**QGIS Process in order to run the full model:**

This document aims to describe the full processing of a model on QGIS. Some parts have already been automatized with Python (Shortest path, ranking), but there is still a lot of programming work to automatize the different steps. The actual used codes for QGIS have been uploaded on GitHub. The work can be summarized in three main parts: data pre-treatment, where we open the different layers and prepare them for the different calculations. Then, demand calculations, which use a shortest-path algorithm to run a full 4-step model. Then demand is compared to supply to prioritize investments.

1. *Data pre-treatment*
2. **Input spatial data-sets**

Network (NVDB)

Actual bicycle infrastructure (NVDB)

DeSO population (SCB)

Points of interest (OSM)

Work and School relationships (SCB, Not free!)

Naturreservat (Läntmateriet)

1. **Reproject layers to EPSG:3006 (OSM only)**
2. **For non-point data, create centroids**

Normally, only DeSO and Naturreservat

1. **For OSM points of interest: separate them according to the type of trips**

Create four sub-layers:

* Leisure
* Services
* Shopping
* Touring (merge this one with Naturreservat layer)

1. **Create a unique ID for the DeSO (beginning at 1), type: Integer**
2. **Create a unique ID for the Destinations (beginning at 1+Number of DeSO) for each purpose**
3. **Create layers with merged origins and destinations**
4. **Create a “relation” vector between each origin and destination (can be done with QNEAT3)**
5. **Merge the network into one feature with “Dissolve”**
6. **Split the network with “Split lines by length”, with a given length (e.g. 500m),**

Give a unique $id to each segment (from 1 to …), and calculate the $length

1. **Create a buffer of 30m around the network**
2. *Run the calculations for demand*
3. **Run the Shortest path Python algorithm for each purpose**
4. **Join the origins sizes to the Shortest-path outputs, or the number of people per relation in the case of commuting trips (joins and relates)**
5. **Use the found distance to apply the gravity and mode choice functions, multiply by the origin size**

This is done by creating new fields, it has already been programmed on Python (“decay.py” for school and work, “Tij.py” for the other purposes)

only thing that didn’t work for now: the origin constraint of the Gravity model

1. **Calculate the flows using “Join attributes by location (summary)”**

Parameters are: “crosses”, fields to be summarized: Tij\_bike and Tij\_ebike, operation: “Sum”

NB: This can be stored as a .json file

1. **Join the results to the network (joins and relates)**

Then the user will be able to plot the maps of demand

An aggregate demand can be calculated by summing all the flows, weighted by “bike” or “ebike”

1. *Compare supply with demand: priorization*
2. **Thanks to the decision table, calculate the needed supply on the original network (python)**
3. **Given the flow, run the code “ranking.py”, which gives the list of the links with the highest potential for each kind of infrastructure**
4. **Copy the list of ID resulting to create sub-layers for each kind of infrastructure.**
5. **Priorization can be done by calculating the product Flow \* log2(LTS) using the attribute table field calculator**