IKOB documentation

Table of Content

Introduction

The IKOB model takes into account the difference in income classes and therefore ability/willingness to pay for different modes of transport. This results in different attractiveness to live/work in certain zones depending on the income classes. CROW wants this to become usable for the whole of the Netherlands.

This is a working document that will develop with the developments of the software. The version of the software that this draft was made for still has a lot of Dutch names, variables etc. Therefore you will encounter those terms while reading. Every chapter describes one of the scripts therefore the chapter-names are in accordance to the script names. The flowcharts are a combination of the folder structure and naming convantions as well as an illustration of in- and output.

Prerequisites

In order for these series of script to run on your computer you need:

- python 3.10 (or later otherwise the wizards don't work) this can be downloaded online
- pip
- the python module XlsxWriter run 'py -m pip install XlsxWriter' in cmd

Python needs full read-write access to your folder structure.

Input files and folder structure

Every model needs input data. So does IKOB. We need socio-economic data. These "SEGS" need to be put in a folder called 'SEGS'. The files need specific naming:

- Inwoners_per_klasse[scenario year].csv (number of inhabitants per incomeclass)
- Arbeidsplaatsen_inkomensklasse[scenario year].csv (number of jobs per incomeclass)
- Inkomensverdeling_per_zone (distribution in percentage per incomeclass per urbanizationclass)
- CBS_autos_per_huishouden (number of cars per household)
- Beroepsbevolking (number of laborers)
- Stedelijkheidsgraad (urbanizationclass) a number between one to five in accordance to CBS
- GeenRijbewijs (percentage distributen of inhabitants without a drivers licence per incomeclass per urbanizationclass)
- GeenAuto (percentage distributen of inhabitants without a car per incomeclass per urbanizationclass)
- WelAuto (percentage distributen of inhabitants with a car per incomeclass per urbanizationclass)
- Voorkeuren (percentage distribution of preferred mode of transport per urbanizationclass)
- VoorkeurenGeenAuto (percentage distribution of preferred mode of transport per urbanizationclass for inhabitants without a car)

Then there is the traffic model information, this needs to be in the folder 'skims' under the respective [part of day] subfolder:

- Auto_Tijd (driving time with car in minutes)
- Auto_Afstand (driving distance with car in km from zone to zone)
- Fiets_Tijd (biking time in minutes)
- OV_Tijd (public transportation time in minutes)
- OV_Afstand (public transportation distance in km)
- OV_Kosten (public transport costs in eurocents) ask if correct

These times are pure travel time (without waiting time etc).

Last but not least there is a file needed that indicates the 'time searching for parking per zone'. You can give it a name you like and put is anywhere.

- Parkeerzoektijdfile (arrival en departure searching time for car in minutes per zone)

For visualizations (maps) it is also necessary to have a file that contains the zones and their geometry. For example: If the data is based on neighborhoods you can use a shapefile with the zone-numbers to visualize the input and output data.

Format

All input files are .csv comma delimited. Zones can be a shapefile or based on neighborhood data for example.

Config file

To start a run you need to first create the config file. Which will point to the correct folder and files.

1. RUN ikobconfig.py
2. Give a name to the Project (scenario description for example)
3. Add the year at verstedelijkingsscenario
4. `Basis directory` is where you want the scripts to make output folders.

5. `SEGS directory` is where you have the SEGS
6. Change modes in accordance to your scenario.
7. Under the tab 'Ervaren Reistijd Berekenen' you need to navigate to the Parkeerzoektijden file. Here you also need to select one or multiple 'parts of day'. It is very important that you put the input file in the consecutive subfolder:
8. In you `Basis directory` make (for example) `Restdag` folder. Here you put your Restdag skims.
9. The other settings you can alter in accordance to your scenario.
10. Save.

Running the model

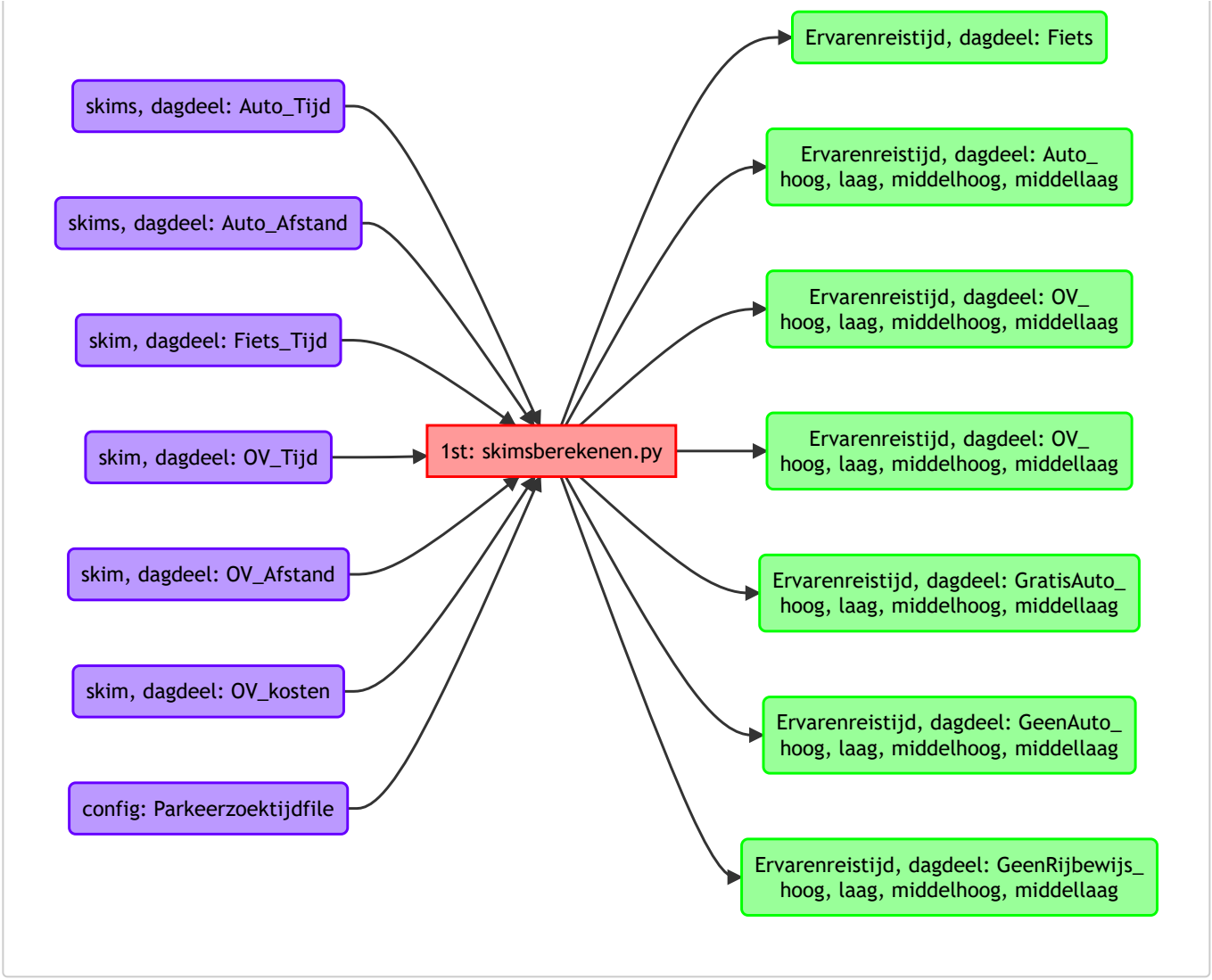
Run ikobrunner.py
Navigate to the json you made in the previous step.
Here you can select the steps you want to take.

1. Ervaren reistijd bereken uit tijd en kosten (runs skimsberekenen.py) generates experienced travel time
2. Verdeling van de groepen over de buurten of zones (runs verdelingovergroepen.py) distributes the various groups over the zones.
3. Gewichten (reistijdvervalscurven) voor auto, OV, fiets en E-fiets apart (runs Gewichtenberekenenenkelscenarios.py) generates the subfolder `Gewichten`(weights) for car, public transport, bike and E-bike seperately
4. Maximum gewichten van meerdere modaliteiten (runs Gewichtenberekenencombis.py) takes the maximum weight of a combination of modalities resulting in the subfolder `Gewichten combinaties` (weight combinations).
5. Bereikbaarheid arbeidsplaatsen voor inwoners (runs Ontplooiingsmogelijkhedenechteinwoners.py) calculates the accesability of jobs for inhabitants in the subfolder `Bestemmingen`.
6. Potentie bereikbaarheid voor bedrijven en instellingen (runs Potentiebedrijven.py) calculates the accesabilitie of labour force for employers in the subfolder `Herkomsten`
7. Concurrentiepositie voor bereik arbeidsplaatsen (runs Concurrentieomarbeidsplaatsen.py) calculates the competitive position of inhabitants to reach a job
8. Concurrentiepositie voor bedrijven qua bereikbaarheid (runs Concurrentieominwoners.py) calculates the competitive positon of a job/bussness to be reached.

Outputs per script

Skimsberekenen

skimsberekenen.py puts files in `Ervarenreistijd`(experienced travel time) then the 'part of day' you selected: Ochtendspits(morning-rush), Avondspits(evening-rush), Restdag(other). For every income group a separate Auto_ OV_ etc will be generated. This script takes files from the `skims` folder and the folder you indicated in the config that the parking time files is located in.



[add formula] This script generates experienced travel time. Therefore the input data has to be the pure travel time. No waiting time etc. The script takes into account that acceptable comfort levels differ per income group. There is no scientific basis for this yet.

Eurocent starting-tariff, eurocent/km variable costs and variable costs car in eurocent per km are divided by 100. Therefore become `Euro` and `Euro per km`.

Next a **PT cost matrix** in `Euro` is generated by multiplying PT distance in km by (km-tariff (€/km) added with the starting tariff in euro).

If the travel time is more than 2 hours (180 min) for **bikes** the weight-matrix value is set to 9999. Bike travel time longer that 2 hours are ignored.

For **car** the matrix is calculated the following way:

1. Total time in `minutes` consist of: car-traveltime-matrix plus parking-search-time.
2. GGR skim is *totale tijd* in `minutes` plus **factor** time car distance in `km` times (car-tariff `€/km` + km-charge `€/km` from the config file)
3. **Factor** is per incomeclass and otherwise you entered a static value in the config file:`minute per €`.

For **Public Transport** the PT cost-matrix is multiplied with the multiplication-fraction generated during the PT time matrix step. If the travel time is less then 0.5 min the weight value will be set to 9999. PT travel time shorter that half a minute are ignored.

If **No car with license** travel time is less than 7 min the skim weight value will be set to 99999. Otherwise the total travel time will be calculated just as for **car**. Next the total travel costs will be determined by the variable costs-no-car (per soort-geen-auto from the config) added to (km-charge multiplied with the distance) this is then added by the multiplication of car-travel-time with time-costs-no-car (per soort-geen-auto from the config)

Next the total costs is multiplied with the multiplication factor per incomeclass added with the totale *time*.

Free car per income-class: First the total time is calculated by adding the time searching for parking (before and after the trip) with the time the trip takes. Total time is added to the multiplication of distance times factor from config file times `€/km`.

FreePT when the time value is below 0.5 minutes the matrix value is set to 9999 otherwise the value from the PTtimematrix is directly written to the experienced-travel-time-matrix for FreePT.

The values in the experienced-travel-time-matrices are in `minute` from zone to zone.

Verdelingovergroepen

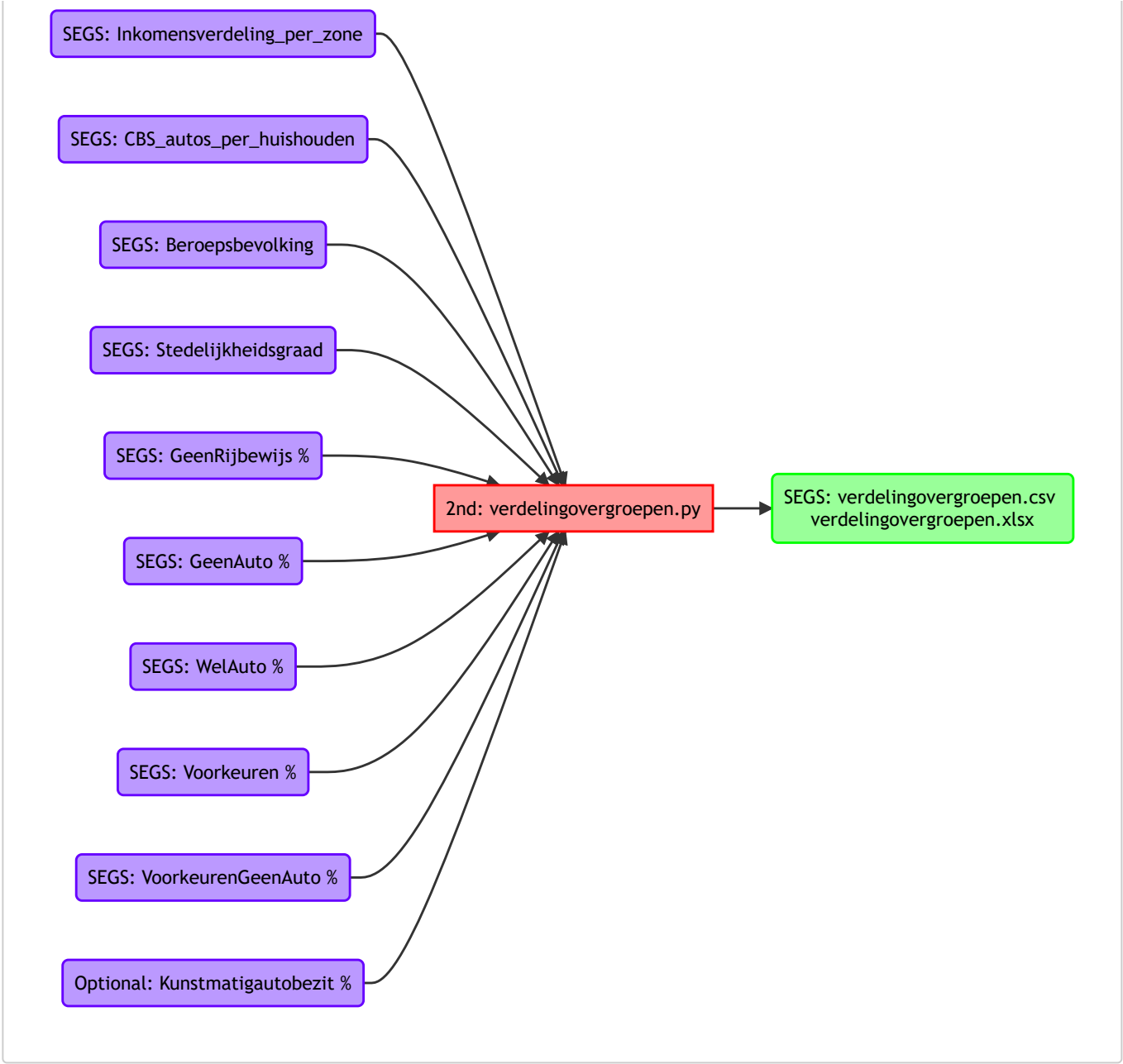
This script is the 2nd one to run in correspondence with the option `Verdeling van de groepen over de buurten of zones` in the wizzard/ui. The resulting file is written to the `SEGS` folder.

There is a big correlation between de CBS data for education level and income level.

There are constants hardcoded in the script for the percentage of Free cars per income-class:

Electrification%	low income	mid-low income	mid-high income	high income
40	0	0.17	0.58	0.95
20	0	0.05	0.25	0.5
-	0	0.02	0.175	0.275

This script can use the `kunstmatig autobezit`(artificial car-ownership) file (desired amount of cars per 10 households). This is useful for cases where you want to see what the effect of (for example) a strict parking policy are. This overrides the CBS data of car ownership per household.



Income distribution per zone is divided by 100 (percentage to fraction) times WelAuto/100 makes a new fraction: `fraction car ownership`

This is checked against `minimum car ownership` (which is either data from CBS or `artificial car ownership`). If `minimum car ownership` is less then the sum of `percentage car ownership` (sum for the various income classes) then a `car ownership correction` is determined by dividing the `minimum car ownership` with `percentage car ownership`. Otherwise the correction is set to 1 and it won't be used to determine the `NO car ownership correction`.

Next the amount of `car ownership` is determined by multiplying the theoretical amount of car ownership from the WelAuto(YesCar) file with the `car ownership correction`. If `car ownership correction` is below 1 you can determine the `NO car ownership correction` fraction by (1 - YesCarPortion)/ (1-YesCarPortionTheory)
This correction factor, for not owning a car, is then multiplied with NoCar en NoLicence and appended to NoCarYesLicence, NoLicence respectively.

The income distribution is divided by 100 to calculate with fractions. `Free Car` is (fraction YesCar) times (fraction FreeCar) per income-class and urbanizationclass. `No Free Car` is (fraction YesCar) MINUS (fraction

`Free Car`) times (1-GratisOVpercentage) times income distribution.

Per incomeclass and urbanizationclass the fraction 'free car', 'no free car' and 'free PT' is determined in order to determine the preferences for the other combinations next. For 'free PT' and 'no free car' it is logical that there are no other preference combinations.

`FreePT` times income distribution times FreePTpercentage (which is a fraction) Preference YesCar but FreePT: NoFreeCar times income distribution times FreePTpercentage.

Now per preference:
Per incomeclass you calculate the chance that you have a preference to use Car, Neutral, PT or Bike if you don't have a free car of free public transport.

Next the chance that one has a preference with free PT, does not own a car but has a license. Then the preference-chance if one had free PT and not license.

Finally the chances of the left over combinations are calculated.

This all results in an Exel with `fractions` per incomeclass per preference for a mode of transport.

Gewichtenberekenenenkelscenarios

This is the third script. Which generates weights based on travel-time-decay-curves for car, PT, Bike and E-Bike separately. The script writes files to the folder `Gewichten`(weights) and then creates a subfolder with the part of the day you selected (ds). It reads all files from the `Ervarenreistijd` folder generated in script 1: skimsberekenen.py

This script contains constants per `mod`(mode of transport) and `preference`.
Preference: car, neutral, bike and PT.
Motive: work, dailyshopping, not-dailyshopping, other
Mode: Car, PT, Bike, EBike, NoCar, NoLicence, FreeCar, FreePT.
Income: high, middel-high, middel-low, low

[add chart]

There formulas and variables are determines based on OVIN data. Do note that OVIN does not include `Experienced travel time` or `costs`

Defaults are:

Preference	mode	alpha	omega	weight
-	-	0.125	45	1
-	Bike	0.225	25	1
-	EBike	0.175	35	1
Preference	**mode**	**alpha**	**omega**	**weight**
Car	Car	0.125	50	1
Car	PT	0.125	30	0.95
PT	Car	0.125	45	0.96

Preference	mode	alpha	omega	weight
PT	PT	0.12	60	1
Bike	Car	0.225	25	0.75
Bike	Bike	0.175	35	1
Bike	EBike	0.125	55	1

#The travel-time-decay-curves are hereby hardcoded into the script. It is easy to find and potentially change with new insights/whishes.
#The motives are constants which are hardcoded in "Constanengenerator.py".
#Very interesting for a sensitivity analyses.

Work

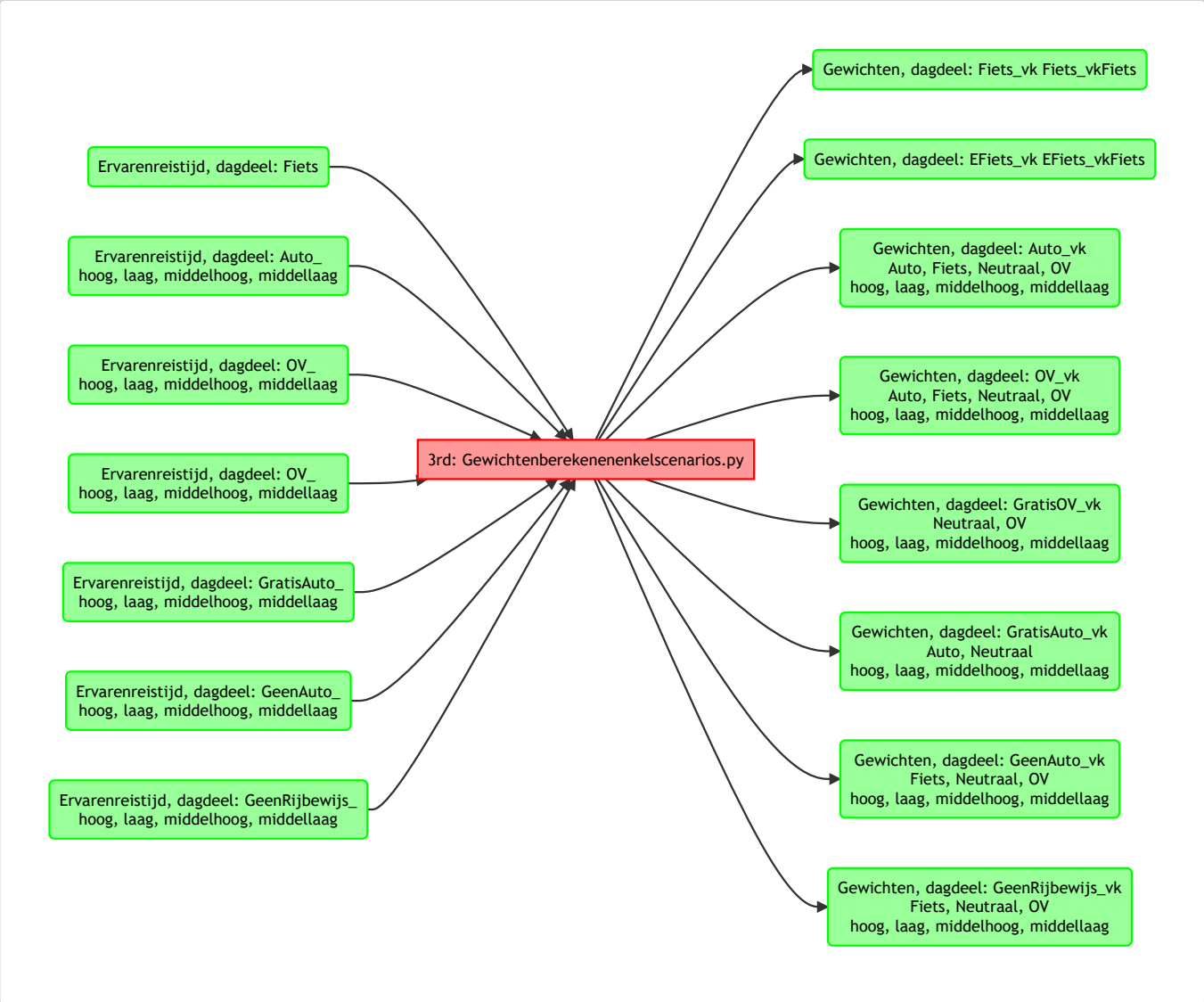
Preference	mode	alpha	omega	weight
default	-	0.125	45	1
default	Bike	0.175	25	1
default	EBike	0.175	35	1
Car	Car	0.125	50	1
Car	PT	0.125	30	0.95
PT	Car	0.125	45	0.96
PT	PT	0.12	60	1
Bike	Car	0.125	45	0.75
Bike	Bike	0.175	35	1
Bike	EBike	0.125	55	1

Daily shopping

Preference	mode	alpha	omega	weight
default	-	0.225	12.5	1
default	Bike	0.225	10	1
default	EBike	0.225	10	1
Bike	Bike	0.225	12.5	1
Bike	EBike	0.225	12.5	1
Bike	Car	0.225	12.5	0.75
PT	PT	0.175	12.5	1

Not daily shopping

Preference	mode	alpha	omega	weight
default	-	0.225	20	1
default	Bike	0.225	15	1
default	EBike	0.225	15	1
Bike	Bike	0.225	20	1
Bike	EBike	0.225	20	1
Bike	Car	0.225	20	0.75
PT	PT	0.175	20	1



A `weight matrix` is made based on `experienced travel time`. If the `experienced travel time` is less than 180 min the travel time value is calculated with the corresponding decay curve. Otherwise the value is set to 0. If the value after the decay curve calculation is smaller than 0.001 the travel time is set to 0.

For E-bike and Bike per part of day with preferences car or bike with the motive work and daily shopping the constants from the tables above are used in the decay curve calculations. for the other combinations the default values are used.

For Car and PT per motive the corresponding constants are used to calculate the weight value per incomeclass.

For FreeCar the Car constants are used and the preferences Neutral and Car combinations are calculated.

For FreePT the PT constants are used and the neutral and PT preferences are calculated.

This results is a lot of csv's with travel time in minuts.

Gewichtenberekenencombis

The maximum weight between multiple modes of transport is determines in this 4th script.
This scrips uses "Routines.py" to write files into certain folders.
Its reads the data from `Ervarenreistijd` en `Gewichten, dagdeel`.
Writes to `Gewichten, Combinaties, dagdeel`.

If you don't have a car or drivers license then you don't have a preference for car but you do have a preference for (Free)PT
If you don't have a car or license and you do have FreePT then your preference is PT (the other modes of transport will be excluded in the calculations).
If you have FreeCar and FreePT your preferense becomes Neutral.
FreeCar but normal PT: preference car.
FreePT: preference PT

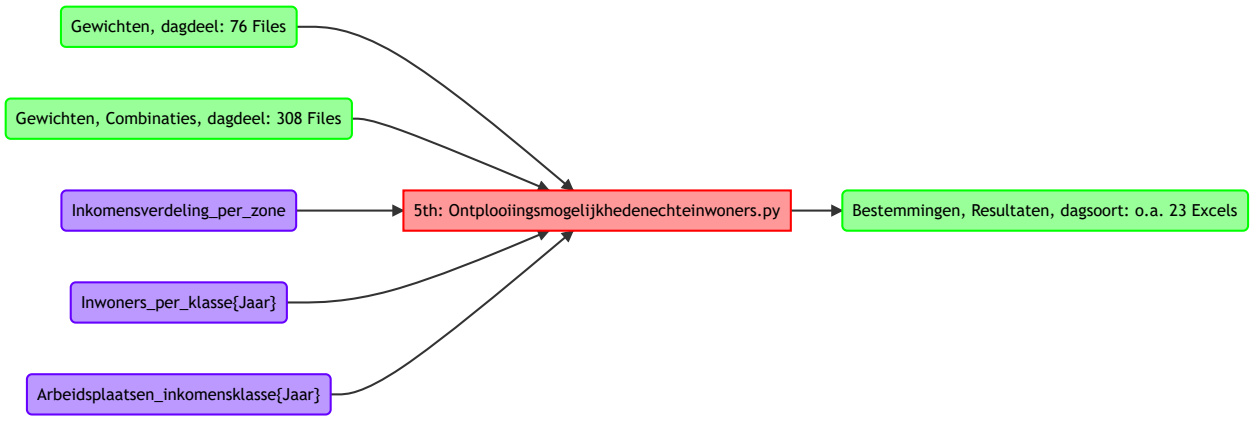
The script determines the maximum values of the following matrix combinations (per incomeclass per preference):

- Bike, EBike VS PT, FreePT
- Bike, EBike VS Car, NoCar, NoLicence, FreeCar
- PT, FreePT VS Caro, NoCar, NoLicence, FreeCar
- Car, NoCar, NoLicence, FreeCar VS Bike, EBike VS PT, FreePT

This results in many combination possibilities and csv's with in it the maximum travel time in minutes of these combinations.

Ontplooiingsmogelijkhedenechteinwoners

This script approaches the accessibility for jobs for inhabitants.
Input from the 'SEG' directory: Inkomensverdeling_per_zone, Inwoners_per_klasse{Year}, Arbeidsplaatsen_inkomensklasse{Year}
Loads everything from `Gewichten: combinaties, dagsoort` and `Gewichten, dagsoort`.
Writes to `Bestemmingen, resultaten, dagsoort`. The xslx's files here will have readable headers (the csv's are intermediate files. Note that these intermediate files will be used again in script 7).



In the Netherlands there is data for what kind of industry needs what percentage of education level and level of salary. This can be combined with LISA data to determine the percentages of different kinds of jobs per zone.

The potential jobs are calculated for a 'mode of transport matrix' with the help of the number of job-places in a zone, the distribution matrix and income distribution per incomeclass.

If the `population share` (which is a percentage) is bigger than 0 then the sum of the weighted matrix is divided by the population-share. Otherwise the value is set to 0.

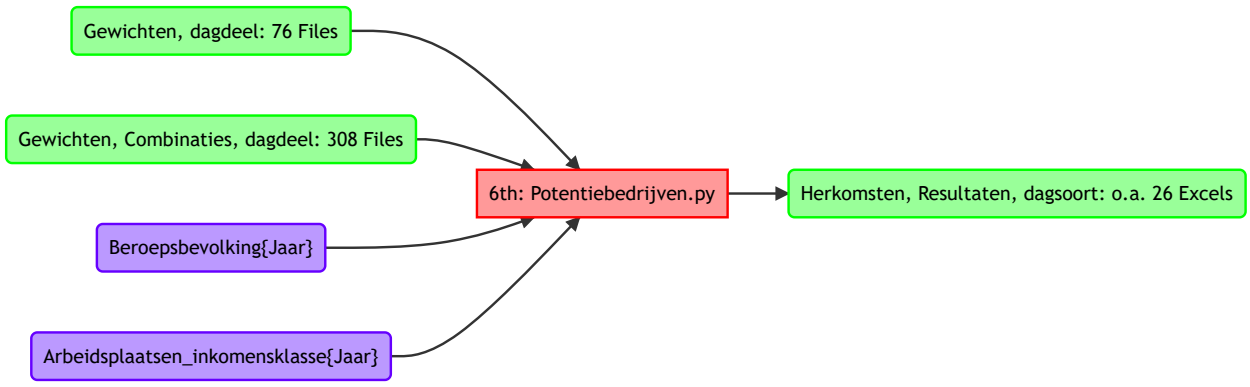
This results in 23 Excels:

- per mode of transport
- per combination of 2 or 3 modes of transport
- per incomeclass

Depending on that you are interested in you can chose the desired excel.
This like in the previous script there are a lot of intermediate csv files. For the Excels with a mode of transport in the title the rows represent the zones and the columns the incomeclasses.
For the Excels with incomeclasses in the title the rows represent the zones and the column represent the various modes of transport and possible combinations.
The values represent `attractiveness`.
When you (with for example a high income and a preference for biking) live in a zone with a high number, it is more attractive to live there than in a zone with a lower number.

Potentiebedrijven

This script approaches the accessibility of a labor force for employers/institutions.
Input from 'SEG' directory: Arbeidsplaatsen_inkomensklasse{Year} Beroepsbevolking{Year}
Reads everything from `Gewichten: combinaties, dagsoort` and `Gewichten, dagsoort`.
Writes to `Herkomsten, resultaten, dagsoort`. The xslx's files here will have readable headers (the csv's are intermediate files. Note that these intermediate files will be used again in script 8).



First a inhabitant matrix is made by multiplying the labor force (per zone) with the group distribution from the config file.

Potentials are calculated by summation of the Weight matrix which was already multiplied by the inhabitant matrix groups.
`minutes * inhabitants * distribution`
Does not take into account what kind of jobs you are trying to reach? No cause you are trying to reach kinds of workers.

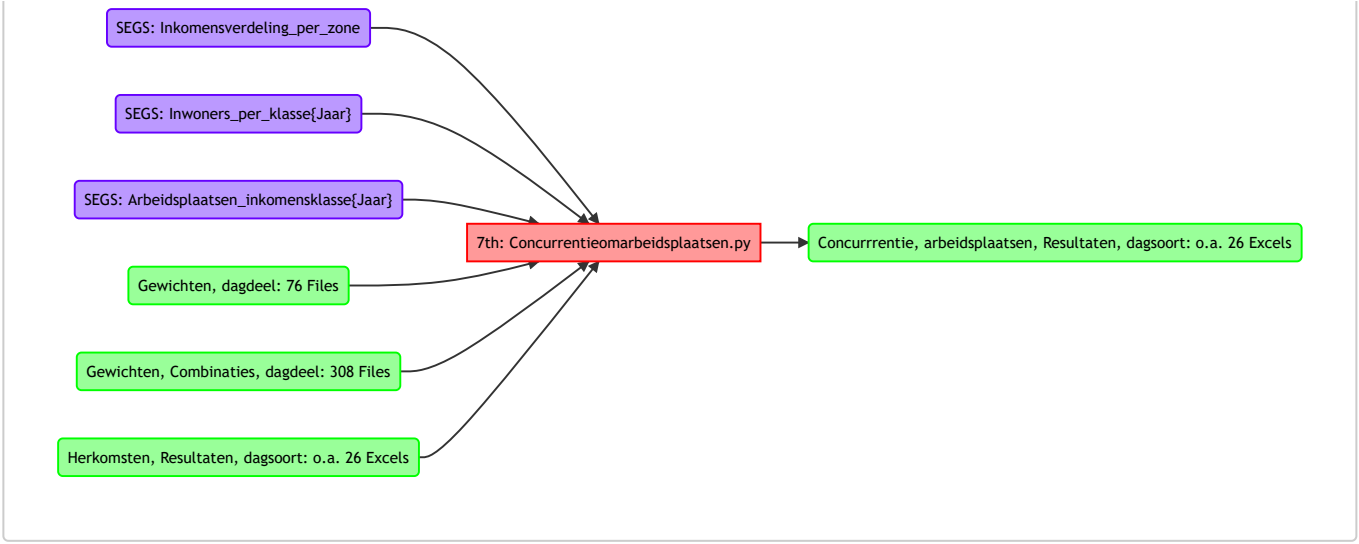
This results in 23 Excels:

- per mode of transport
- per combination of 2 or 3 modes of transport
- per incomeclass

Depending on that you are interested in you can chose the desired excel.
This like in the previous script there are a lot of intermediate csv files. For the Excels with a mode of transport in the title the rows represent the zones and the columns the incomeclasses.
For the Excels with incomeclasses in the title the rows represent the zones and the column represent the various modes of transport and possible combinations.
The values represent `attractiveness`.
The values are interpretable as an `attractiveness` for a company to settle somewhere where they are easily reachable to a certain group of employees compared to other zones.

Concurrentieomarbeidsplaatsen

This calculated the competitive position of an inhabitant to reach jobs.

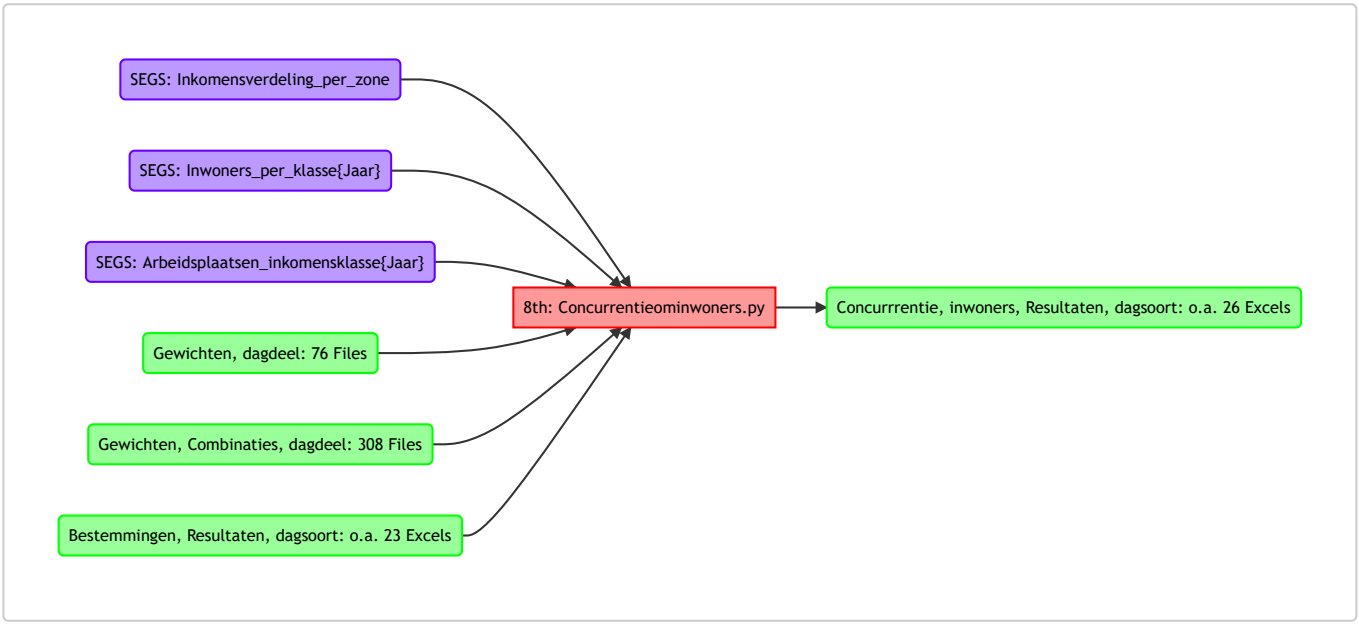


This script multiplies the `Mode of transport matrix` with `jobs` (per incomegroup) divided by `reach` (herkomst). This is a literal translation and I don't think it is correct. This chapter needs formulas and more explanation.

A high value in a zone means that there are a lot of inhabitants able to travel to that zone for (for example) a job. So a lot of competition for example jobs, schools, shops or dentists.

Concurrentieominwoners

This calculated the competitive position of employers to be reached by inhabitant.



Per zone the `travel time weights` are multiplied with `inhabitants` divided by `Destinations` (aantrekkelijkheid). This is a literal translation and I don't think it is correct. This chapter needs formulas and more explanation.

If, as a company, you are located in a zone with a high values it will be easier for you to attract employees from bigger distances.
With a small value you can attract employees close by but the pool is smaller and you are in competition with other companies who are also trying to attract people.

Benchtest

Test is done with 2015 socio-economic data from Tilburg distributed over 1425 zones.
On a laptop reading a writing from SSD. 16GB RAM, Processor i7 2.6GHz.

Script	start-time	end-time	runtime	notes
1 skimsberekenen	12:22	12:23	1 min	-
2 verdelingovergroepen	-	-	a few sec	-
3 Gewichtenberekenenenkelscenarios	12:23	12:26	3 min	-
4 Gewichtenberekenencombis	12:26	12:36	10 min	no update messages
5 Ontplooiingsmogelijkhedenechteinwoners	12:38	12:55	17 min	-
6 Potentiebedrijven	13:44	13:53	9 min	-
7 Concurrentieomarbeidsplaatsen	15:28	15:37	9 min	-
8 Concurrentieominwoners	15:38	15:47	9 min	-

feedback/ideas

- remove temp files after using them
- don't write to folders that are used for input: write to a 'scenario' output folder
- naming of processes (in scrips) output files and folders, wizard items, and script names could be more consistent. And in English
- https://choosealicense.com/licenses/gpl-3.0/
- Karin: add fomula's to documentation.
- expand "concurentie" chapters (with examples and forumlas etc)
- LISA data is closed. There is work being done to get the aggregated (to zones) subset of the data needed for this model (because we don't need everything from LISA) and Sander vd Drift is working on a possible estimation.
- Where to put the 'zakelijk auto gebruik' bit of text: The percentage of business car use can be estimated using the incomegroupes and location of where these people live.
- input file: OV_Kosten (public transport costs in eurocents) ask if correct