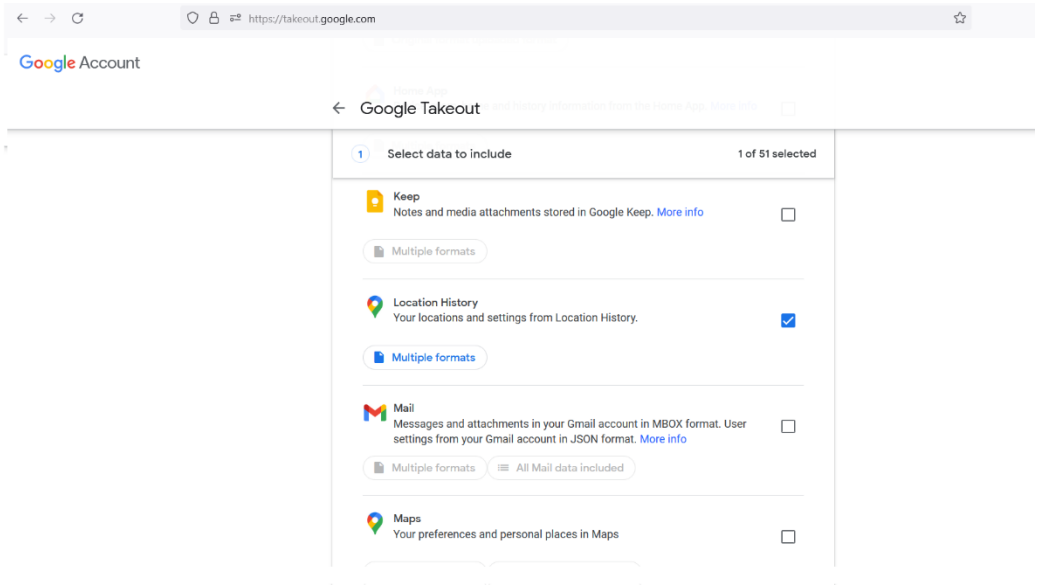
Additional detailed documentation is provided as comments in the python programs.

Steps (see pictures and comments below)

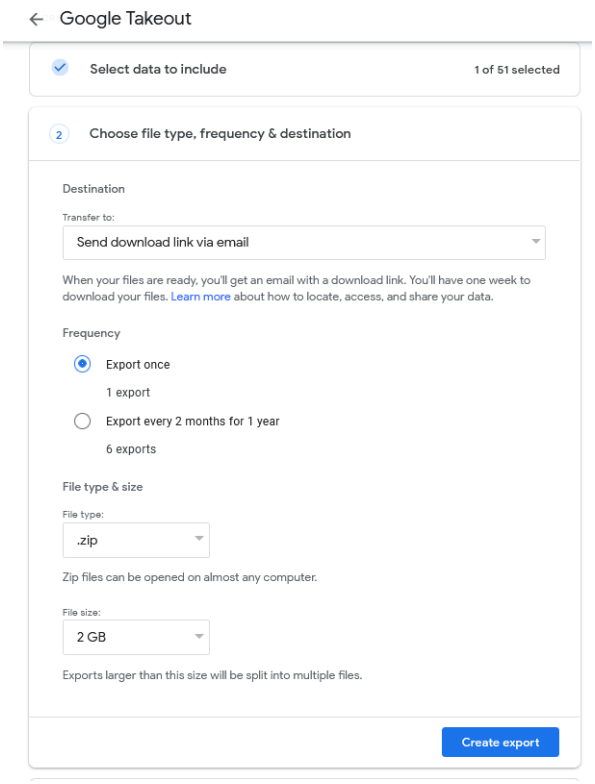
1. Connect to your Google account and download your location history from https://takeout.google.com/
2. Unzip the folders you want to analyze, under 'Location History\ Semantic Location History'. It's possible to select data from one year, or from multiple years.
3. Run the 'json_redact.py' python program to remove sensitive and unnecessary data from your location history.
4. Run the main python program ('annual_ticket_calculation_from_google_data.py') and follow the instructions you receive.

Note: If you prefer not to download your own location history data, and would like to test the main program works on my example data, skip steps 1-3 and run the main program on the provided 'Redacted' folder (provided as a Zip file). The 'Redacted' folder already contains an SQLite file containing the information extracted from the provided JSON files ('2023_SQL.sqlite'). If you want the main program to export this data to a new SQLite file, please delete or rename this file.

Step 1: Download location history from your Google account

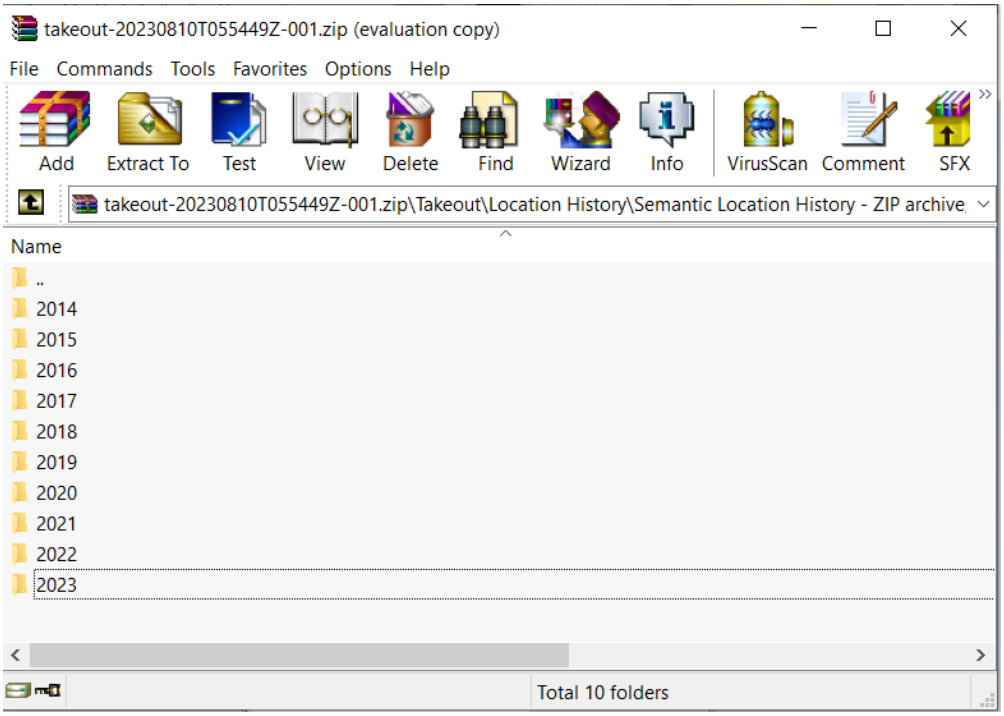


1.1 Go to https://takeout.google.com/ , select 'Location History', and then scroll down and click on 'Next step'.



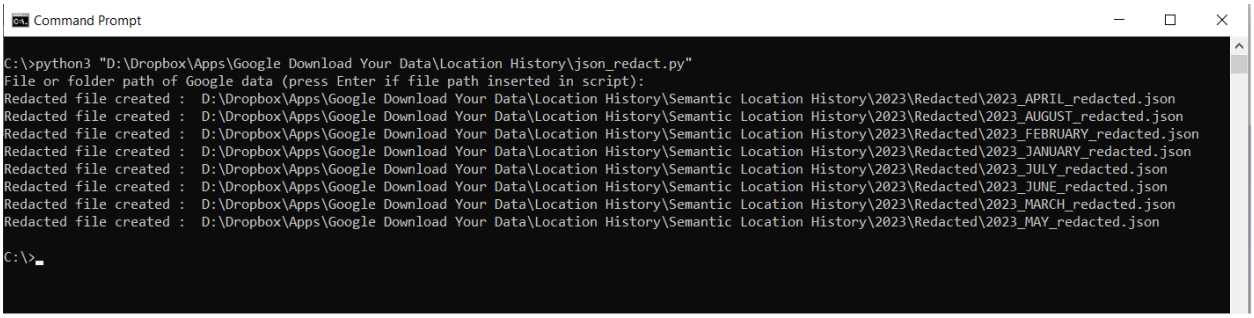
1.2 Choose your preferred transfer method (e.g., via email). Mark 'Export once', and set 'File type & size' according to your preferences. Then click on 'Create export'.

Step 2: Unzip your location history data



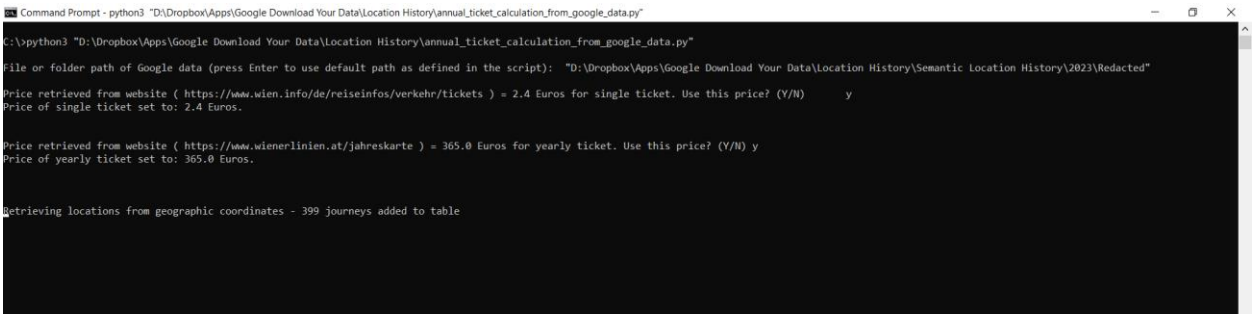
Example of zipped location history.

Step 3: Use the 'json_redact.py' program to edit JSON files obtained from Google

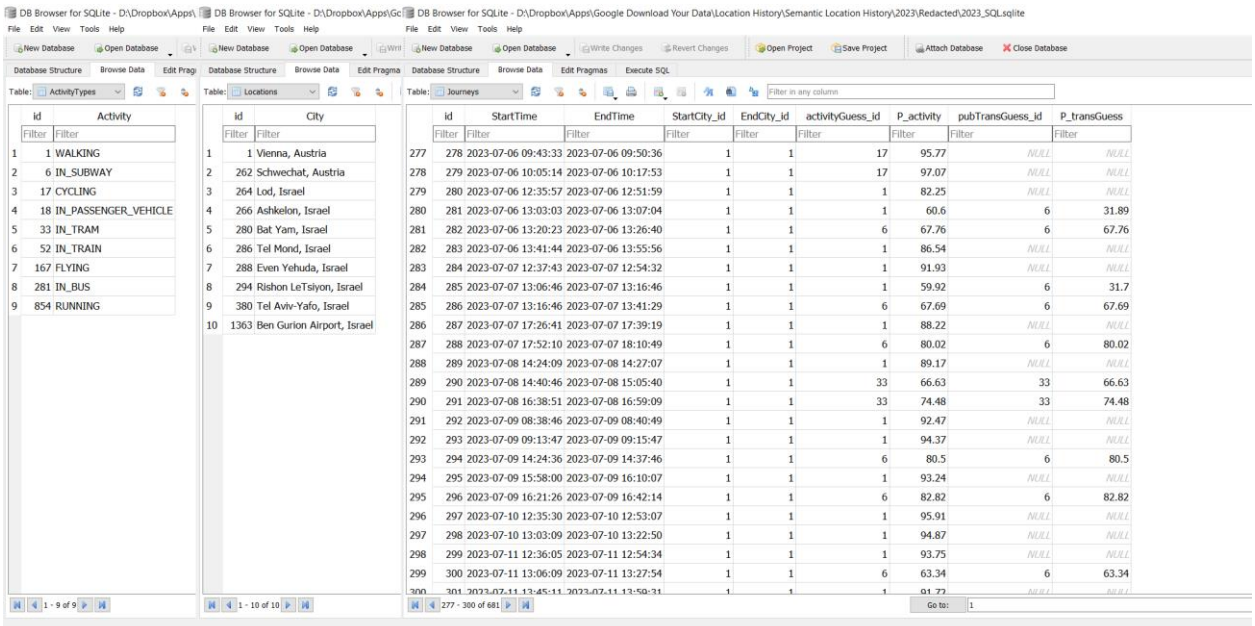


Output of the 'json_redact.py' program, showing the paths of the newly created redacted location history JSON files.

Step 4 (using the main python program):



4.1 When you activate the main program, the prices for the different ticket types will be retrieved from the web and displayed. Once prices are set, the program will start retrieving locations based on the low-resolution geographic coordinates, using a mock-version of the Google API provided by Dr. Charles Severance (http://py4e-data.dr-chuck.net/json?). I chose to use this API since it doesn't require each potential user to obtain their own Google API key. The process of retrieving locations in this way might take some time, so progress is continually displayed. It is possible to restart the program multiple times (e.g., after downloading and redacting additional data) – unique journeys will be added to the SQL table as required. Note that the provided 'Redacted' folder already contains a filled SQLite file ('2023_SQL.sqlite') – to fill the SQL data yourself, please delete or rename this SQLite file.

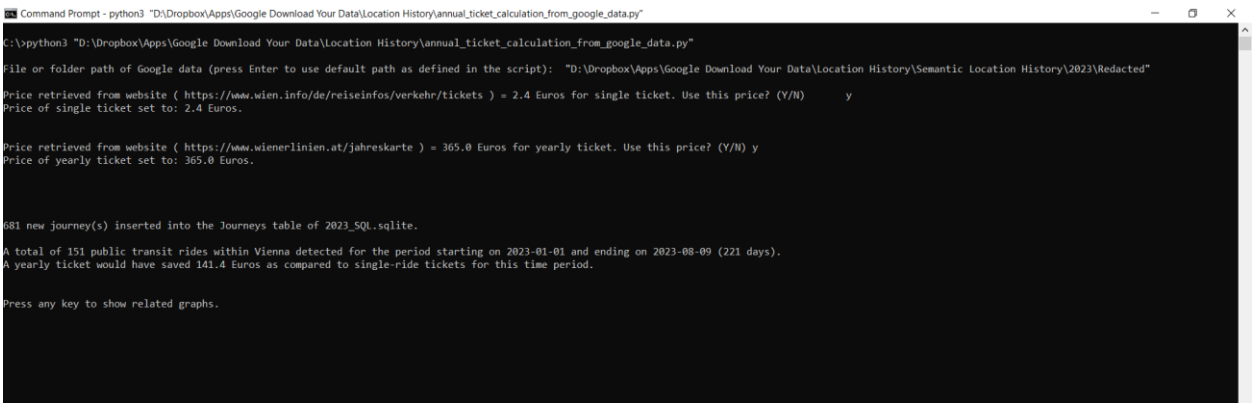


4.2 Example SQL data showing the tables created. The main table is called 'Journeys' (shown on the right). Columns ending with '_id' refer to variables stored in the other tables.

'StartCity_id' and 'EndCity_id' refer to the locations in which journeys began and ended, respectively; the main python program uses this information to decide whether public transit trips were within Vienna (otherwise, other types of tickets, with different prices, would be required).

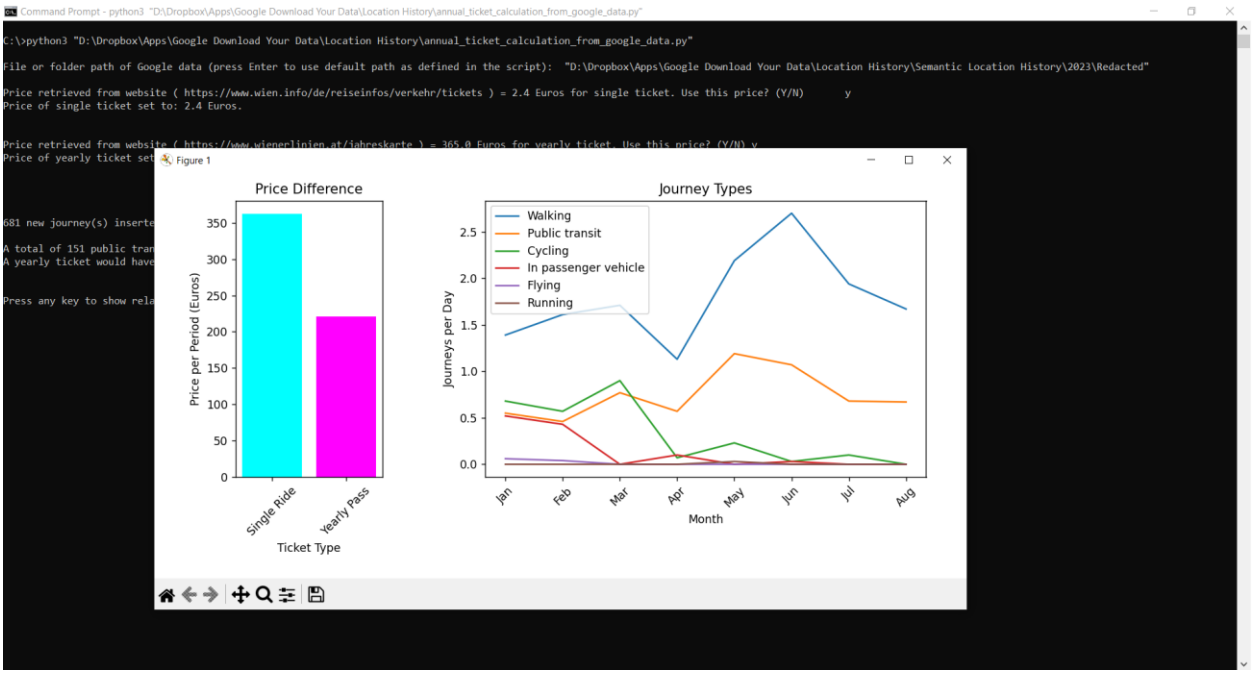
'activityGuess_id' refers to Google's best guess regarding the type of journey recorded (see 'Activity' table, on the left). Google's level of confidence regarding this guess (as %) is shown under 'P_activity'. 'pubTransGuess_id' and 'P_transGuess' refer to Google's best guess involving public transit use for each journey and its confidence level, respectively. For both types of guesses ('activityGuess_id' and 'pubTransGuess_id'), only journeys where the confidence level ('P_activity' and 'P_transGuess', respectively) was higher than the set threshold were included in the table. Threshold value can be adjusted in the python main program as required; I set it to 30%, since this seemed to work well in cases where the ground truth was available to me. Note that, depending on Google's confidence in its guesses, the 'activity_Guess_id' is often, but not always, identical to the 'pubTransGuess_id' of a given journey.

'StartTime' and 'EndTime' refer to the beginning and end times of journeys, respectively; this is important for determining the duration of a journey and for deciding whether multiple journeys actually belong to the same trip for which one single-ride ticket would suffice (e.g., subway + tram).



4.3 Once all journeys in the selected JSON files have been stored in the SQLite file, the program reports how many new journeys were inserted into the SQLite file during the current run. It then provides a short summary of the information extracted from the existing SQLite file.

The program tells you whether a yearly ticket (Jahreskarte) would be more cost-effective than using single-ride tickets for the selected periods, and how much money would be saved by using the more cost-effective method for the selected time period.



4.4 The program displays two graphs showing relevant information obtained from Google's location history. Left: A comparison between single-ride and yearly pass tickets for the selected period. Right: The normalized frequency of different kinds of journeys during the selected period, to facilitate the identification of travel mode trends over time.