Spatial Microsimulation Urban Metabolism Documentation

Release 0.2.1

Dr. Esteban Muñoz

Jun 12, 2018

Contents

Author Dr. M. Esteban Munoz H. <emunozh@gmail.com> or <esteban.munoz@un.org>

Version 0.2.0

Date 2018-06-11

Note: This documentation is generated automatically from the main github repository. The build of the documentation is not always successful and you might end up reading a documentation of an outdated version of the model. Please always verify the version of the model you are running. This simulation library is under development and we are constantly changing the simulation libraries.

This is the main documentation for the Spatial Microsimulation Urban Metabolism model. This model combines two powerful approaches for the simulation of resource flows within cities. The first approach is Urban Metabolism (UM). This approach describes the metabolic performance of cities by quantifying and balancing all resource inputs and outputs from a predefined city-system. The second component of the simulation model is the Spatial Microsimulation (SM) model. This component of the simulation library constructs a synthetic population for the specific city-system and allocates consumption values to the individual city agents. The simulation library benchmarks this synthetic sample to the aggregated consumption values from the UM model.

The aim of this documentation is twofold:

1. Describe the methodological approach of the simulation model; and
2. Explain how to use the components of the library and present some simulation examples.

This simulation library is build on top of some well know python libraries as well as some specific python an R libraries.

Python libraries	Use
pandas	Data library for python
numpy	Numerical model in python
scipy	Scientific python
statsmodels	Statistical models
Theano	Compiled numerical computation library.
jupyterhub	Used as main UI
matplotlib	De facto python plotting library
seaborn	Statistical plots
pymc3	Bayesian Statistics, MCMC
ipfn	Iterative Proportional Fitting
XlsxWriter	Create excel files

R libraries	Use
GREGWT	Sample reweighting library

CHAPTER 1

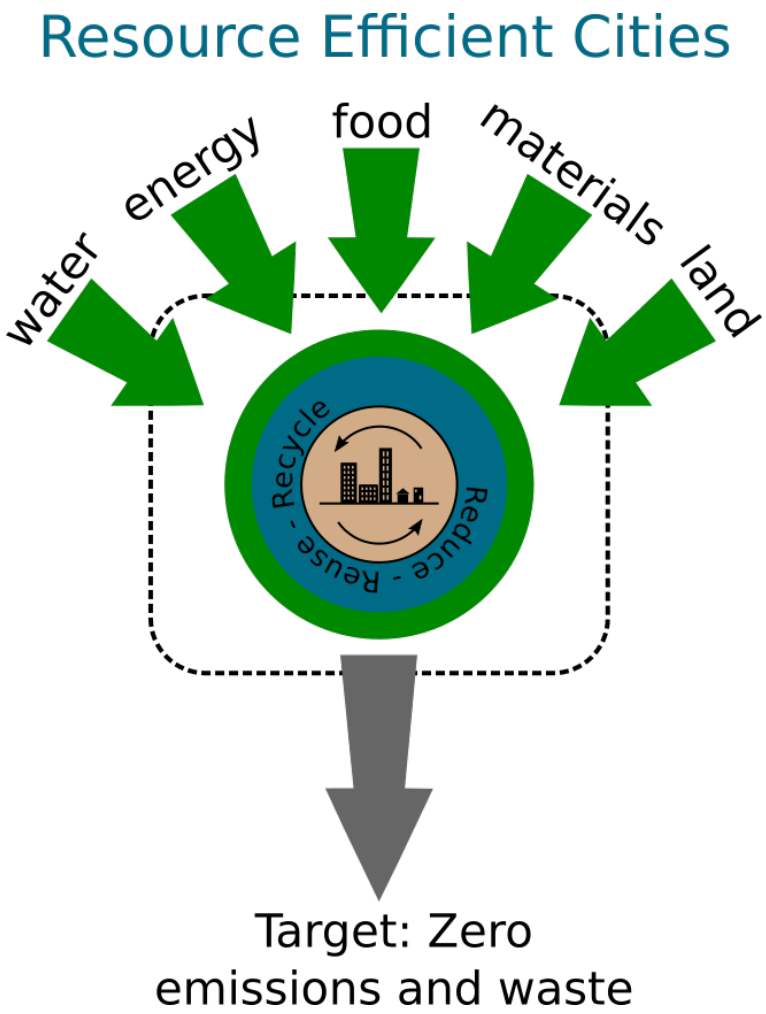
Introduction

In order to understand the flow of resources occurring within a city-system we represent all the inputs and outputs from these city-system. For the computation of these inputs and outputs the library makes use of the urban metabolism approach.

The quantification of resources flow at an aggregate level is not enough for cities to take knowledge-based decisions on future infrastructure investment and policies targeting a sustainable urban development. In order to understand the impact of city level policies and investment strategies cities need to understand:

1. The drivers of consumption and
2. The plausible impact of these policies on their citizens.

In order to get this level of understanding we propose the simulation of consumption intensities at a micro-level. By describing the consumption intensities at this level of detail (and implicitly their consumption drivers) cities have a tool to assess the impact at this micro-level (e. g. at municipal or district level). The microsimulation module of this library constructs a micro-level synthetic sample with demographic-variables (drivers) and consumption values (benchmarked to aggregated values from the Urban Metabolism (UM) model).



This documentations aims to describe the main rationale behind the development of this python library and to present an overview of the main modules and functions implemented on the library.

The python library is composed of two main components:

1. An Urban Metabolism section, see *Top-Down: City-Systems (Urban Metabolism)* that aims to balance all resource flows of city systems at an aggregated level (i.e. city-level) and;
2. A Spatial Microsimulation section, see *Bottom-Up: Synthetic Populations (Spatial Microsimulation)*. which constructs a synthetic city and allocates consumption values to micro-level agents

For a complete implementation example of the Spatial Microsimulation Urban Metabolism Model (SMUM), please refer to the following link: ipython notebook

This set documentations provides the documentation of the individual library modules and all the functions within these modules as well as some extended examples on how to use the provided functions.

The main library function documentation can be found at:

- *API: Top-Down (Urban Metabolism)* for the Urban Metabolism functions and at *Top-Down: City-Systems (Urban Metabolism)* for a general description of the UM module; and
- *API: Bottom-Up (Spatial Microsimulation)* for the Spatial Microsimulation functions and at *Bottom-Up: Synthetic Populations (Spatial Microsimulation)* for a general description of the SM module.

A complete list of examples can be found at:

- *Examples*

A complete list of the library authors and contributors can be found at:

- *Authors*; and
- *Contributing*

CHAPTER 2

Top-Down: City-Systems (Urban Metabolism)

The Urban Metabolism (UM) module aims to describe the resource flows of a city-system at an aggregated level with the use of input-output tables.

This module aims to provide:

1. A framework for the description of resource flows;
2. A description of macro-level drivers for the changes of these flows; and
3. A description of linkages between different resource flows.

The advantgage of this library module–and of this python library– is that each module can be used independently. The UM module can be run independently from the rest of the library.

The library is structured as two types of functions:

1. A function dedicated to the description of a city and city-data, see city.
2. Resource flow specific classes:
    - *Materials* & api: *Materials*
    - *Water* & api: *Water*
    - *Consumption model: Energy* & api: *Energy*
    - *Food* & api: *Food*
    - *Waste* & api: *Waste*

For each of these classes a Unified Modeling Language (UML) class diagram has been generated.

The description of the individual functions of this module can be found below under: *API: Top-Down (Urban Metabolism)*.

2.1 Consumption model: Energy

2.1.1 Energy Class: Energy Flow

"Energy flow is the ultimate of the urban metabolism. The magnitude of energy flows from heating and cooling are typically related to climate, but other components of urban energy use can be linked back to the shape and form of a city, reflected by its infrastructure systems and hence material stocks. (Kennedy.2012)"

$$I_E = I_{E,buildings} + I_{E,transport} + I_{E,industry} + I_{E,construction} + I_{E,waterpumping} + I_{E,waste}$$

The total energy demand of a city is expressed as the total sum of all energy demands for the different energy sectors. Each energy sector energy demand is calculated, based on different components that are the cause of energy consumed.

Buildings: For the building sector the total energy consumption is computed based on climatic conditions of the city (HDD and CDD) and energy consumption intensities based on building type.

$$I_{E,buildings} = I_{E,heating} + I_{E,cooling} + I_{E,light-and-appl.} + I_{E,water-heating}$$

$$I_{E,heating} = \sum_{building-type} HDD * i_{E,heating} * P * f$$

$$I_{E,cooling} = \sum_{building-type} CDD * i_{E,cooling} * P * f * cp$$

Where:

- HDD Heating degree days.
- CDD Cooling degree days.
- $i_{E,cooling}$ heating intensity.
- $i_{E,heating}$ cooling intensity.
- P Population of the urban agglomeration.
- f Floor space area per capita.
- cp

Transport: The total energy demand for the transport sector is computed based on the different types of transportation observed within the analyzed system:

$$I_{E,transport} = I_{E,passenger} + I_{E,freight} + I_{E,aviation} + I_{E,marine}$$

When anaylzing a city the first xx (Summand) surface passenger transport will be the most relevant. For this transportation category the energy demand is calculated based on the different types of passenger transport found in the city:

$$I_{E,passenger} = \sum_{mode} \frac{1}{P_p} * P * \rho_i * h * \varepsilon$$

Where:

- $mode$
- P_p Average population density $[km^{-2}]$.
- ρ_i Density of transportation infrastructure $[km * km^{-2}]$.
- h utilization intensity of infrastructure [veh-$km * km^{-2}$].

- ε Fuel efficiency $[J * \text{veh-}km^{-1}]$.

The product of the first four terms within the summation is equivalent to the vehicle-kilometers traveled (VKT):

$$VKT = \frac{1}{P_p} * P * \rho_i * h$$

A widely used city indicator when analyzing sustainability in urban environments.

Energy surface balance (NOT IMPLEMENTED):

$$I_{E,S} + I_{E,F} + I_{E,I} = O_{E,L} + O_{E,G} + O_{E,E}$$

Where:

- $I_{E,S}$ Rate of arrival of radiant energy from the sun.
- $I_{E,F}$ Rate of generation of heat due to combustion and dissipation in machinery.
- $I_{E,I}$ Rate of heat arrival from the earth's interior.
- $O_{E,L}$ Rate of loss of heat by evapotranspiration.
- $O_{E,G}$ Rate of loss of heat by conduction to soil, buildings, roads, etc.
- $O_{E,E}$ Rate of loss of heat by radiation.

2.1.2 Energy class: Stock

This class defines the existing energy stock by sector.

All energy streams are aggregated by sector.

A data-set with the detail energy stream is generated as a *csv* file and stored under the */results* folder.

The Energy Stock is computed as follows:

2.2 Water

2.2.1 Water class: Water Demand

Similar to Energy Flow, Water Demand is computed as the sum of different water consumers. In a city most water is consumed at the building level. Therefore total Water Demand (Q_W) is determined based on residential and non-residential water demand.

$$Q_W = Q_{W,D}^{hh} + Q_{W,D}^{nr}$$

Where:

- $Q_{W,D}^{hh}$ Household water consumption.
- $Q_{W,D}^{nr}$ Non-Residential water consumption.

** Residential Water Demand:** The residential water demand or household demand model is computed as function of the following indicators:

- Demographic characteristics of the household.
- Disposable income of the household.
- Average water price in the city.

- Water saving penetration rate (SP) Yuan, X.-C. et al. (2014).
- Water saving rate (SR) Yuan, X.-C. et al. (2014).

$$Q^{hh}_{W,D} = \beta_0 + \sum_{i}^{n} \beta_i HH_i + \beta_y Y_{hh,}$$

+ $\beta_p P$

Where:

- $Q^{hh}_{W,D}$ Household water consumption.
- HH Household characteristic.
- $Y_{hh,}$ Household income.
- $P_{W,}$ Water price.
- β_i
- ϵ_{err} Random error term.

Depending on the water tariff in place the variable $P_{W,}$ cannot be modeled as a dependent variable. If the water tariff is computed as a function of consumed volume, the error term cannot be assumed.

The Household characteristics (HH) are based on data availability and the definitions made within the water consumption.

Efficiency rate:

The water saving penetration (SP) and water saving rate (SR) are computed at each simulation step. The water saving rate is an indicator for governmental actions to reduce water consumption. And the penetration rate is the likelihood that a household has adopted the respective the water saving behaviour or technology.

$$Q^{base}_{W,D}(SP_{W,D}, SR_{W,D}) = \begin{cases} Q^{hh}_{W,D} \times (1 - SR_{W,D}) & \text{if } rand < SP_{W,D} \\ Q^{hh}_{W,D} & \text{else} \end{cases}$$

Where:

- $Q^{base}_{W,D}$ Base water consumption.
- $SP_{W,D}$ Water saving penetration rate.
- $SR_{W,D}$ Water saving rate.

Non-residential Water Demand: The non-residential water demand model is defined as the sum of (source: DGNB):

- Water consumption by buildings occupants. Q^{nr}_{DU}
- Water consumption for cleaning. Q^{nr}_{DC}
- Water consumption by spa facilities. Q^{nr}_{DS}
- Water consumption by laundering facilities. Q^{nr}_{DL} (not implemented)

$$Q^{nr}_{W,D} = Q^{nr}_{W,DU} + Q^{nr}_{W,DC} + Q^{nr}_{W,DS} + Q^{nr}_{W,DL}$$

Where:

$$Q^{nr}_{W,DU} = \sum_{i=1}^{n} wb_I$$

$$wb_I = (n_{NU} \times f_I \times as_I \times d/a) \, /1000$$

Where:

- n_{NU} Number of users/occupants/employees/visitors/customers
- f_I Installation factor of equipment (see *Tab. W1*) $[s/d]$
- as_I Equipment water demand factor (see *Tab. W2*) $[l/u]$
- d Occupancy rate in days

Table 2.1: Tab. W1. Installed equipment factors f_I

Equipment	Office	Hospital (number of beds (number of beds n_e)			Commerce		Hotel (single n_{ez}, double n_{dz})	Residential
	Employee	Employee	Patient	Visitor	Employee	Customer	Customer	Occupant
n_{NU}			$0.8 * n_e$	$0.5*0.8* n_e$			$(n_{ez} + (n_{DZ} * 1.2)) * 0.65$	
Toilet sink	75	45	135	15	45	15	75	195
WC-Saving	4	1	2	0.5	1	0.3	1	4
WC	1	1	1	0.5	1	0.5	1	1
Urinal	4	1		0.5	1	0.2	1	
Shower	30	60	90		30			120
Kitchen sink	20	20			20			
Sink-Spa							15	
WC-Saving-Spa							1	
Shower-Spa							600	
Dishwasher								0.5
Washing machine								0.25

Table 2.2: Tab. W2. Water demand factors

Equipment	Office	Hospital	Commerce	Hotel	Residential
Toilet sink $[l/s]$	0.15	0.15	0.15	0.15	0.15
WC-Saving $[l/u]$	4.5	4.5	4.5	4.5	4.5
WC $[l/u]$	9	9	9	9	9
Urinal $[l/u]$	3	3			
Shower $[l/s]$	0.25	0.25	0.25	0.25	0.25
Bathtub $[l/u]$					Capacity
Kitchen sink $[l/s]$		0.25	0.25		
Dishwasher $[l/u]$					20
Washing machine $[l/u]$					60

$$Q^{nr}_{W,DC} = \sum_{i=1}^{n} \left(A_{R,i} \times wb_{R/A} \right) /1000$$

$$Q^{nr}_{W,DS} = \sum_{i=1}^{n} wb_I$$

$$wb_I = (n_{SPA} \times f_I \times as_I \times 360d/a) /1000$$

$$n_{SPA} = n_{NU} \times 0.25$$

$$Q^{nr}_{W,DL} = \sum_{i=1}^{n} wb_I$$

Where:

- A_R Cleaning floor space $[m^3/a]$
- wb_R Water demand per cleaning area (see *Tab. W3*) $[l/(m^2 \times a)]$
- wb_I Specific water demand of spa/laundry installations (see *Tab. W1* and *Tab. W2*) $[m^3/a]$

Table 2.3: Tab. W3. Water demand per cleaning area. wb_R in $[l/m^2a]$

Type of area	Frequency	Office	Hospital	Commerce	Hotel	Residential
Floor	1 x Month	1.50	1.50	1.50	1.50	1.50
	1 x Week	6.25	6.25	6.25	6.25	6.25
	3 x Week	18.75	18.75	18.75		18.75
	4.5 x Week				28.125	
	5 x Week		31.25			
	6 x Week		37.50	37.50		
	7 x Week		43.75		43.75	
Glass surface	2 x Year	0.60				0.60
	4 x Year	1.20	1.20	1.20	1.20	1,20
	6 x Year	1.80				1.80
	12 x Year		3.60	3.60	3.60	
	24 x Year			7.20	7.20	

2.2.2 Flow

This water flow is balanced as follows:

$$I_{W,percip} + I_{W,pipe} + I_{W,sw} + I_{W,gw} = O_{W,evap} + O_{W,out} + \Delta S_w$$

Where:

- $I_{W,percip}$ Is natural inflow from precipitation.
- $I_{W,pipe}$ Is water piped into the city.
- $I_{W,sw}$ Is the net surface water flow into the city.
- $I_{W,gw}$ Is the net ground water flow into city aquifers.
- $O_{W,evap}$ Evapotranspiration.
- $O_{W,out}$ Water piped out of cities
- ΔS_w Change in water storage of urban agglomeration.

Anthropogenic Water Use:

The anthropogenic water consumption is computed as follows:

$$Q_W = Q_{W,D} + Q_{W,L}$$

Where:

- $Q_{W,D}$ Water demand.
- $Q_{W,L}$ Water losses.

$$Q_{W,D} = \sum_{hh} Q_{W,D,hh}^{base} + CDD * i_W^{cooling}$$

Where:

- $Q_{W,D}^{base}$ Base water consumption.
- CDD Cooling Degree Days.
- $i_W^{cooling}$ Intensity of water use for cooling.

$$Q_{W,L} + A * p_{ti} * l$$

Where:

- $Q_{W,L}$ Water losses.
- A Surface area of urban agglomeration.
- p^{ti} Density of urban infrastructure.
- l Annual leakage rate per length of linear infrastructure.

$$Q_{WWT} = Q_{WWE} + Q_{WWF} + Q_{INF}$$

Where:

- Q_{WWT} Treated waste water.
- Q_{WWE} Generated waste water.
- Q_{WWF} Wet weather water flow.
- Q_{INF} Base infiltration.

Urban Aquifers:

$$\Delta S_{W,gw} = \Delta Q_{W,RO} + Q_{W,ar} + \Delta I_{W,gw} - \Delta Q_{W,DO} - Q_{W,gwpump}$$

Where:

- $\Delta S_{W,gw}$ Change in ground water storage of urban agglomeration.
- $\Delta Q_{W,RO}$ Change in natural recharge from virgin conditions.
- $Q_{W,ar}$ Net anthropogenic urban water recharge rate.
- $\Delta I_{W,gw}$ Net change on ground-water inflow.
- $\Delta Q_{W,DO}$ Change in natural discharge from virgin conditions.
- $Q_{W,gwpump}$ Net pump rate of urban agglomeration.

Internal Renewable Water Resources (IRWR)

$$IRWR = S_{W,sw} + S_{W,gw} - S_{W,overlap}$$

External Renewable Water Resources (ERWR)

$$ERWR = I_{W,sw} - O_{W,sw} + I_{W,gw} - O_{W,gw}$$

Total Renewable Water Resources (TRWR)

$$TRWR = (S_{W,sw} + I_{W,sw} - O_{W,sw}) + (S_{W,gw} + I_{W,gw} - O_{W,gw}) - S_{W,overlap}$$

Where:

- $S_{W,sw}$ Surface water, produced internally.
- $S_{W,gw}$ Groudwater, produced internally.
- $S_{W,overlap}$ Overlap between surface water and groundwater.

2.2.3 Stock

2.3 Materials

2.3.1 Flow

2.3.2 Stock

All material streams are aggregated by sector.

A data-set will the detail material stream is generated as a *csv* file and stored under the */results* folder.

The Material Stock is computed as follows:

$$S_M = \sum_s \sum_m S^s_{M,m}$$

The total materials stock of a city is expressed as the total sum of all type of materials m of all urban structures s.

$$S^{rb}_{M,m} = P * f^{rb} * i^{rb}_{M,m}$$

Where:

- $S^{rb}_{M,m}$ Material stock of residential buildings.
- P Population of the urban agglomeration.
- f^{rb} Per-capita floor space for residential buildings.
- $i^{rb}_{M,m}$ Material intensity per squared meter.

$$S^{ti}_{M,m} = A * p^{ti} * i^{ti}_{M,m}$$

Where:

- $S^{ti}_{M,m}$ Material amount in linear transportation infrastructure.
- A Surface area of urban agglomeration.
- p^{ti} Density of urban infrastructure.
- $i^{ti}_{M,m}$ Material intensity per kilometer of urban infrastructure.

2.4 Waste

2.4.1 Flow

2.4.2 Stock

2.5 Food

2.5.1 Demand

** Residential Food Demand:** The residential food demand or household demand model is computed as function of the following indicators:

- Demographic characteristics of the household.
- Disposable income of the household.

$$Q_{F,D}^{hh} = \beta_0 + \sum_{i}^{n} \beta_i HH_i + \beta_y Y_{hh,}$$

+ ϵ

Where:

- $Q_{F,D}^{hh}$ Household food consumption.
- HH Household characteristic.
- $Y_{hh,}$ Household income.
- β_i
- ϵ_{err} Random error term.

The Household characteristics (HH) are based on data availability and the definitions made within the food consumption.

2.5.2 Flow

$$I_F + P_F + I_{W,Kit} = O_{F,RetFW} + O_{F,ResFW} + O_{F,Met} + O_{F,S}$$

Where:

- I_F mass of food and packaged drinks imported to the city.
- P_F mass of food and packaged drinks produced in the city, for internal consumption.
- $I_{W,Kit}$ mass of kitchen water used during food preparation or drunk during meals.
- $O_{F,RetFW}$ mass of retail food waste produced by grocery stores and restaurants.
- $O_{F,ResFW}$ mass of residential food waste going to landfill, compost, or organic waste collection.
- $O_{F,Met}$ mass of carbon and water lost via respiration and transpiration in residents metabolism.
- $O_{F,S}$ mass of feces and urine exported to sewerage system.

2.5.3 Stock

CHAPTER 3

Bottom-Up: Synthetic Populations (Spatial Microsimulation)

The presented simulation framework on this report implements a simple model for the description of urban resource flows and their projection into the future under predefined scenarios.

The description of the individual functions of this module can be found below under: *API: Bottom-Up (Spatial Microsimulation)*.

The simulation framework is constructed as a hybrid model. The Model balance input-output tables at an aggregated level, its contra-part module computes consumption levels at a micro-level. In order to describe the consumption of resources at a micro-level, the model requires a micro-data-set for the construction of a consumption model. For a traditional analysis, the implemented micro-data-set would be a survey that:

1. is representative of the underlying population and
2. contains parameters required for the estimation of consumption intensities.

A traditional approach would simply reweight this survey to the analysis area (in this case a specific city) and recompute the consumption intensities. The problems with this approach are three-fold:

1. this approach requires a detailed survey for the specific resource to be estimated;
2. the survey is not representative for the projection of the population and

3) the method does not allow for an integrated analysis, i.e. combining variables from different surveys.

The second point can be solved through the implementation of a dynamic population model at the expense of a considerable increase in data input requirements.

The presented approach solves all three problems of the traditional approach by constructing a synthetic sample via a Markov-Chain-Monte-Carlo (MCMC) sampling procedure. Instead of using a sample survey as input the model defines probability distributions for the individual variables (and eventually links between these variables). The probability distributions can be defined based on variables from a sample survey but can also be derived from other data sources. Because the sample survey is synthetically constructed it can be re-constructed on each simulation step, by doing this the synthetic sample survey is always representative of the underlying population. The constructing of a sample survey at each simulation step comes at the cost of computational time. The synthetic sample survey is benchmarked to know aggregated demographic variables—and consumption values if available—with help of a sample reweighting algorithm (GREGWT).

3.1 The simulation framework

The first step of the simulation framework is the construction of the synthetic sample survey via the MCMC algorithm. On the second step, GREGWT reweights this sample to aggregated statistics. This means that the weight for each record is recomputed. The sample starts with a uniform distributed weight, e.g. every record represents w number of households. This means that the difference between the sample size n (number of records in the sample) and the actual number of households N_{HH} is equal to $n \times w$. The sample size can be redefined to any given number, this means that the sample size can be larger than the total number of households. The GREWT algorithm will re-compute these weights in order to match them to known aggregates. This procedure assures that the marginal sums of the sample survey match aggregated statistics. For example, from national statistics the model knows that in a specific city there are 100 households that do not have air conditioning. GREGWT will make sure that when we sum the weight of all households in the synthetic sample survey, the number of households that do not have air conditioning will be equal to 100.

This procedure is performed at each simulation step. A disadvantage of performing these steps (MCMC + GREGWT) at each simulation step is that the MCMC algorithm is very computationally intensive. Depending on the defined simulation scenarios a simpler method can be applied. Instead of resampling on each simulation step (MCMC + GREGWT; resampling method) the model can resample only once, e.g: for a benchmark year, and then reweight this sample for each consequent simulation year (reweighting method).

3.2 Internal and external model error

Fig. 3.1 shows a graph describing the model error. The upper-right plot shows the Percentage Specific Absolute Error (PSAE) error for each variable category, this value measures the distance between estimated and known (or extrapolated) aggregated values. The upper-right plot shows the percentage deviation between estimated consumption levels and known consumption levels (the plots for Figure 24 and Figure 26 are empty because known consumption levels are only available for the benchmark year 2016). The lower-right plot shows a regression line between simulated values (x-axis) and observed values (y-axis). Observed values are the known or extrapolated values at an aggregate level. The lower-right plot shows the initial uniform distributed weight (dotted red line, the output of MCMC) and the reweighted weights distribution (output of GREGWT).

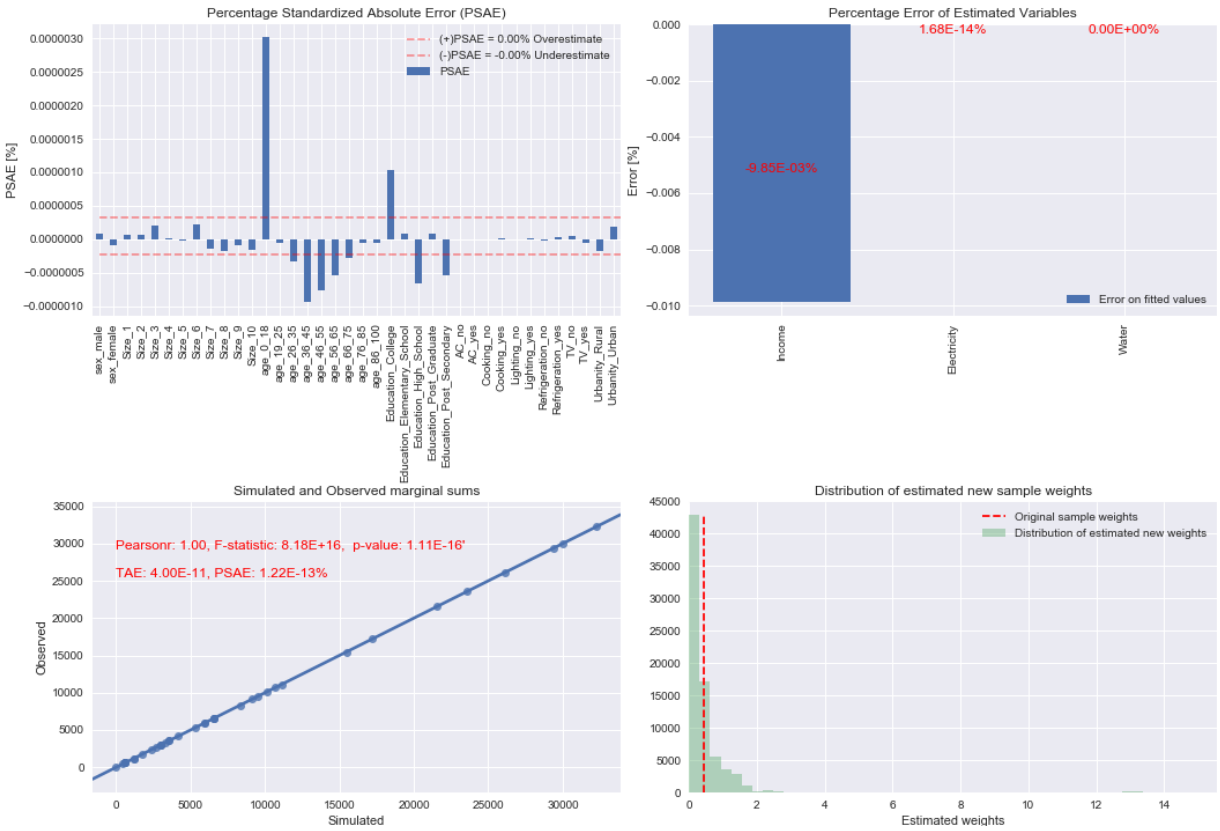


Fig. 3.1: Internal simulation error

CHAPTER 4

Examples

4.1 Running a simple example

The following steps are required to construct a minimal simulation example.

1. Define model parameters.
2. We define a formula for the electricity model. This model will compute the electricity demand based on previously computed income levels.
3. We define a python dictionary to tell the function *run_calibrated_model* how to calibrate the model. The order of the models (i.e. dictionary keys) matter as the model will calibrate them in the specified order. In this case, we need to calibrate the income model first in order to calibrate the electricity model because the computation of electricity directly depends on the estimation of income.
4. We run the model with the defined parameters. The model with iterate until all models are calibrated.

```
# load libraries
import pandas as pd
from urbanmetabolism.population.model import run_calibrated_model

# load model coefficients
elec  = pd.read_csv('data/test_elec.csv',   index_col=0)
inc   = pd.read_csv('data/test_inc.csv',    index_col=0)
water = pd.read_csv('data/test_water.csv',  index_col=0)

# define model, order matters
model = {"Income":      {'table_model': inc },
         "Water":       {'table_model': water},
         "Electricity": {'table_model': elec}

# run simulation
run_calibrated_model(
    model,
    census_file = 'data/benchmarks_projected.csv',
    year = 2016)
```

This subsection describes the required steps to perform a simple simulation. The steps to perform a simulation are two-fold:

1. The definition of a consumption model; and
2. The construction of scenarios

The definition of consumption models is required for the estimation of consumption levels at a micro-level. The consumption models estimate resource consumption intensities at an individual level based on predefined consumption drivers. The consumption model implemented in the simulation can be any type of resource demand model.

The definition of scenarios is performed at an aggregate level (simple scenario) or at a micro-level (advanced scenario). The construction of simple scenarios is performed by extrapolating the driver variables at an aggregated level. An advanced scenario will update the consumption model itself for each simulation year.

4.1.1 Consumption models

In order to explain the consumption of resources at a micro-level, the model requires a defined consumption model. Table 4.1 lists the input data passed to the urban metabolism model, defining the model used for the estimation of income. Income levels are subsequently used for the estimation of electricity and water demand.

For the construction of a sample survey, the model requires a set of parameters for each variable:

1. Mean value of variable coefficient *[co_mu]*

   This is the *mu* value (μ value) used to define a sampling probability distribution (normal distributed) for the variable coefficient.

   This coefficient indicates the effect that the variable has on income.

   If the variable is defined as categorical (see parameter *dis*) the model requires a *co_mu* value for each category.

   If the variable is defined as deterministic (see parameter *dis*) the model requires no *co_mu* value.

2. Standard deviation of variable coefficient *[co_sd]*

   The second parameter the model needs in order to define a normal distribution is the standard deviation of the sampling probability distribution.

   With the values of *co_mu* and *co_sd*, the model constructs a normal distribution. The model will take samples based on this distribution.

   If the variable is defined as categorical (see parameter *dis*) the model requires a *co_sd* value for each category.

   If the variable is defined as deterministic (see parameter *dis*) the model requires no *co_sd* value.

3. Probability value for dichotomous variables (single value) or categorical distributions (probability vector) *[p]*

   For dichotomous variables (yes/no, female/male, 1/0, etc.. . .) a single value is assigned to parameter *p*. Parameter *p* is the probability for the variable to take one of these values.

   The *p value* of the first row of Table 4.1 is set to 0.19. This means that at simulation year 2010 the model will sample a household with a female head (male is the reference category) with a 19% probability.

   If the expected variable is categorical (highmiddle/low, 1/2/3, etc.) the value assigned to parameter *p* is a vector with size equal to the number of categories.

4. Mean value of distribution *[mu]*

   This is the *mu* value (μ value) used to define a sampling probability distribution for the variable value.

   This parameter has to be defined for either continuous variables (normal distribution) of for discrete variables (poisson distribution).

5. Standard deviation for distribution *[sd]*

   For continuous distributed variables (normal distribution) the model needs to define a value for its standard deviation.

6. A distribution type *[dis]*

   This parameter defines the sampling probability distribution for the variable values. The Markov-Chain-Monte-Carlo routine will use this distribution (define through parameters *mu* and *sd*) to sample the variable values.

   The variable coefficients are by default sampled along a normal distribution, defined by *co_mu* and *co_sd*.

7. An upper boundary *[ub]*

   An optional Upper Boundary defined by parameter *ub* can be imputed into the simulation model. This parameter will cap the sampled variable value at this value.

8. A lower boundary *[lb]*

   Identical to parameter *ub* but for the lower boundary of the sampled variable.

Table 4.1: Income table-model for Sorsogon City (benchmark year 2016; in Philippine-Pesos)

	co_mu	*co_sd*	*p*	*mu*	*sd*	*dis*	*ub*	*lb*
i_Intercept			1,147.66			Deterministic		
i_Sex	919.01	161.50	0.20			Bernoulli		
i_Urbanity	7,105.22	127.94	0.47			Bernoulli		
i_FamilySize	1,666.85	29.03	5.25	2.24		Poisson	10	1
i_Age	116.58	4.68		54.18	14.07	Normal	100	18
i_Education	1.0, . . . , 16788.04	0.0, . . . , 742						

The defined income model estimates income levels at a household level. The variables used for the estimation are:

1. Gender of the household head

   This variable defines the gender of the household head. The model assumes a 919.01 Philippine-Pesos increase in household income if the household head is female. The model will construct a synthetic sample where 20% of all household heads are female.

2. Urbanity

   Defines if the household is classified as urban or rural. The category "rural" is used as reference category. The coefficient describes a positive influence on income, urban households are attributed on average 7 105.22 Philippine-Pesos with a standard deviation of 127.94 Pesos.

3. Family size

   The impact of family income by household size is an additional 1666.85 Pesos per additional family member.

4. Age of head of household

   The age of the household head is modelled as a continues variable. This is defined by setting the parameter *dis* to "Normal" (the normal distribution is a continuous distribution). The shape of the distribution is defined by variables *mu* and *sd*.

   The impact of household head age on income is defined by parameter *co_mu*, set to 116.58 with a standard deviation of 4.68 (via parameter *sd*).

5. The education level of head of household

   The education level of the household head also impacts the income level of the household. In this case, the model defines this variable as categorical. For each category the impact on income, relative to the reference level (Elementary School), has to be defined on parameter *co_mu*.

The model will create a synthetic sample following these parameters. On a second step, the model calibrates the estimated income levels to a known income value i.e. the aggregated total income of the city.

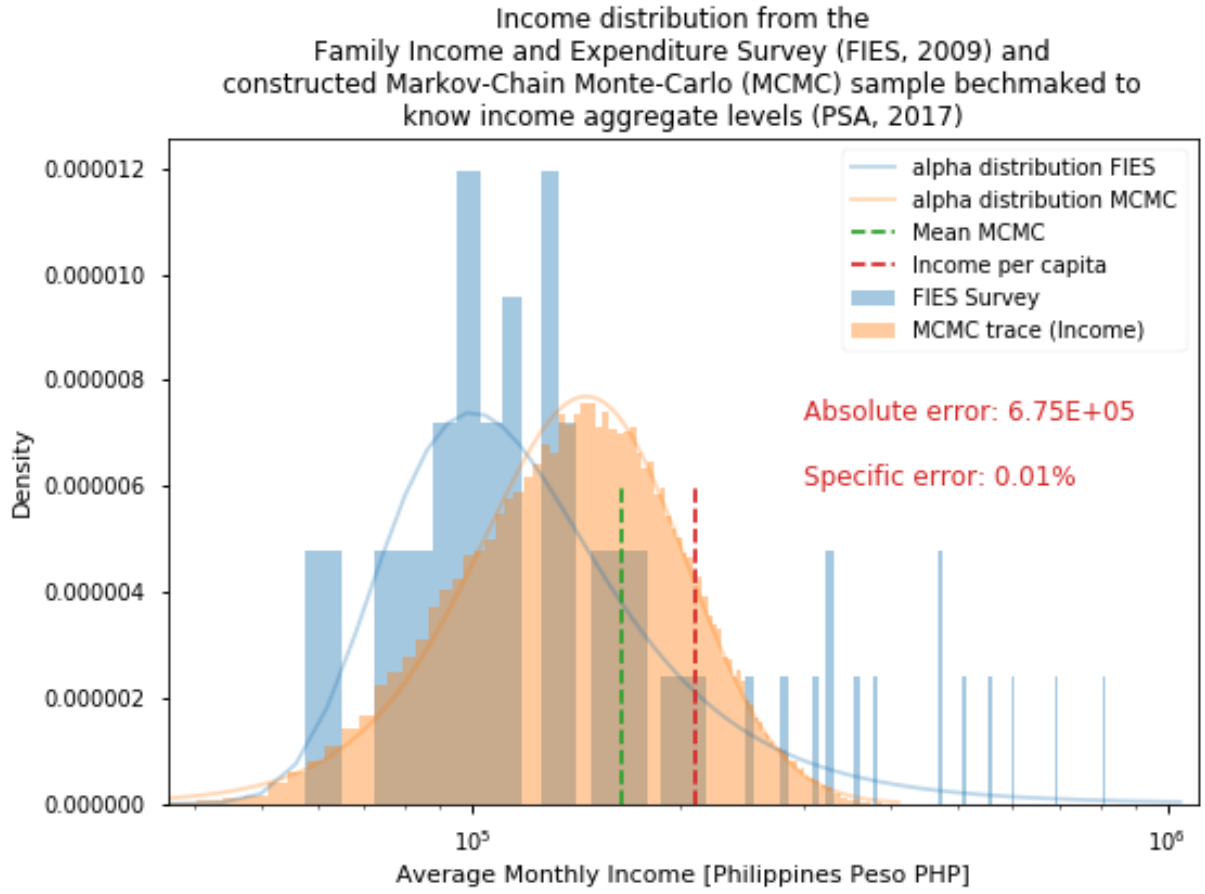


Fig. 4.1: Prior income distribution and calibrate posterior distribution

Fig. 4.1 shows the histogram of the original data used in the regression model, required for the estimation of regression coefficients used on variable *co_mu*, the known income level for 2016 (dotted green line) and the histogram of the constructed sample survey income levels. The figure also shows the absolute and specific error of the calibration. The estimated total income, i.e. the sum of all households' income in the synthetic sample survey differs by 0.01% from the official total income of the city reported in 2016. This means that the income estimation of the model has been calibrated properly. The computed calibration *k-factor* is used for the estimation of income for all other simulation years.

Following this schema, the model is able to compute all type of variables. In this section, the model is implemented for the estimation of electricity consumption levels as well as water consumption levels. The estimation of water and electricity consumption makes use of previously estimated income levels for their computation as well as demographic variables sampled for the estimation of income levels.

Table 4.2: Electricity table-model

	co_mu	co_sd	p	dis	ub	lb
e_Intercept			3.30	Deterministic		
e_Lighting	0.83	18.67	0.92	Bernoulli		
e_TV	18.79	1.76	0.72	Bernoulli		
e_Cooking	28.89	1.97	0.01	Bernoulli		
e_Refrigeration	59.24	1.56	0.34	Bernoulli		
e_AC	203.32	3.13	0.10	Bernoulli		
e_Urban	24.59	1.39	1.00	Bernoulli		
e_Income	0.00	0.00		None	inf	0.00

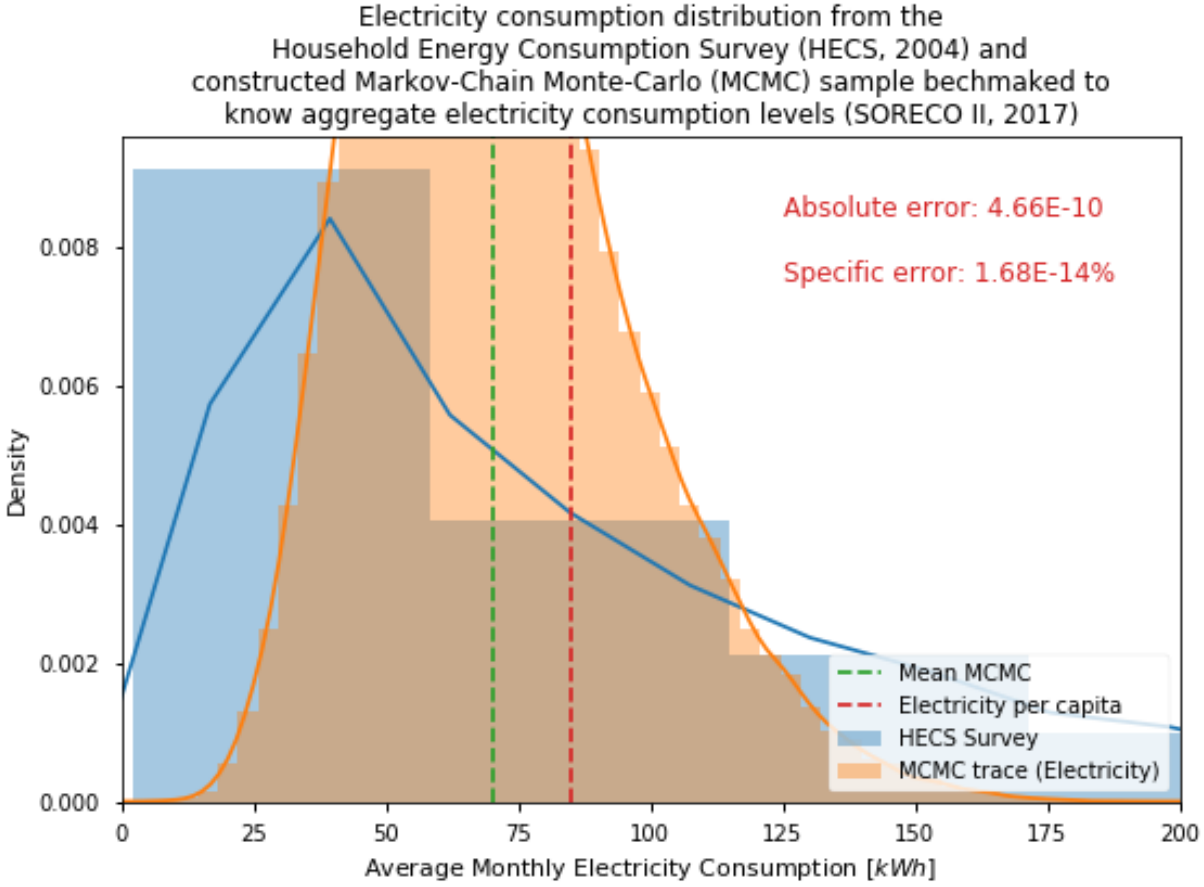


Fig. 4.2: Estimated electricity distribution

Table 4.2 describes the implemented model for the estimation of electricity demand. Analogues to the model defined for the estimation of income, the table list a set of variables used for the estimation of electricity consumption. These variables are described by their distribution (required for sampling them) and their coefficients.

The variables used in this example for the estimation of electricity consumption are the following:

1. AC

   This variable is one of the most important variables for the estimation of electricity consumption levels of individual households.

This variable describes the use of Air Conditioning in the household for cooling purposes.

2. Cooking

   This variable describes the impact on electricity demand of using an electric device for cooking.

3. Lighting

   This variable indicates the use of electric energy for the lighting of the house. This variable is directly related to electrification rate. By 2016 it is assumed that 97% of all households use electric energy for the lighting of their houses in Sorsogon City, the Philippines.

4. Refrigeration

   This variable describes the use of electricity for refrigeration purposes. Similar to the lighting variable, the model assumes that by 2016 all households in the city use electric energy for refrigeration.

5. TV

   This variable describes the use of electricity for TV and other leisure electric equipment like radios, computers and mobile phones.

6. Urban

   Analogues to the income estimation, the urbanization of a household has an impact on its electricity consumption.

   For 2016 the model assumes an urbanization rate of 65%.

Similar to the estimation of income, the estimation electricity is calibrated to the known city level electricity consumption level for the residential sector. Fig. 4.2 shows the estimation error of the model by comparing the calibrated estimated electricity consumption values from the synthetic sample survey to the consumption values from the Household Energy Consumption Survey HECS (PSA, 2004). The specific estimation error is close to zero with a value of 1.83e-4% (0.000183%).

Table 4.3: Water table-model

	co_mu	co_sd	p	dis
w_Intercept			-601.59	Deterministic
w_Sex	98.50	29.44	0.20	None
w_Urbanity	1,000.98	25.42	0.47	None
w_Total_Family_Income	0.05	0.00		None
w_FamilySize	49.74	5.90		None
w_Age	6.09	0.91		None
w_Education	1.0, . . . , 40.19	0.0, . . . , 119.68		None;i;Categorical

4.2 GI-REC Pilot Cities

4.2.1 Simple Example 1: Recife

Recife: Simple Simulation Example

```
In [1]: from smum.microsim.run import run_calibrated_model
        from smum.microsim.table import TableModel
```

```
/usr/lib/python3.6/site-packages/h5py-2.7.1-py3.6-linux-x86_64.egg/h5py/__init__.py:36: FutureWarning
  from ._conv import register_converters as _register_converters
```

1. Define model

```
In [52]: census_file = 'data/benchmarks_br_rec_year_bias.csv'

In [3]: tm = TableModel(census_file = census_file)

In [4]: tm.from_excel("data/tableModel_Water_rec.xlsx", "Water")
        formula_water = "w_Intercept +\
            c_w_ban       * w_ban + \
            c_w_connectio * w_connectio + \
            c_w_age       * w_age + \
            c_w_dutyp     * w_dutyp + \
            c_w_urban     * w_urban + \
            c_w_sex       * w_sex"
        tm.add_formula(formula_water, 'Water')

In [5]: tm.from_excel("data/tableModel_Electricity_rec.xlsx", "Electricity")
        formula_electricity ="e_Intercept + \
            c_e_edu    * e_edu + \
            c_e_sqm    * e_sqm + \
            c_e_income * e_income + \
            c_e_hhsize * e_hhsize + \
            c_e_dutyp  * w_dutyp  + \
            c_e_urban  * w_urban  + \
            c_e_sex    * w_sex  + \
            e_cdd"
        tm.add_formula(formula_electricity, 'Electricity')

In [168]: table_model = tm.make_model()
```

2. Define model variables

```
In [7]: labels_age = [
            'age_20a24', 'age_25a29', 'age_30a34',
            'age_35a39', 'age_40a44', 'age_45a49',
            'age_50a54', 'age_55a59', 'age_60a64',
            'age_65a69', 'age_70a74', 'age_75a79',
            'age_80anosou'
                  ]
        cut_age = [19,
               24, 29, 34,
               39, 44, 49,
               54, 59, 64,
               69, 74, 79,
               120]
        labels_hh = ['hhsize_{}'.format(i) for i in range(1, 8)]
        cut_hh = [0,1.55,2.55,3.55,4.55,5.55,6.55,20]
        to_cat = {'w_age':[cut_age, labels_age], 'e_hhsize':[cut_hh, labels_hh]}
```

3. Run Simulation

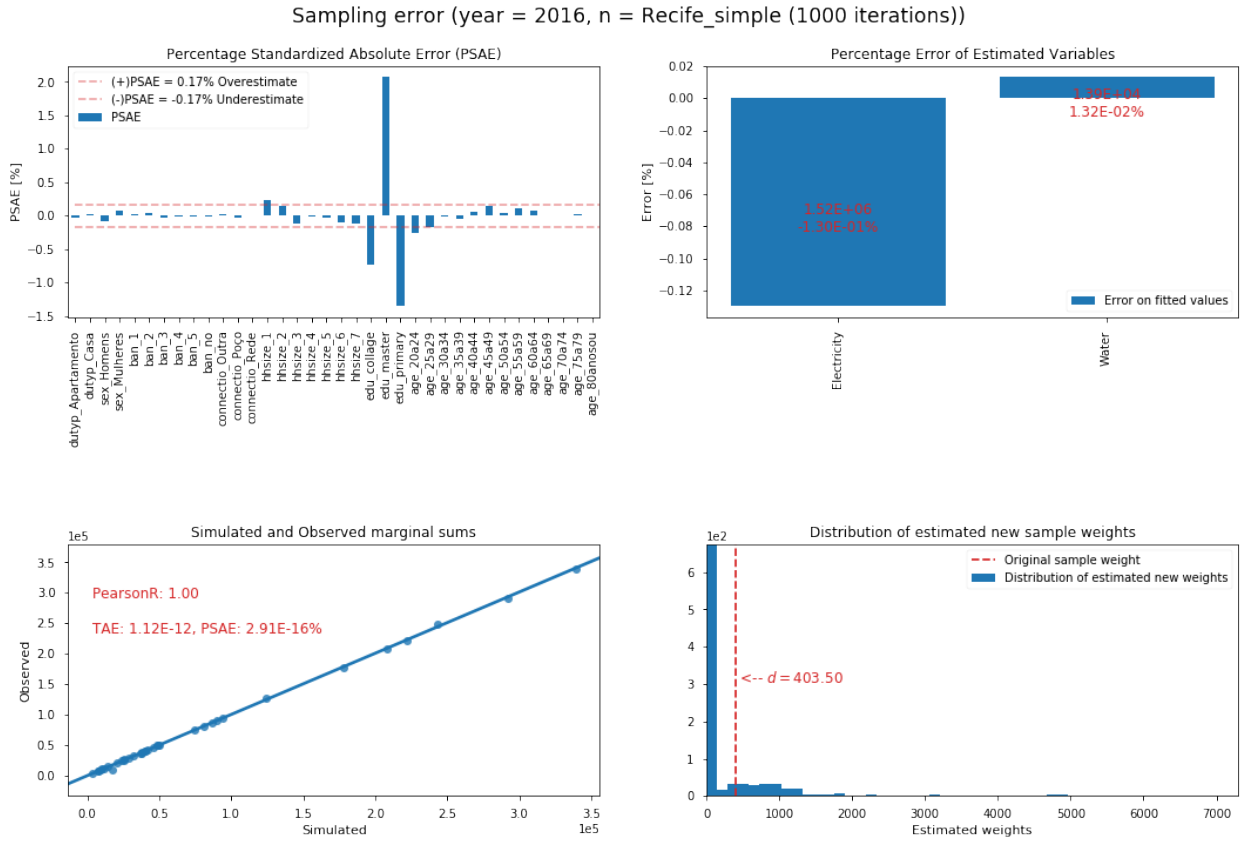
```
In [ ]: fw = run_calibrated_model(
            table_model,
            census_file = census_file,
            year = 2016,
            name = "Recife_simple",
            drop_col_survey = ['e_income', 'e_cdd'],
```

```
            to_cat = to_cat,
        )
```

4. Validate Simulation

```
In [9]: from smum.microsim.util_plot import plot_error
```

```
In [10]: REC = plot_error(
             'data/survey_Recife_simple.csv',
             'data/benchmarks_br_rec_year_bias.csv',
             'Recife_simple (1000 iterations)',
             year = 2016,
             skip = ['e_sqm'],
             fit_cols = ['Electricity', 'Water'],
         )
```




```
In [12]: import pandas as pd
         survey = pd.read_csv('data/survey_Recife_simple.csv', index_col=0)
```

```
In [22]: iss = survey.shape[0]
         rss = survey.loc[survey.wf > 0].shape[0]
         print("""
         The valid synthetic sample size is small!
         Initial sample size: {:0.0f}, representative sample size: {:0.0f}""".format(iss, rss))
         print("Only {:0.2%} of the synthetic sample is valid for this population".format(rss/iss))
```

```
The valid synthetic sample size is small!
Initial sample size: 956, representative sample size: 307
Only 32.11% of the synthetic sample is valid for this population
```

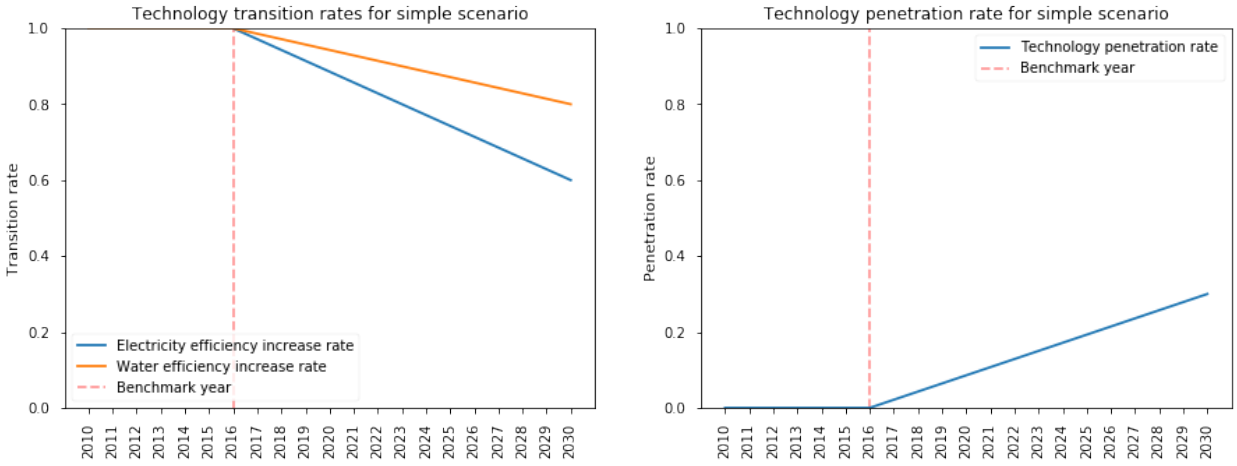
5. Define transition scenarios

```
In [1]: from smum.microsim.run import transition_rate
        from smum.microsim.util_plot import plot_transition_rate
        from smum.microsim.run import reduce_consumption
```

```
/usr/lib/python3.6/site-packages/h5py-2.7.1-py3.6-linux-x86_64.egg/h5py/__init__.py:36: FutureWarning
  from ._conv import register_converters as _register_converters
```

```
In [2]: Elec  = transition_rate(0, 0.4, start=2016)
        Water = transition_rate(0, 0.2, start=2016)
        pr    = transition_rate(0, 0.3, start=2016)
```

```
In [3]: plot_transition_rate(
            {"Technology penetration rate": pr,
             "Electricity efficiency increase rate": Elec,
             "Water efficiency increase rate": Water},
             "simple scenario")
```




```
In [4]: sampling_rules = {
            "e_edu == 'edu_master'": 30,
            "e_hhsize == 'hhsize_1'": 20,
            "e_hhsize == 'hhsize_2'": 10,
            "w_dutyp == 'dutyp_Casa'": 5,
        }
        file_name = "data/survey_Recife_simple.csv"
        scenario_name="simple_scenario"
        reduction = {'Electricity':Elec, 'Water':Water}
```

```
In [5]: reduce_consumption(
            file_name, pr, sampling_rules, reduction, scenario_name=scenario_name)
```

```
00.00%   Electricity   reduction; efficiency 00.00%; penetration 00.00; year 2010
-0.00%      Water      reduction; efficiency 00.00%; penetration 00.00; year 2010
00.00%   Electricity   reduction; efficiency 00.00%; penetration 00.00; year 2011
-0.00%      Water      reduction; efficiency 00.00%; penetration 00.00; year 2011
00.00%   Electricity   reduction; efficiency 00.00%; penetration 00.00; year 2012
-0.00%      Water      reduction; efficiency 00.00%; penetration 00.00; year 2012
-0.00%   Electricity   reduction; efficiency 00.00%; penetration 00.00; year 2013
00.00%      Water      reduction; efficiency 00.00%; penetration 00.00; year 2013
00.00%   Electricity   reduction; efficiency 00.00%; penetration 00.00; year 2014
00.00%      Water      reduction; efficiency 00.00%; penetration 00.00; year 2014
00.00%   Electricity   reduction; efficiency 00.00%; penetration 00.00; year 2015
-0.00%      Water      reduction; efficiency 00.00%; penetration 00.00; year 2015
```

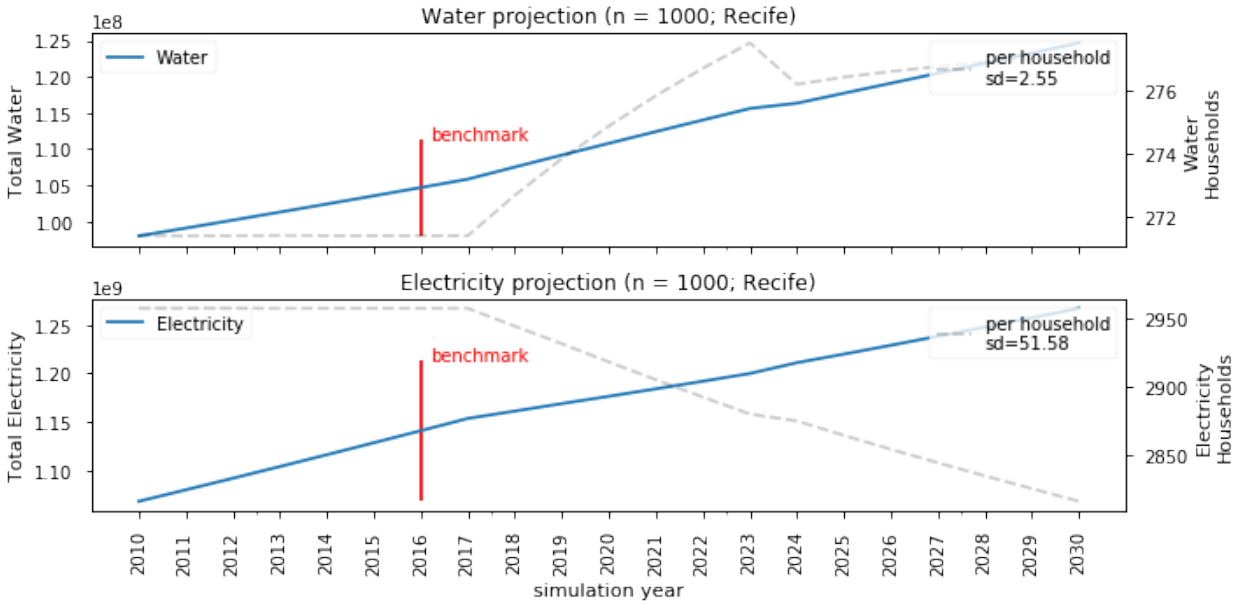
```
-0.00%   Electricity   reduction; efficiency 00.00%; penetration 00.00; year 2016
00.00%      Water      reduction; efficiency 00.00%; penetration 00.00; year 2016
00.06%   Electricity   reduction; efficiency 02.86%; penetration 00.02; year 2017
00.03%      Water      reduction; efficiency 01.43%; penetration 00.02; year 2017
00.24%   Electricity   reduction; efficiency 05.71%; penetration 00.04; year 2018
00.12%      Water      reduction; efficiency 02.86%; penetration 00.04; year 2018
00.53%   Electricity   reduction; efficiency 08.57%; penetration 00.06; year 2019
00.27%      Water      reduction; efficiency 04.29%; penetration 00.06; year 2019
00.94%   Electricity   reduction; efficiency 11.43%; penetration 00.09; year 2020
00.48%      Water      reduction; efficiency 05.71%; penetration 00.09; year 2020
01.47%   Electricity   reduction; efficiency 14.29%; penetration 00.11; year 2021
00.75%      Water      reduction; efficiency 07.14%; penetration 00.11; year 2021
02.12%   Electricity   reduction; efficiency 17.14%; penetration 00.13; year 2022
01.08%      Water      reduction; efficiency 08.57%; penetration 00.13; year 2022
02.89%   Electricity   reduction; efficiency 20.00%; penetration 00.15; year 2023
01.48%      Water      reduction; efficiency 10.00%; penetration 00.15; year 2023
03.78%   Electricity   reduction; efficiency 22.86%; penetration 00.17; year 2024
01.93%      Water      reduction; efficiency 11.43%; penetration 00.17; year 2024
04.78%   Electricity   reduction; efficiency 25.71%; penetration 00.19; year 2025
02.45%      Water      reduction; efficiency 12.86%; penetration 00.19; year 2025
05.90%   Electricity   reduction; efficiency 28.57%; penetration 00.21; year 2026
03.02%      Water      reduction; efficiency 14.29%; penetration 00.21; year 2026
07.15%   Electricity   reduction; efficiency 31.43%; penetration 00.24; year 2027
03.66%      Water      reduction; efficiency 15.71%; penetration 00.24; year 2027
08.50%   Electricity   reduction; efficiency 34.29%; penetration 00.26; year 2028
04.35%      Water      reduction; efficiency 17.14%; penetration 00.26; year 2028
09.98%   Electricity   reduction; efficiency 37.14%; penetration 00.28; year 2029
05.10%      Water      reduction; efficiency 18.57%; penetration 00.28; year 2029
11.58%   Electricity   reduction; efficiency 40.00%; penetration 00.30; year 2030
05.92%      Water      reduction; efficiency 20.00%; penetration 00.30; year 2030
```

6. Visualize transition scenarios

```
In [6]: from smum.microsim.util_plot import plot_data_projection
```

```
In [7]: iterations = 1000
        typ = 'reweighted'
        reweighted_survey = 'data/survey_Recife_simple'
```

```
In [9]: var = ['Water', 'Electricity']
        data, cap = plot_data_projection(
            reweighted_survey, var, "{}; Recife".format(iterations),
            benchmark_year=2016,
        )
```



4.2.2 Advanced Example 1: Sorsogon

Sorsogon, Philippines

Abstract:

This is a simple implementation example of the developed **Spatial Microsimulation Urban Metabolism Model (SMUM)**.

The aim of this model is to identify and quantify the impact of transition pathways to a circular economy.

Two main algorithms implemented in the model, giving it it's name, are:

1. A Spatial Microsimulation, used for the construction of a synthetic population; and
2. An Urban Metabolism approach, used to benchmark consumption level at a city-level or neighbourhood level (making it spatial).

(Step 1) Constructing a synthetic population

On this section the example shows how to construct representative samples given the **distributions of aggregated variables**. The algorithm constructs the samples either by constructing a new sample for each simulation year (**resample method**) or by reweighting an initial sample for each simulation year (**reweighting method**).

In order to construct the samples the model needs input on: (a) the distribution functions of aggregate variables and (b) the changes over time. This means that the model required the **projected aggregated values**. The model provides some function to facilitate and visualize the projection of aggregated values.

For a spatial microsimulation, the model requires also values for each simulation area. In this case a combination of the resample and reweighting methodologies is implemented. The algorithm will construct a new sample for each simulation year at an aggregated level (e.g. city-level) and reweight this sample for each area (e.g. statistical census areas).

For the computation of resource consumption the model requires a **consumption model**. This model defines how the resource values are computed. This can be anything from a simple linear model to the implementation of external libraries.

Aggregate level benchmarks

(Step 1.a) Projecting demographic variables

Micro level consumption models

(Step 1.b) Micro-level Income model

(Step 1.c) Micro-level Electricity demand model

(Step 1.d) Micro-level Water demand model

(Step 1.e) Micro-level Non-Residential model

(Step 2) Sampling and reweighting

This section takes care of the actual **sampling procedure** and subsequent reweighting of the proxy data. The sample will be constructed with help of an MCMC algorithm and reweighted with help of the GREGWT algorithm.

This section also presents the internal validation of the sampling procedure.

Dynamic samplic models

(Step 2.a) Dynamic Sampling Model and GREGWT

(Step 2.b) Non-Residential Model

Model internal validation

(Step 2.c) Model Internal Validation

(Step 3) Constructing scenarions:

The construction of scenarios can be define at **different steps** of the model. In a sense, the definition of scenarios start by the projection of aggregated values (see section 1). This section defines scenarios by defining technology adoption rates and changes in technology efficiency.

(Step 3.a) Define Transition Scenarions

(Step 3.b) Visualize transition scenarios

Sorsogon. Step 1.a Projecting demographic variables

```
In [1]: import datetime; print(datetime.datetime.now())
```

```
2018-03-15 13:51:56.478760
```

Notebook Abstract:

This notebook shows an example on how to prepare projected aggregated benchmarks, as well as how to introduce **bias** (a proxy for simulating development scenarios at an aggregated level) to the benchmarks.

This step is not a requirement for running the simulation, the data containing projections at an aggregated level are a **requirement** for running the simulation.

A spatial model requires projected aggregated values for each simulation area. Depending on the simulation model the estimation of such values might be difficult.

Aggregated census data

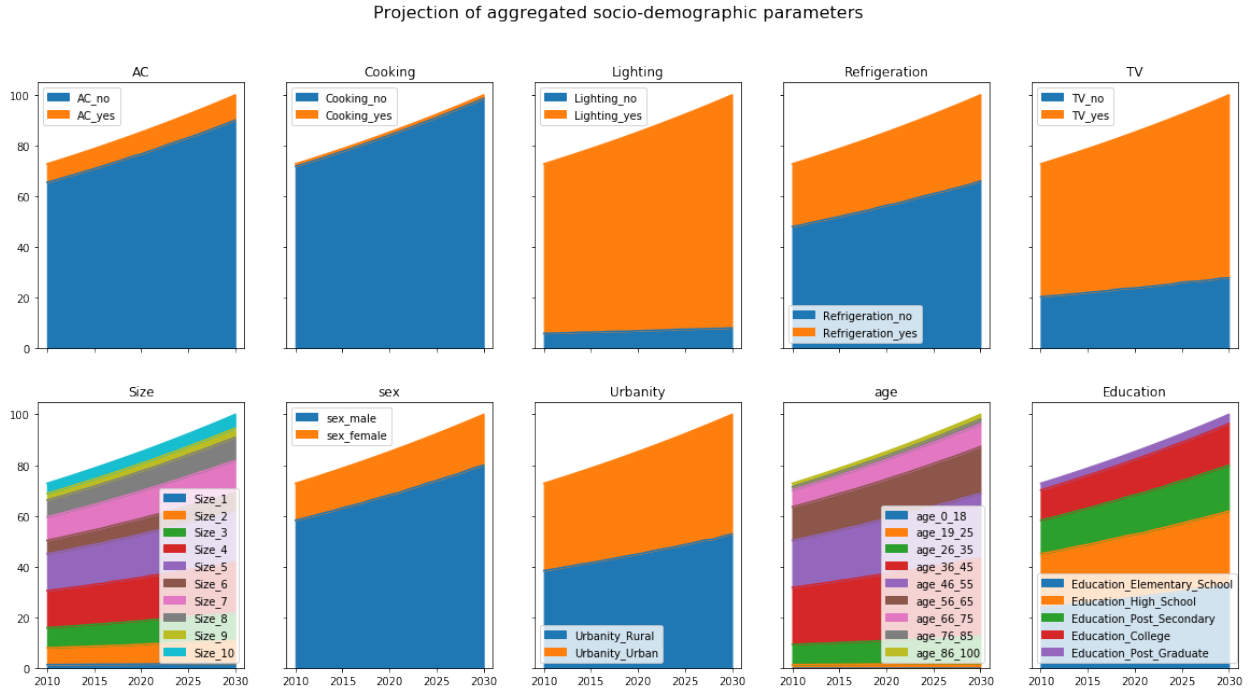
```
In [2]: import pandas as pd
        from smum._scripts.aggregates import print_all
```

```
In [3]: census = pd.read_csv('data/benchmarks_projected.csv', index_col=0)
        skip = ['pop', 'Income', 'Water', 'Electricity']
```

We read a csv file containing the projected data with help of pandas. The `print_all` function is used to print the aggregated data.

A list of strings is constructed to tell the function which columns to ignore during plotting.

```
In [4]: _ = print_all(
            census, 'year',
            skip = skip,
            total_pop = census.loc[:, 'pop'],
            title="Projection of aggregated socio-demographic parameters"
                  )
```



We pass the following variables to the `print_all` function:

1. A pandas DataFrame containing the projected aggregated data.
2. A string defining the sufix used to save the file on disk.

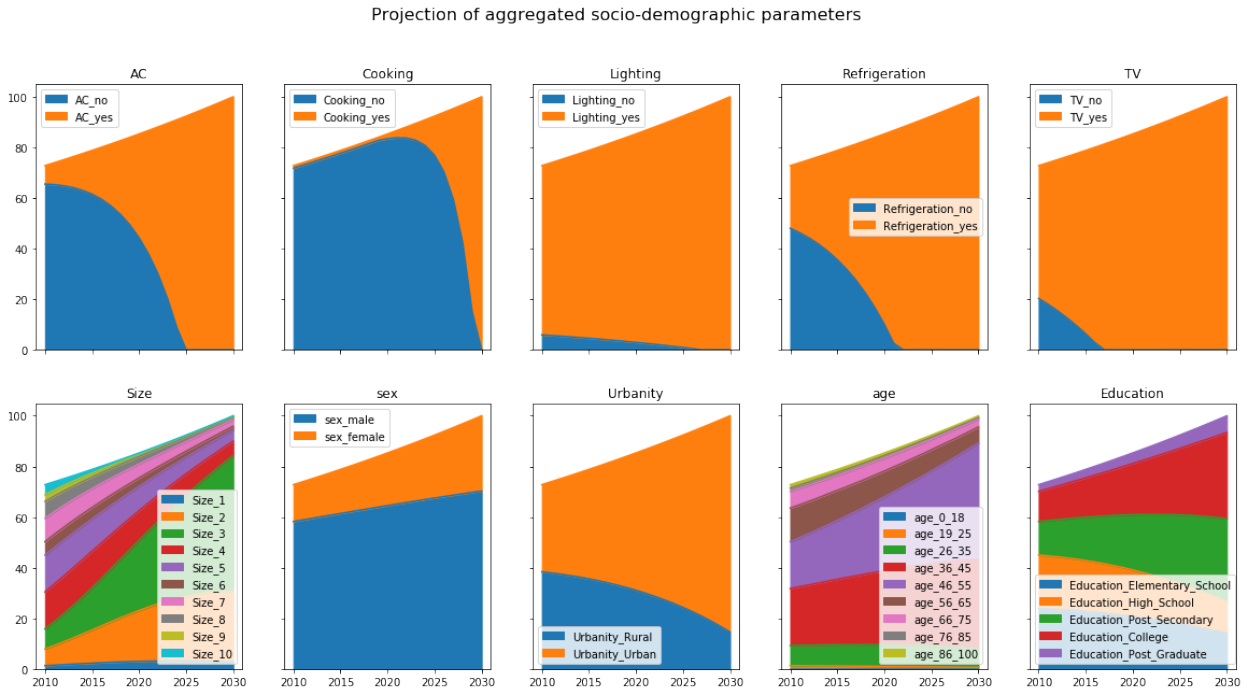
3. `var` = defines the type of plot to use.
4. `skip` = defines the list of columns to skip in the plot.
5. `total_pop` = defines a column to use for data normalization.
6. `title` = defines the plot title.

Introduce bias to census data

The manipulations are expressed as **growth factors**. This factors increase the share (>1) or decrease the share (<1) of specific categories. In addition we can define gradual changes on these growth factors. If no starting point is given the function assumes initial simulation year.

The define growth rates are passed as a python dictionary to the function (`{'key'=value}`). Where `key` is the variable category to be modified and `value` is either a single number (assumed initial year to be initial simulation year) or a dictionary attributing a growth rate to sequential steps.

```
In [5]: bias_to = {
            'AC_yes': 1.17,
            'Cooking_yes': {2010: 1.01, 2020: 1.5},
            'Lighting_yes': 1.005,
            'Refrigeration_yes': 1.1,
            'TV_yes': 1.05,
            'Size_1': 1.05,
            'Size_2': 1.07,
            'Size_3': 1.07,
            'Size_4': 0.93,
            'Size_5': 0.92,
            'Size_6': 0.92,
            'Size_7': 0.92,
            'Size_8': 0.90,
            'Size_9': 0.88,
            'Size_10': 0.80,
            'sex_female':1.02,
            'Urbanity_Urban': 1.03,
            'age_26_35':1.03,
            'age_36_45':1.03,
            'age_46_55':1.03,
            'Education_Post_Secondary':1.05,
            'Education_College':1.04,
            'Education_Post_Graduate':1.03
        }

In [6]: _ = print_all(
            census, 'year_bias',
            skip = skip,
            bias = bias_to,
            total_pop = census.loc[:, 'pop'],
            title="Projection of aggregated socio-demographic parameters",
            save_data = 'data/benchmarks_year_bias.csv'
                  )
```



Sorsogon. Step 1.b Micro-level Income model

```
In [1]: import datetime; print(datetime.datetime.now())
```

```
2017-10-25 14:37:54.542583
```

Notebook Abstract:

A simple micro-level income model. The following notebook presents the defined income model for the simulation.

The model computes income levels for each household on the proxy sample data. The data used for the estimation of income drivers and their corresponding coefficients is the **Philippines Family Income and Expenditure Survey 2009**.

Prior income model

```
In [19]: import statsmodels.api as sm
         import pandas as pd
         import numpy as np
         from urbanmetabolism._scripts.micro import compute_categories, change_index
```

```
In [20]: income_data = pd.read_csv('data/income.csv', index_col=0)
         formula = "Total_Family_Income ~\
         Family_Size + C(HH_head_Sex) + HH_head_Age + C(Education) + C(Urbanity)"
```

```
In [21]: income_data.head()
```

```
Out[21]: Family_Size  HH_head_Sex  HH_head_Age  Education  Electricity_expenditure  \
         0          5.5            1           52        2.0                     1500
         1          7.5            1           70        1.0                     1608
         2          3.0            1           49        2.0                     8880
         3          2.0            2           51        1.0                      900
         4          6.0            1           36        1.0                     3360
```

```
   Water_expenditure  Total_Family_Income  Urbanity
0                  0         23939.666667         0
1                  0         16078.166667         0
2                  0         20925.000000         0
3               2190          9932.333333         0
4                  0         13589.500000         0
```

In [22]:
```
model_inc = sm.WLS.from_formula(formula, income_data)
model_results_inc = model_inc.fit()
```

In [23]:
```
model_results_inc.summary()
```

Out[23]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            WLS Regression Results
==============================================================================
Dep. Variable:     Total_Family_Income   R-squared:                       0.315
Model:                             WLS   Adj. R-squared:                  0.315
Method:                  Least Squares   F-statistic:                     1908.
Date:                 Mon, 23 Oct 2017   Prob (F-statistic):               0.00
Time:                         16:37:05   Log-Likelihood:            -3.5601e+05
No. Observations:                33208   AIC:                         7.120e+05
Df Residuals:                    33199   BIC:                         7.121e+05
Df Model:                            8
Covariance Type:             nonrobust
==============================================================================
                             coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept            1147.6640    313.997      3.655      0.000     532.218    1763.110
C(HH_head_Sex)[T.2]   919.0121    161.503      5.690      0.000     602.460    1235.565
C(Education)[T.2.0]  6023.8625    140.904     42.751      0.000    5747.685    6300.040
C(Education)[T.3.0]  1.196e+04    217.209     55.058      0.000    1.15e+04    1.24e+04
C(Education)[T.4.0]  1.873e+04    282.176     66.368      0.000    1.82e+04    1.93e+04
C(Education)[T.5.0]  1.679e+04    742.048     22.624      0.000    1.53e+04    1.82e+04
C(Urbanity)[T.1]     7105.2245    127.941     55.535      0.000    6854.455    7355.994
Family_Size          1666.8464     29.035     57.409      0.000    1609.937    1723.756
HH_head_Age           116.5759      4.681     24.902      0.000     107.400     125.752
==============================================================================
Omnibus:                     3597.783   Durbin-Watson:                   1.606
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             4994.573
Skew:                           0.865   Prob(JB):                         0.00
Kurtosis:                       3.786   Cond. No.                         642.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
"""
```

In [24]:
```
params_inc = change_index(model_results_inc.params)
bse_inc = change_index(model_results_inc.bse)
inc = pd.concat([params_inc, bse_inc], axis=1)
inc.columns = ['co_mu', 'co_sd']
inc = compute_categories(inc)
```

In [25]:
```
inc.loc['Urbanity', 'p'] = (income_data.Urbanity == 1).sum() / income_data.shape[0]
inc.loc['Sex', 'p'] = (income_data.HH_head_Sex == 2).sum() / income_data.shape[0]
```

In [26]:
```
inc.loc[:, 'mu'] = np.nan
inc.loc[:, 'sd'] = np.nan
inc.loc['Intercept', 'p'] = inc.loc['Intercept', 'co_mu']
inc.loc['Intercept', ['co_mu', 'co_sd']] = np.nan
```

```
In [27]: inc.loc['Education','dis'] = 'Categorical'
         inc.loc['Urbanity', 'dis'] = 'Bernoulli'
         inc.loc['Sex', 'dis'] = 'Bernoulli'
         inc.loc['FamilySize', 'dis'] = 'Poisson'
         inc.loc['Intercept', 'dis'] = 'Deterministic'
         inc.loc['Age', 'dis'] = 'Normal'

In [28]: inc.loc[:,'ub'] = np.nan
         inc.loc[:,'lb'] = np.nan
         inc.loc['FamilySize', 'lb'] = 1
         inc.loc['FamilySize', 'ub'] = 10
         inc.loc['Age', 'ub'] = 100
         inc.loc['Age', 'lb'] = 18

In [29]: inc.index = ['i_'+i for i in inc.index]

In [30]: inc.to_csv('data/table_inc.csv')

In [31]: inc

Out[31]: co_mu  \
         i_Intercept                                                    NaN
         i_Sex                                                      919.012
         i_Urbanity                                                 7105.22
         i_FamilySize                                               1666.85
         i_Age                                                      116.576
         i_Education   1.0,6023.86254599,11959.091528,18727.4606703,1...

                                                                      co_sd            p  \
         i_Intercept                                                    NaN  1147.663992
         i_Sex                                                      161.503     0.193718
         i_Urbanity                                                 127.941     0.403005
         i_FamilySize                                               29.0348          NaN
         i_Age                                                      4.68139          NaN
         i_Education   1e-10,140.904404522,217.208790314,282.17614554...          NaN

                       mu  sd            dis     ub    lb
         i_Intercept  NaN NaN  Deterministic    NaN   NaN
         i_Sex        NaN NaN      Bernoulli    NaN   NaN
         i_Urbanity   NaN NaN      Bernoulli    NaN   NaN
         i_FamilySize NaN NaN        Poisson   10.0   1.0
         i_Age        NaN NaN         Normal  100.0  18.0
         i_Education  NaN NaN    Categorical    NaN   NaN
```

The income model is defined as a **table model**. This table contains all the required information for the simulation model to construct a proxy sample.

The table model defines the coefficient used for the estimation of income `co_mu`, with a corresponding standard deviation `co_sd`. A value to model the distribution (`p`, `mu`, `sd`), the distribution type is defined on column `dis`. The values `ub` and `lb` are used to give the distribution an upper and lower bound.

Sorsogon. Step 1.c Micro-level Electricity demand model

```
In [1]: import datetime; print(datetime.datetime.now())

2018-03-15 13:51:59.686128
```

Notebook Abstract:

A simple micro-level electricity demand model. Similar to the income demand model, the electricity demand model used available micro level data for the estimation of regression coefficients. This regression coefficients are used to

define a table model. The electricity table model is used for the construction of a proxy micro level sample data set.

Prior electricity demand model

```
In [2]: import statsmodels.api as sm
        import pandas as pd
        import numpy as np
        from smum._scripts.micro import compute_categories, change_index
```

```
/usr/lib/python3.6/site-packages/statsmodels-0.8.0-py3.6-linux-x86_64.egg/statsmodels/compat/pandas.p
  from pandas.core import datetools
```

```
In [3]: electricity_data = pd.read_csv('data/electricity.csv', index_col=0)
        formula = "Electricity ~ C(Lighting) + C(TV) + C(Cooking) + C(Refrigeration) + C(AC) + C(Urba
```

```
In [4]: electricity_data.head()
```

```
Out[4]: Lighting  TV  Cooking  Refrigeration  AC  Urban   Income  Electricity
        0         1   1        1              1   0      0  16000.0        110.0
        1         1   1        0              1   0      0   4000.0         80.0
        2         1   1        0              1   0      0   6000.0         47.0
        3         1   1        0              0   0      0   6300.0         17.0
        4         1   1        0              0   0      0   5000.0         17.0
```

```
In [5]: model_elec = sm.WLS.from_formula(formula, electricity_data)
        model_results_elec = model_elec.fit()
```

```
In [6]: model_results_elec.summary()
```

```
Out[6]: <class 'statsmodels.iolib.summary.Summary'>
        """
                                    WLS Regression Results
        ==============================================================================
        Dep. Variable:            Electricity   R-squared:                       0.516
        Model:                            WLS   Adj. R-squared:                  0.516
        Method:                 Least Squares   F-statistic:                     2519.
        Date:                Thu, 15 Mar 2018   Prob (F-statistic):               0.00
        Time:                        13:52:02   Log-Likelihood:                -96932.
        No. Observations:               16522   AIC:                         1.939e+05
        Df Residuals:                   16514   BIC:                         1.939e+05
        Df Model:                           7
        Covariance Type:            nonrobust
        ================================================================================================
                                       coef    std err          t      P>|t|      [0.025      0.975]
        ------------------------------------------------------------------------------------------------
        Intercept                    3.3000     18.699      0.176      0.860     -33.351      39.951
        C(Lighting)[T.1]             0.8257     18.668      0.044      0.965     -35.765      37.416
        C(TV)[T.1]                  18.7899      1.760     10.678      0.000      15.341      22.239
        C(Cooking)[T.1]             28.8862      1.969     14.671      0.000      25.027      32.746
        C(Refrigeration)[T.1]       59.2432      1.556     38.073      0.000      56.193      62.293
        C(AC)[T.1]                 203.3226      3.130     64.956      0.000     197.187     209.458
        C(Urban)[T.1]               24.5935      1.391     17.680      0.000      21.867      27.320
        Income                       0.0014    4.1e-05     34.765      0.000       0.001       0.002
        ==============================================================================
        Omnibus:                    20742.858   Durbin-Watson:                   1.789
        Prob(Omnibus):                  0.000   Jarque-Bera (JB):          9719176.595
        Skew:                           6.463   Prob(JB):                         0.00
        Kurtosis:                     121.115   Cond. No.                     8.75e+05
        ==============================================================================

        Warnings:
```

```
        [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
        [2] The condition number is large, 8.75e+05. This might indicate that there are
        strong multicollinearity or other numerical problems.
        """
```

```
In [7]: params_elec = change_index(model_results_elec.params)
        bse_elec = change_index(model_results_elec.bse)
        elec = pd.concat([params_elec, bse_elec], axis=1)
        elec.columns = ['co_mu', 'co_sd']
```

```
In [8]: elec.loc['Lighting', 'p'] = (electricity_data.Lighting == 1).sum() / electricity_data.shape[0]
        elec.loc['TV', 'p'] = (electricity_data.TV == 1).sum() / electricity_data.shape[0]
        elec.loc['Cooking', 'p'] = (electricity_data.Cooking == 1).sum() / electricity_data.shape[0]
        elec.loc['Refrigeration', 'p'] = (electricity_data.Refrigeration == 1).sum() / electricity_da
        elec.loc['AC', 'p'] = (electricity_data.AC == 1).sum() / electricity_data.shape[0]
        elec.loc['Urban', 'p'] = (electricity_data.Urban == 1).sum() / electricity_data.shape[0]
```

```
In [9]: elec.loc[:, 'mu'] = np.nan
        elec.loc[:, 'sd'] = np.nan
        elec.loc['Intercept', 'p'] = elec.loc['Intercept', 'co_mu']
        elec.loc['Intercept', ['co_mu', 'co_sd']] = np.nan
```

```
In [10]: elec.loc[:, 'dis'] = 'Bernoulli'
         elec.loc['Income', 'dis'] = 'None'
         elec.loc['Intercept', 'dis'] = 'Deterministic'
```

```
In [11]: elec.loc[:, 'ub'] = np.nan
         elec.loc[:, 'lb'] = np.nan
         elec.loc['Income', 'ub'] = np.inf
         elec.loc['Income', 'lb'] = 0
```

```
In [12]: elec.index = ['e_'+i for i in elec.index]
```

```
In [13]: elec.to_csv('data/table_elec.csv')
```

```
In [14]: elec
```

Out[14]:

	co_mu	co_sd	p	mu	sd	dis	ub	\
e_Intercept			NaN		NaN	3.299984	NaN	NaN
e_Lighting			0.825662		18.667601	0.998729	NaN	NaN
e_TV			18.789909		1.759621	0.782774	NaN	NaN
e_Cooking			28.886242		1.968938	0.167474	NaN	NaN
e_Refrigeration			59.243236		1.556048	0.436812	NaN	NaN
e_AC			203.322615		3.130158	0.059375	NaN	NaN
e_Urban			24.593500		1.391044	0.550236	NaN	NaN
e_Income			0.001426		0.000041	NaN	NaN	NaN

	dis	ub	lb
e_Intercept	Deterministic	NaN	NaN
e_Lighting	Bernoulli	NaN	NaN
e_TV	Bernoulli	NaN	NaN
e_Cooking	Bernoulli	NaN	NaN
e_Refrigeration	Bernoulli	NaN	NaN
e_AC	Bernoulli	NaN	NaN
e_Urban	Bernoulli	NaN	NaN
e_Income	None	inf	0.0

Sorsogon. Step 1.d Micro-level Water demand model

```
In [1]: import datetime; print(datetime.datetime.now())
```

```
2018-03-15 13:52:14.007748
```

Notebook abstract

A simple micro-level water demand model.

A simple micro-level water demand model. Similar to the income demand model and the electricity demand model, the water demand model uses micro level consumption demand data for the construction of a table model. The table model describes simple rules for the construction of the proxy micro level sample data.

Prior water demand model

```
In [2]: import statsmodels.api as sm
        import pandas as pd
        import numpy as np
        from smum._scripts.micro import compute_categories, change_index
```

```
/usr/lib/python3.6/site-packages/statsmodels-0.8.0-py3.6-linux-x86_64.egg/statsmodels/compat/pandas.p
  from pandas.core import datetools
```

```
In [3]: water_data = pd.read_csv('data/water.csv', index_col=0)
        formula = "Water_expenditure ~ Total_Family_Income + Family_Size + C(HH_head_Sex)\
        + HH_head_Age + C(Education) + C(Urbanity)"
```

```
In [4]: water_data.head()
```

Out[4]:

	Family_Size	HH_head_Sex	HH_head_Age	Education	Electricity_expenditure	\
3	2.0	2	51	1.0	900	
9	3.5	1	41	4.0	17202	
11	3.0	2	75	1.0	4212	
12	4.5	1	74	1.0	6210	
13	6.5	2	55	2.0	3900	

	Water_expenditure	Total_Family_Income	Urbanity
3	2190	9932.333333	0
9	300	47233.833333	0
11	3024	16521.333333	0
12	4086	20254.333333	0
13	2940	16368.000000	0

```
In [5]: model_water = sm.WLS.from_formula(formula, water_data)
        model_results_water = model_water.fit()
```

```
In [6]: model_results_water.summary()
```

Out[6]:

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            WLS Regression Results
==============================================================================
Dep. Variable:      Water_expenditure   R-squared:                       0.344
Model:                            WLS   Adj. R-squared:                  0.344
Method:                 Least Squares   F-statistic:                     925.6
Date:                Thu, 15 Mar 2018   Prob (F-statistic):               0.00
Time:                        13:52:16   Log-Likelihood:            -1.3853e+05
No. Observations:               15904   AIC:                         2.771e+05
Df Residuals:                   15894   BIC:                         2.771e+05
Df Model:                           9
Covariance Type:            nonrobust
================================================================================================
                           coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
Intercept            -601.5920     62.632     -9.605      0.000    -724.358    -478.826
C(HH_head_Sex)[T.2]    98.4950     29.444      3.345      0.001      40.782     156.208
```

```
C(Education)[T.2.0]   214.4011     28.816      7.440      0.000     157.919     270.883
C(Education)[T.3.0]   260.3273     40.057      6.499      0.000     181.810     338.844
C(Education)[T.4.0]   101.7028     49.996      2.034      0.042       3.705     199.700
C(Education)[T.5.0]    40.1879    119.681      0.336      0.737    -194.400     274.775
C(Urbanity)[T.1]     1000.9789     25.416     39.384      0.000     951.161    1050.797
Total_Family_Income     0.0532      0.001     54.145      0.000       0.051       0.055
Family_Size            49.7394      5.898      8.434      0.000      38.179      61.300
HH_head_Age             6.0889      0.913      6.671      0.000       4.300       7.878
==============================================================================
Omnibus:                     3226.790   Durbin-Watson:                   1.399
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             7597.976
Skew:                           1.142   Prob(JB):                         0.00
Kurtosis:                       5.499   Cond. No.                     3.10e+05
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.1e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

```
In [7]: params_water = change_index(model_results_water.params)
        bse_water = change_index(model_results_water.bse)
        water = pd.concat([params_water, bse_water], axis=1)
        water.columns = ['co_mu', 'co_sd']
        water = compute_categories(water)
```

```
In [8]: water.loc['Urbanity', 'p'] = (water_data.Urbanity == 1).sum() / water_data.shape[0]
        water.loc['Sex', 'p'] = (water_data.HH_head_Sex == 2).sum() / water_data.shape[0]
```

```
In [9]: water.loc[:,'dis'] = 'None'
        water.loc['Education', 'dis'] = 'None;i;Categorical'
        water.loc['Intercept', 'dis'] = 'Deterministic'
```

```
In [10]: water.loc[:, 'mu'] = np.nan
         water.loc[:, 'sd'] = np.nan
         water.loc['Intercept', 'p'] = water.loc['Intercept', 'co_mu']
         water.loc['Intercept', ['co_mu', 'co_sd']] = np.nan
```

```
In [11]: water.loc[:,'ub'] = np.nan
         water.loc[:,'lb'] = np.nan
```

```
In [12]: water.index = ['w_'+i for i in water.index]
```

```
In [13]: water.to_csv('data/table_water.csv')
```

```
In [14]: water
```

```
Out[14]: co_mu  \
         w_Intercept                                                    NaN
         w_Sex                                                        98.495
         w_Urbanity                                                  1000.98
         w_Total_Family_Income                                      0.053187
         w_FamilySize                                                49.7394
         w_Age                                                       6.08894
         w_Education           1.0,214.4011453125436,260.32727427717964,101.7...

                                                                       co_sd  \
         w_Intercept                                                     NaN
         w_Sex                                                         29.4438
         w_Urbanity                                                    25.4159
         w_Total_Family_Income                                    0.000982306
         w_FamilySize                                                  5.89779
```

```
w_Age                                                          0.912741
w_Education          1e-10,28.815802440470176,40.0574490885231,49.9...

                                  p                 dis  mu  sd  ub  lb
w_Intercept            -601.591950       Deterministic NaN NaN NaN NaN
w_Sex                     0.224786                None NaN NaN NaN NaN
w_Urbanity                0.593939                None NaN NaN NaN NaN
w_Total_Family_Income          NaN                None NaN NaN NaN NaN
w_FamilySize                   NaN                None NaN NaN NaN NaN
w_Age                          NaN                None NaN NaN NaN NaN
w_Education                    NaN  None;i;Categorical NaN NaN NaN NaN
```

Sorsogon Step 1.e Micro-level Non-Residential model

```
In [1]: import datetime; print(datetime.datetime.now())
```

```
2018-03-15 13:52:18.424295
```

Notebook abstract

A simple micro-level building stock model. The consumption model defined for the building stock works in theory exactly like the other micro level consumption model. The difference between this model and the income, electricity and water demand models is that we don't have a micro-level consumption data set in order to extract regression coefficients. In order to define a consumption model we use predefine building typologies.

Compile building level data

```
In [2]: from smum._scripts.sqm_data import get_pop_data, get_sqm_data
```

```
In [3]: census_file = 'data/benchmarks_year_bias.csv'
```

```
In [4]: pop_data_1 = get_pop_data(census_file, sqm_nonres_mean = 800)
        nr_data = get_sqm_data()
        pop_data = get_pop_data(
            census_file,
            sqm_nonres_mean = nr_data.loc[:, 'sqm'].mean(),
        )
        pop_data_3 = get_pop_data(census_file, sqm_nonres_mean = 500)
```

```
/usr/lib/python3.6/site-packages/pandas-0.22.0-py3.6-linux-x86_64.egg/pandas/util/_decorators.py:118
  return func(*args, **kwargs)
```

```
In [5]: benchmarks = pop_data.loc[:,['sqm_nonres', 'num_nonres']]
        benchmarks.columns = ['BuildingSqm', 'pop']
```

```
In [6]: kwh_2016 = 6.52 * 1000000
```

```
In [7]: pb_kwh = kwh_2016 / benchmarks.loc[2016, 'BuildingSqm']
        pb_kwh_sqm = benchmarks.loc[:, 'pop'].mul(pb_kwh)
        benchmarks.insert(1, 'BuildingKwh', pb_kwh_sqm)
```

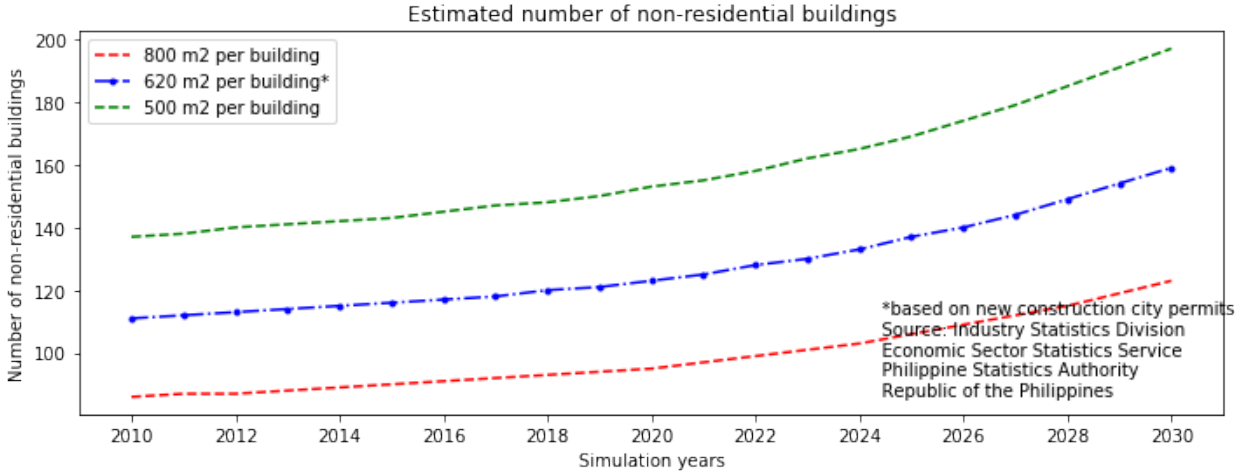
```
In [8]: benchmarks.loc[2016, 'NonRElectricity'] = kwh_2016
```

```
In [9]: benchmarks.head()
```

```
Out[9]: BuildingSqm   BuildingKwh    pop  NonRElectricity
        2010  68716.843636   9989.175844  111.0              NaN
        2011  69223.535896  10079.168419  112.0              NaN
        2012  69772.846840  10169.160994  113.0              NaN
        2013  70361.528759  10259.153569  114.0              NaN
        2014  71001.761591  10349.146145  115.0              NaN
```

```
In [10]: benchmarks.to_csv('data/benchmarks_nonresidential.csv')
```

```
In [11]: from smum._scripts.sqm_data import plot_nr
         plot_nr(pop_data_1, pop_data, pop_data_3, nr_data)
```



Prior non-residential model

```
In [12]: from smum._scripts.sqm_data import get_count_data
         import pandas as pd
         count_data = get_count_data()
         count_data = count_data.div(count_data.sum())
```

```
/usr/lib/python3.6/site-packages/pandas-0.22.0-py3.6-linux-x86_64.egg/pandas/util/_decorators.py:118
  return func(*args, **kwargs)
```

```
In [13]: nrb_elec = pd.DataFrame(columns=['co_mu', 'co_sd', 'p', 'dis', 'lb', 'ub'])

         nrb_elec.loc['BuildingSqm', 'co_mu'] = ",".join([str(i) for i in nr_data['sqm']])
         nrb_elec.loc['BuildingSqm', 'co_sd'] = ",".join([str(i) for i in nr_data['sqm_sd']])

         nrb_elec.loc['BuildingKwh', 'co_mu'] = ",".join([str(i) for i in nr_data['kwh']])
         nrb_elec.loc['BuildingKwh', 'co_sd'] = ",".join([str(i) for i in nr_data['kwh_sd']])

         nrb_elec.loc[:, 'p'] = ",".join([str(i) for i in count_data['counts']])
         nrb_elec.loc['BuildingSqm', 'dis'] = "Deterministic;n;Categorical"
         nrb_elec.loc['BuildingKwh', 'dis'] = "Deterministic;BuildingSqm;Categorical"
```

```
In [14]: nrb_elec.to_csv('data/table_elec_nr.csv')
```

```
In [15]: nrb_elec.loc['BuildingSqm', 'dis'] = "Normal;n;Categorical"
         nrb_elec.to_csv('data/test_elec_nr_normal.csv')
```

```
In [16]: nrb_elec
```

```
Out[16]: co_mu  \
         BuildingSqm  719.5871280224214,312.5947515174914,1165.99458...
         BuildingKwh                262,631,592,316,293,233,296,137,243

                                                                co_sd  \
         BuildingSqm  45.93150475011659,27.88881729659839,510.591052...
         BuildingKwh  135.8105297832241,649.5509987676103,344.818792...

                                                                   p  \
```

```
BuildingSqm  0.06084986413783436,0.09160606304100238,0.0106...
BuildingKwh  0.06084986413783436,0.09160606304100238,0.0106...

                                                    dis   lb   ub
BuildingSqm                    Normal;n;Categorical  NaN  NaN
BuildingKwh  Deterministic;BuildingSqm;Categorical  NaN  NaN
```

Sorsogon. Step 2.a Dynamic Sampling Model and GREGWT

```
In [1]: import datetime; print(datetime.datetime.now())
```

```
2018-03-26 01:28:43.554147
```

Notebook abstract

This notebook shows the main sampling and reweighting algorithm.

Import libraries

```
In [2]: from smum.microsim.run import run_calibrated_model
        from smum.microsim.table import TableModel
```

```
/usr/lib/python3.6/site-packages/h5py-2.7.1-py3.6-linux-x86_64.egg/h5py/__init__.py:36: FutureWarning
  from ._conv import register_converters as _register_converters
```

Global variables

```
In [3]: iterations = 1000
        benchmark_year = 2016
        census_file = 'data/benchmarks_year_bias.csv'
        typ = 'resampled'
        model_name = 'Sorsogon_Electricity_Water_wbias_projected_dynamic_{}'.format(typ)
        verbose = False
        #The number of chains to run in parallel.
        njobs = 4
```

Define Table model

```
In [4]: tm = TableModel(census_file = census_file, verbose=verbose)
```

Income model

```
In [5]: tm.add_model('data/table_inc.csv',    'Income')
        tm.update_dynamic_model('Income', specific_col = 'Education')
        tm.update_dynamic_model('Income',
                                specific_col = 'FamilySize',
                                specific_col_as = 'Size',
                                val = 'mu', compute_average =  0)
        tm.update_dynamic_model('Income',
                                specific_col = 'Age',
                                val = 'mu', compute_average =  0)
```

```
In [6]: tm.models['Income'].loc[2020]
```

```
Out[6]: co_mu  \
        i_Intercept                                                  NaN
        i_Sex                                                919.012059036333
        i_Urbanity                                          7105.2244566329355
        i_FamilySize                                         1666.846395220964
        i_Age                                               116.57589770606201
        i_Education   1.0,6023.86254599,11959.091528,18727.4606703,1...

                                                                      co_sd  \
        i_Intercept                                                     NaN
        i_Sex                                               161.50344091572538
        i_Urbanity                                          127.94148635675795
        i_FamilySize                                         29.03482607534048
        i_Age                                                   4.681393204635
        i_Education   1e-10,140.904404522,217.208790314,282.17614554...

                                                                         p       mu  \
        i_Intercept                                                1147.66      NaN
        i_Sex                                                     0.243795      NaN
        i_Urbanity                                                  0.6356      NaN
        i_FamilySize                                                   NaN  3.70878
        i_Age                                                          NaN  52.5153
        i_Education   0.2430379746835443,0.21581625995041107,0.25540...      NaN

                           sd            dis   ub   lb
        i_Intercept       NaN  Deterministic  NaN  NaN
        i_Sex             NaN      Bernoulli  NaN  NaN
        i_Urbanity        NaN      Bernoulli  NaN  NaN
        i_FamilySize  1.83794        Poisson   10    1
        i_Age         12.2451         Normal  100   18
        i_Education       NaN    Categorical  NaN  NaN
```

```
In [7]: formula_inc = "i_Intercept+"+"+".join(
            ["c_{0} * {0}".format(e) for e in tm.models['Income'][benchmark_year].index if\
                (e != 'i_Intercept')])
        tm.add_formula(formula_inc, 'Income')
```

```
In [8]: tm.print_formula('Income')
```

```
Income =
         i_Intercept +
         c_i_Sex * i_Sex +
         c_i_Urbanity * i_Urbanity +
         c_i_FamilySize * i_FamilySize +
         c_i_Age * i_Age +
         c_i_Education * i_Education +
```

Electricity model

```
In [9]: tm.add_model('data/table_elec.csv',  'Electricity', reference_cat = ['yes'])
        tm.update_dynamic_model('Electricity', specific_col = 'Income', val = 'mu', compute_average =
```

```
In [10]: tm.models['Electricity'].loc[2016]
```

```
Out[10]: co_mu        co_sd          p      mu       sd  \
         e_Intercept             NaN          NaN    3.29998     NaN      NaN
         e_Lighting         0.825662      18.6676   0.946022     NaN      NaN
         e_TV                18.7899      1.75962   0.964932     NaN      NaN
         e_Cooking           28.8862      1.96894  0.0142662     NaN      NaN
         e_Refrigeration     59.2432      1.55605   0.602102     NaN      NaN
```

```
e_AC                 203.323      3.13016   0.256521     NaN      NaN
e_Urban              24.5935      1.39104          1     NaN      NaN
e_Income          0.00142607  4.10201e-05        NaN  190472  1904.72

                              dis   ub   lb
e_Intercept       Deterministic  NaN  NaN
e_Lighting            Bernoulli  NaN  NaN
e_TV                  Bernoulli  NaN  NaN
e_Cooking             Bernoulli  NaN  NaN
e_Refrigeration       Bernoulli  NaN  NaN
e_AC                  Bernoulli  NaN  NaN
e_Urban               Bernoulli  NaN  NaN
e_Income                   None  inf    0
```

```
In [11]: formula_elec = "e_Intercept+"+"+".join(
             ["c_{0} * {0}".format(e) for e in tm.models['Electricity'][benchmark_year].index if\
                 (e != 'e_Intercept') &\
                 (e != 'e_Income') &\
                 (e != 'e_Urban')
             ])
         formula_elec += '+c_e_Urban * i_Urbanity'
         formula_elec += '+c_e_{0} * {0}'.format('Income')
```

```
In [12]: tm.add_formula(formula_elec, 'Electricity')
```

```
In [13]: tm.print_formula('Electricity')
```

```
Electricity =
         e_Intercept +
         c_e_Lighting * e_Lighting +
         c_e_TV * e_TV +
         c_e_Cooking * e_Cooking +
         c_e_Refrigeration * e_Refrigeration +
         c_e_AC * e_AC +
         c_e_Urban * i_Urbanity +
         c_e_Income * Income +
```

Water model

```
In [14]: tm.add_model('data/table_water.csv', 'Water')
         tm.update_dynamic_model('Water', specific_col = 'Education')
         tm.update_dynamic_model('Water',
                                 specific_col = 'FamilySize',
                                 specific_col_as = 'Size',
                                 val = 'mu', compute_average =  0)
         tm.update_dynamic_model('Water',
                                 specific_col = 'Age',
                                 val = 'mu', compute_average =  0)
```

```
In [15]: tm.models['Water'].loc[2020]
```

```
Out[15]: co_mu  \
         w_Intercept                                                          NaN
         w_Sex                                                  98.49504620801835
         w_Urbanity                                            1000.9789077676428
         w_Total_Family_Income                                0.05318701200857999
         w_FamilySize                                           49.73935151831777
         w_Age                                                  6.088941881654669
         w_Education           1.0,214.4011453125436,260.32727427717964,101.7...
```

```
                                                                    co_sd  \
w_Intercept                                                           NaN
w_Sex                                                   29.44380722589748
w_Urbanity                                              25.415910606032206
w_Total_Family_Income                                 0.0009823058551951082
w_FamilySize                                             5.897790558149098
w_Age                                                   0.9127405886772298
w_Education          1e-10,28.815802440470176,40.0574490885231,49.9...

                                                                       p  \
w_Intercept                                                     -601.592
w_Sex                                                           0.243795
w_Urbanity                                                        0.6356
w_Total_Family_Income                                                NaN
w_FamilySize                                                         NaN
w_Age                                                                NaN
w_Education          0.2430379746835443,0.21581625995041107,0.25540...

                                        dis       mu       sd   ub   lb
w_Intercept                  Deterministic      NaN      NaN  NaN  NaN
w_Sex                                 None      NaN      NaN  NaN  NaN
w_Urbanity                            None      NaN      NaN  NaN  NaN
w_Total_Family_Income                 None      NaN      NaN  NaN  NaN
w_FamilySize                          None  3.70878  1.83794  NaN  NaN
w_Age                                 None  52.5153  12.2451  NaN  NaN
w_Education          None;i;Categorical       NaN      NaN  NaN  NaN
```

```
In [16]: formula_water = "w_Intercept+"+"+".join(
             ["c_{0} * {1}".format(e, "i_"+"_".join(e.split('_')[1:]))\
                  for e in tm.models['Water'][benchmark_year].index if \
                                       (e != 'w_Intercept') &\
                                       (e != 'w_Total_Family_Income')   &\
                                       (e != 'w_Education')
             ])
         formula_water += '+c_w_Total_Family_Income*Income'
         formula_water += '+c_w_Education*i_Education'
```

```
In [17]: tm.add_formula(formula_water, 'Water')
```

```
In [18]: tm.print_formula('Water')
```

```
Water =
         w_Intercept +
         c_w_Sex * i_Sex +
         c_w_Urbanity * i_Urbanity +
         c_w_FamilySize * i_FamilySize +
         c_w_Age * i_Age +
         c_w_Total_Family_Income*Income +
         c_w_Education*i_Education +
```

Make model and save it to excel

```
In [19]: table_model = tm.make_model()
```

```
In [20]: tm.to_excel()
```

```
creating data/tableModel_Income.xlsx
creating data/tableModel_Electricity.xlsx
creating data/tableModel_Water.xlsx
```

Define model variables

```
In [21]: labels = ['age_0_18', 'age_19_25', 'age_26_35',
                   'age_36_45', 'age_46_55', 'age_56_65',
                   'age_66_75', 'age_76_85', 'age_86_100']
         cut = [0, 19, 26, 36, 46, 56, 66, 76, 86, 101]
         to_cat = {'i_Age':[cut, labels]}
         drop_col_survey = ['e_Income', 'e_Urban', 'w_Total_Family_Income', 'w_Education']
```

```
In [ ]: fw = run_calibrated_model(
            table_model,
            project = typ,
            njobs = njobs,
            #rep = {'FamilySize': ['Size']},
            #rep={'urb': ['urban', 'urbanity']},
            census_file = census_file,
            year = benchmark_year,
            population_size = False,
            name = '{}_{}'.format(model_name, iterations),
            to_cat = to_cat,
            iterations = iterations,
            verbose = verbose,
            drop_col_survey = drop_col_survey)
```

Sorsogon. Step 2.b Non-Residential Model

```
In [1]: import datetime; print(datetime.datetime.now())
```

```
2018-03-26 01:53:46.635614
```

Notebook abstract

This notebook shows the main sampling and reweighting algorithm for the non-residential sector.

Import libraries

```
In [2]: from smum.microsim.util_plot import plot_data_projection
        from smum.microsim.run import run_calibrated_model
        from smum.microsim.table import TableModel
```

Global variables

```
In [3]: iterations = 1000
        benchmark_year = 2016
        census_file = 'data/benchmarks_nonresidential.csv'
        typ = 'resampled'
        model_name = 'Sorsogon_NonResidentialElectricity_wbias_projected_dynamic_{}'.format(typ)
        verbose = False
        drop_col_survey = ['n_BuildingKwh']
```

Define model

```
In [4]: table_model_name = 'data/table_elec_nr.csv'
        estimate_var = 'NonRElectricity'
        tm = TableModel(census_file = census_file, verbose=verbose)
```

```
        tm.add_model(table_model_name, estimate_var, static = True)
        #tm.update_dynamic_model(estimate_var, specific_col = 'BuildingKwh', static=True)
```

In [5]:
```
tm.models[estimate_var].loc[2020]
```

Out[5]:
```
                                                                co_mu  \
n_BuildingSqm  719.5871280224214,312.5947515174914,1165.99458...
n_BuildingKwh                262,631,592,316,293,233,296,137,243

                                                                co_sd  \
n_BuildingSqm  45.93150475011659,27.88881729659839,510.591052...
n_BuildingKwh  135.8105297832241,649.5509987676103,344.818792...

                                                                p  \
n_BuildingSqm  0.06084986413783436,0.09160606304100238,0.0106...
n_BuildingKwh  0.06084986413783436,0.09160606304100238,0.0106...

                                                    dis   lb   ub
n_BuildingSqm            Deterministic;n;Categorical  NaN  NaN
n_BuildingKwh  Deterministic;BuildingSqm;Categorical  NaN  NaN
```

In [6]:
```
formula_nrb = " + ".join(["c_{0} * c_{0}".format(i) for i in tm.models[estimate_var].loc[201(
tm.add_formula(formula_nrb, estimate_var)
table_model = tm.make_model()
tm.to_excel()
```

```
creating data/tableModel_NonRElectricity.xlsx
```

In [7]:
```
formula_nrb
```

Out[7]:
```
'c_n_BuildingSqm * c_n_BuildingSqm + c_n_BuildingKwh * c_n_BuildingKwh'
```

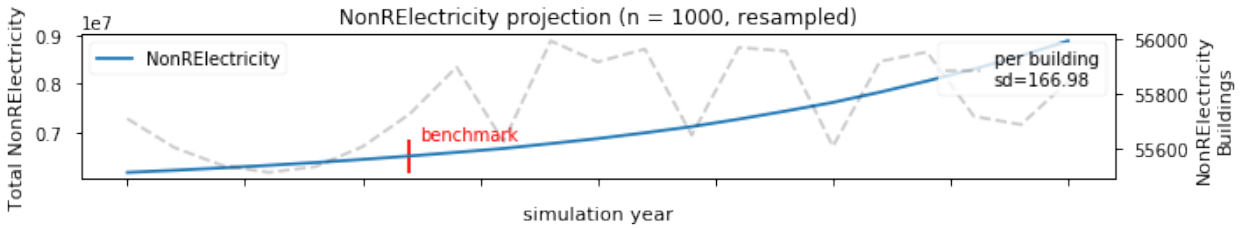
Run model

In [ ]:
```
fw = run_calibrated_model(
    table_model,
    verbose = verbose,
    project = typ,
    census_file = census_file,
    year = benchmark_year,
    population_size = False,
    name = '{}_{}'.format(model_name, iterations),
    iterations = iterations,
    align_census = False
    #drop_col_survey = drop_col_survey
)
```

Plot results

In [9]:
```
reweighted_survey = 'data/survey_{}_{}'.format(model_name, iterations)
```

In [10]:
```
data = plot_data_projection(
    reweighted_survey, [estimate_var], "{}, {}".format(iterations, typ),
    benchmark_year = benchmark_year, unit = "building")
```



Sorsogon. Step 2.c Model Internal Validation

```
In [1]: import datetime; print(datetime.datetime.now())
```

```
2018-03-26 01:55:54.033876
```

Notebook abstract

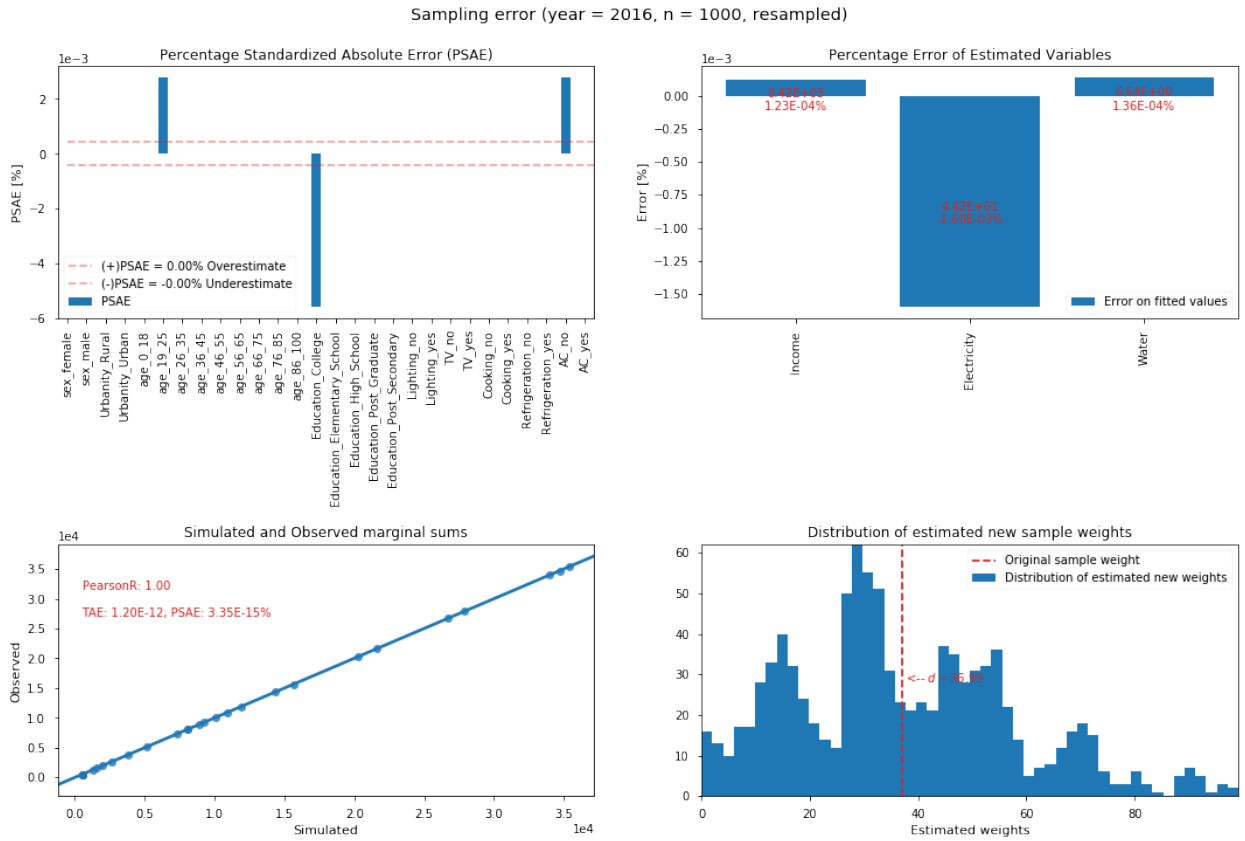
Model internal validation.

```
In [2]: from smum.microsim.util_plot import plot_error
```

Residential Sector

```
In [3]: iterations = 1000
        benchmark_year = 2016
        census_file = 'data/benchmarks_year_bias.csv'
        typ = 'resampled'
        model_name = 'Sorsogon_Electricity_Water_wbias_projected_dynamic_{}'.format(typ)
        survey_file = 'data/survey_{}_{}_{}.csv'.format(model_name, iterations, benchmark_year)
```

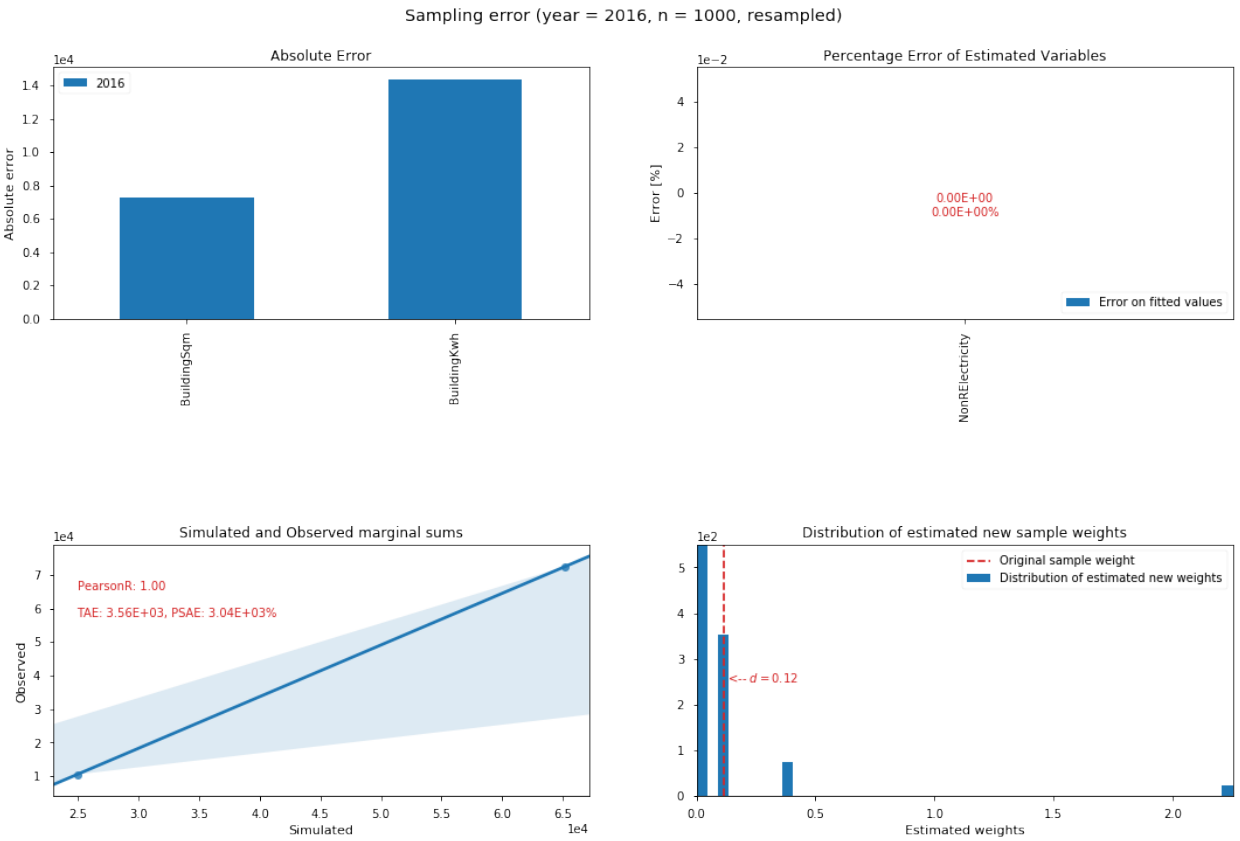
```
In [4]: REC = plot_error(
            survey_file,
            census_file,
            "{}, {}".format(iterations, typ),
            year = benchmark_year)
```



Non-Residential Sector

```
In [5]: iterations = 1000
        benchmark_year = 2016
        census_file = 'data/benchmarks_nonresidential.csv'
        typ = 'resampled'
        model_name = 'Sorsogon_NonResidentialElectricity_wbias_projected_dynamic_{}'.format(typ)
        survey_file = 'data/survey_{}_{}_{}.csv'.format(model_name, iterations, benchmark_year)
```

```
In [6]: REC = plot_error(
            survey_file,
            census_file,
            "{}, {}".format(iterations, typ),
            fit_cols = ['NonRElectricity'],
            verbose = True,
            #save_all = True,
            is_categorical = False,
            year = benchmark_year)
```



Sorsogon. Step 3.a Defining simple transition scenarios

```
In [1]: import datetime; print(datetime.datetime.now())
```

```
2018-03-26 01:59:08.226709
```

Notebook Abstract:

The following notebook describes the process to construct simple **transition scenarios**.

The transition scenarios are define as **efficiency** rates induced by **technology development** or **behavioral** changes. These rates can be used as proxies for all types of efficiency improvements.

In order to define transition scenarios the model need the following information:

1. A technology **penetration rate**. This defines the share of the population *adopting* the technology. The model uses `sampling rules` for the selection of the population adopting this technology.
2. Development of **efficiency rates**. This define the actual technology development rate.

Import libraries

```
In [2]: from smum.microsim.run import transition_rate
        from smum.microsim.util_plot import plot_transition_rate
        from smum.microsim.run import reduce_consumption
```

```
/usr/lib/python3.6/site-packages/h5py-2.7.1-py3.6-linux-x86_64.egg/h5py/__init__.py:36: FutureWarning
  from ._conv import register_converters as _register_converters
```

In order to compute the transition scenarios we make use of three modules of the `urbanmetabolism` library:

1. `growth_rate`. This module will return a vector with linear **growth rates** given a starting and end rate.
2. `plot_growth_rate`. This a simple function to **visualize** the defined growth rates.
3. `reduce_consumption`. This function creates **new samples** with reduced consumption levels for the selected selection of the population.

Define simple population selection rules

```
In [3]: sampling_rules = {
            #"i_Education == 'Education_Elementary_School'": 100,
            "i_Education == 'Education_Post_Secondary'": 20,
            "i_Education == 'Education_College'":         20,
            "i_Education == 'Education_Post_Graduate'":   20,
            "i_Urbanity == 'Urbanity_Urban'":             30,
            "Income >= 180000":                           30
        }
```

Part of the scenario development is to identified which section of the population will adopt the new technology. The model defined this by a **sampling probability**. This probability is initially define as a uniform distribution (i.e. each individual on the sample has equal probability of being selected). A scenario is defined by allocating new sampling probabilities to a section of the population, by defining sampling rules. The sampling rules are passes to the query function of a pandas `DataFrame`.

On the example above, all individuals with a `Income` larger of equal to 180 000 Philippine Pesos are 30 times more likely to adopt the technology that the rest of the population. The sampling probabilities will sum up. This means that an individual with `Income >= 180000` and living on an urban area `e_Urban == 'Urbanity_Urban'` is 60 times more likely to be selected (i.e. adopt a new technology) that other individuals.

Initial sample data

```
In [4]: import pandas as pd
        file_name = "data/survey_Sorsogon_Electricity_Water_wbias_projected_dynamic_resampled_1000_{
        sample_survey = pd.read_csv(file_name.format(2010), index_col=0)
        sample_survey.head()
```

```
Out[4]: index     i_Sex      i_Urbanity i_FamilySize      i_Age  \
        0      0  sex_male  Urbanity_Rural       Size_5  age_19_25
        1      1  sex_male  Urbanity_Rural       Size_2  age_56_65
        2      2  sex_male  Urbanity_Rural       Size_7  age_26_35
        3      3  sex_male  Urbanity_Urban       Size_4  age_26_35
        4      4  sex_male  Urbanity_Rural       Size_2  age_66_75

                          i_Education    e_Lighting    e_TV   e_Cooking  \
        0  Education_Elementary_School  Lighting_yes  TV_yes  Cooking_no
        1            Education_College  Lighting_yes   TV_no  Cooking_no
        2  Education_Elementary_School  Lighting_yes  TV_yes  Cooking_no
        3            Education_College  Lighting_yes   TV_no  Cooking_no
        4    Education_Post_Secondary  Lighting_yes   TV_no  Cooking_no

             e_Refrigeration   e_AC         Income  Electricity       Water  \
        0   Refrigeration_no  AC_no   60354.807334    17.356259   36.414496
        1   Refrigeration_no  AC_no  324476.009011    88.241617  217.607975
        2   Refrigeration_no  AC_no   80395.582383    24.303869   52.374996
        3   Refrigeration_no  AC_no  359650.342076   112.707133  247.005030
        4  Refrigeration_yes  AC_no  178130.514100    58.126489  126.879872
```

```
           w          wf
0  33.739938  37.150195
1  33.739938  36.127915
2  33.739938  51.494575
3  33.739938  46.852140
4  33.739938  27.914082
```

The `reduce_consumption` module will use as input the samples created by the MCMC algorithm and select specific sections of the population to reduce their consumption levels.

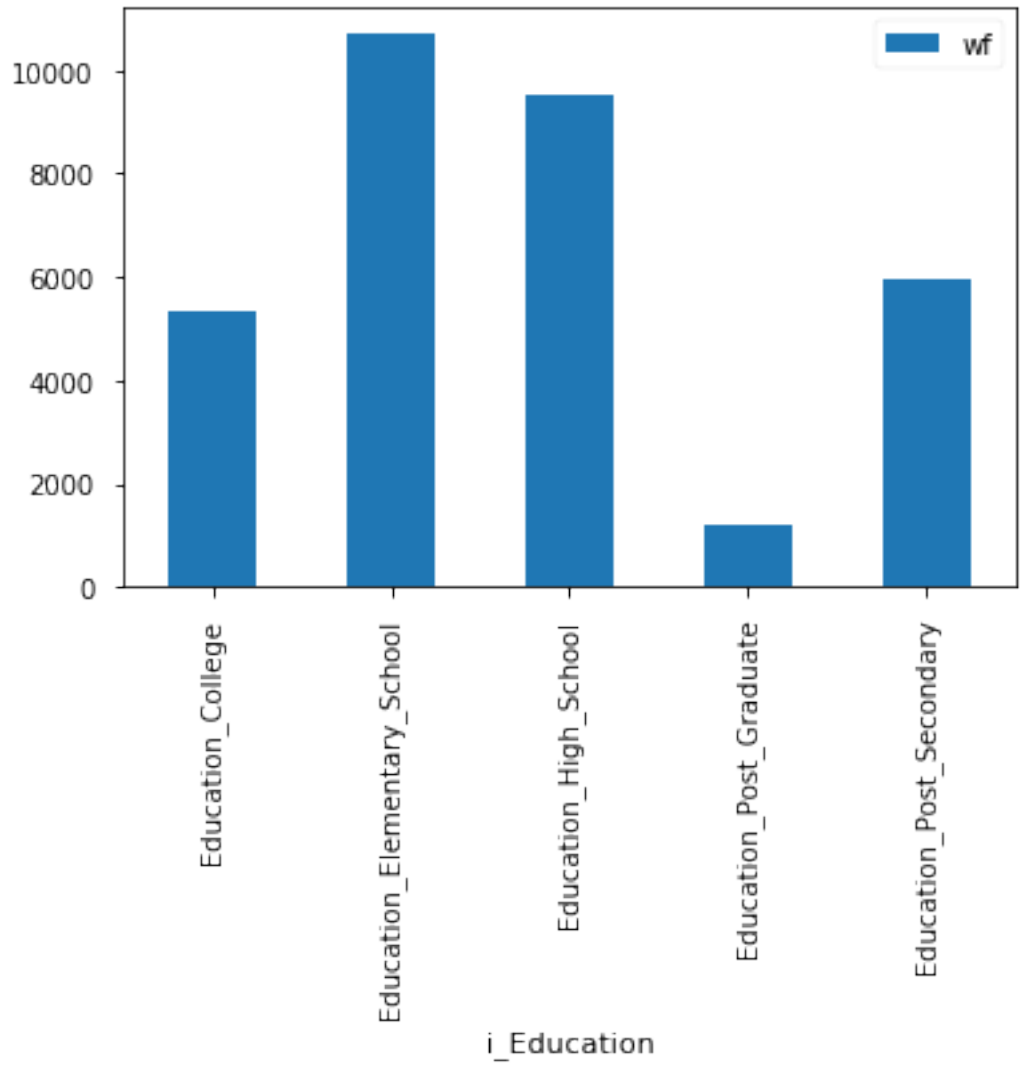
The input data is the constructed proxy sample data. Depending on the simulation type (reweight/resample) the input data is a single file containing the weights for each simulation year (reweight) or individual samples for each simulation year (resample).

Most of the variables of the sample data is formatted as categorical data. The sample data shown above presents individual records with the predefine simulation variables as well as the computed **Income**, **Electricity** and **Water** consumption levels. The sample also contains two sets of weights: **w** and **wf**. The `w` weights correspond to the weights assign by the MCMC algorithm (uniform distributed) while the `wf` (final weights) are the weights computed by the GREGWT algorithm. The `reduce_consumption` function will use the `wf` weights for the selection of individuals.

```
In [5]: sample_survey.loc[:, 'i_Education'].unique()
```

```
Out[5]: array(['Education_Elementary_School', 'Education_College',
               'Education_Post_Secondary', 'Education_High_School',
               'Education_Post_Graduate'], dtype=object)
```

```
In [6]: sample_survey.loc[:, ['i_Education', 'wf']].groupby('i_Education').sum().plot.bar();
```

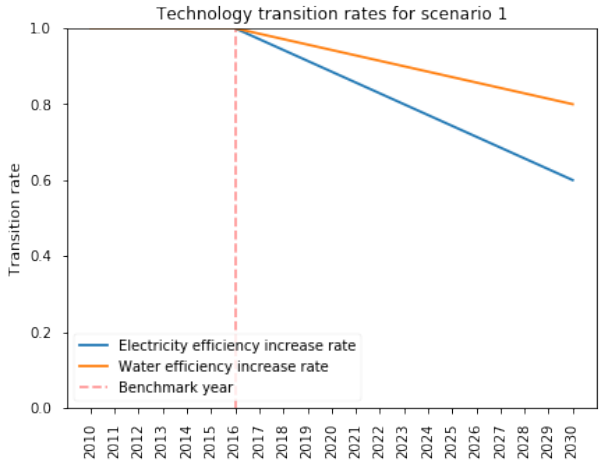


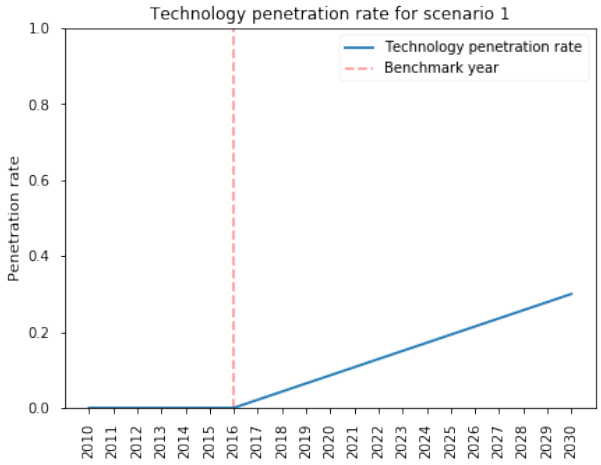
Define growth rates

```
In [7]: Elec  = transition_rate(0, 0.4, start=2016)
        Water = transition_rate(0, 0.2, start=2016)
        pr    = transition_rate(0, 0.3, start=2016)
```

With help of the `growth_rate()` function we define the **efficiency growth rate** and the technology penetration rate. For the technology penetration rate we define the start year to be equal to the benchmark year (2016). The function will automatically include the necessary zeros at the beginning of the growth rate vector. The function `plot_growth_rate()` allow us to **visualize** the predefined efficiency growth rates and the technology growth rates.

```
In [8]: plot_transition_rate(
            {"Technology penetration rate": pr,
             "Electricity efficiency increase rate": Elec,
             "Water efficiency increase rate": Water},
             "scenario 1")
```





Reduce consumption

The actual modifications on the sample is performed by the `reduce_consumption()` function. The function requires as input the following parameters:

1. A base file name for the samples (in case of implementing the resample method).
2. The sample year (in this case generated within the loop via `range(2010, 2031)`.
3. The penetration rate for the sample year (iterated from vector `pr`).
4. The predefined `sampling_rules`.
5. A dictionary containing the efficiency rates for specific variables. (in this case for Electricity and Water).
6. A name for the scenario.

```
In [9]: for y, p, elec, water in zip(range(2010, 2031), pr, Elec, Water):
            _ = reduce_consumption(
                file_name,
                y, p, sampling_rules,
                {'Electricity':elec, 'Water':water},
                scenario_name = "scenario 1")
```

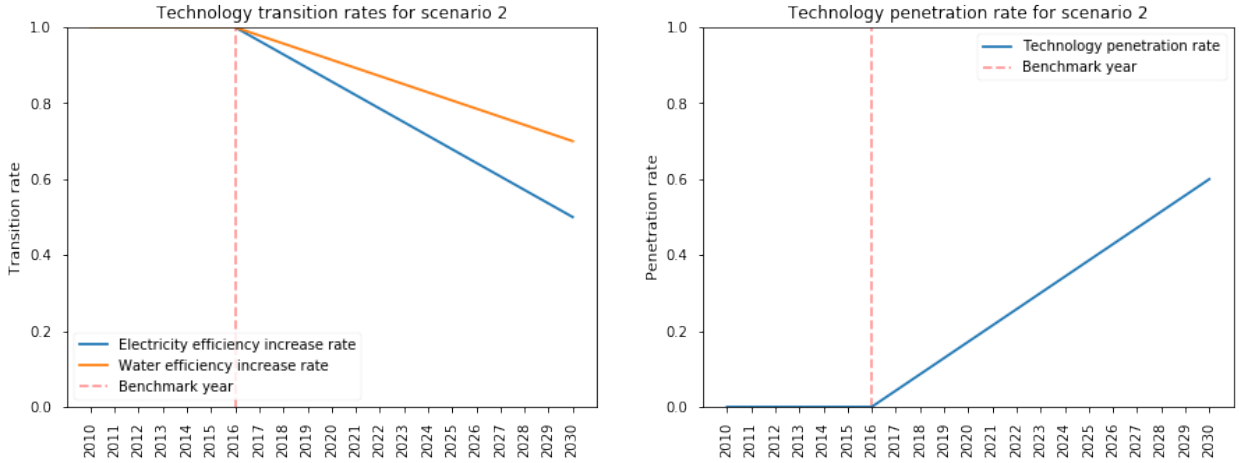
```
00.00%       both       reduction; efficiency rate 00.00%;            year 2010 and penetration rate
00.00%       both       reduction; efficiency rate 00.00%;            year 2011 and penetration rate
00.00%       both       reduction; efficiency rate 00.00%;            year 2012 and penetration rate
00.00%       both       reduction; efficiency rate 00.00%;            year 2013 and penetration rate
00.00%       both       reduction; efficiency rate 00.00%;            year 2014 and penetration rate
00.00%       both       reduction; efficiency rate 00.00%;            year 2015 and penetration rate
00.00%       both       reduction; efficiency rate 00.00%;            year 2016 and penetration rate
00.08%   Electricity    reduction; efficiency rate 02.86%;            year 2017 and penetration rate
00.04%      Water       reduction; efficiency rate 01.43%;            year 2017 and penetration rate
00.30%   Electricity    reduction; efficiency rate 05.71%;            year 2018 and penetration rate
00.16%      Water       reduction; efficiency rate 02.86%;            year 2018 and penetration rate
00.66%   Electricity    reduction; efficiency rate 08.57%;            year 2019 and penetration rate
00.36%      Water       reduction; efficiency rate 04.29%;            year 2019 and penetration rate
01.15%   Electricity    reduction; efficiency rate 11.43%;            year 2020 and penetration rate
00.63%      Water       reduction; efficiency rate 05.71%;            year 2020 and penetration rate
01.80%   Electricity    reduction; efficiency rate 14.29%;            year 2021 and penetration rate
00.97%      Water       reduction; efficiency rate 07.14%;            year 2021 and penetration rate
02.51%   Electricity    reduction; efficiency rate 17.14%;            year 2022 and penetration rate
01.38%      Water       reduction; efficiency rate 08.57%;            year 2022 and penetration rate
```

```
03.38%   Electricity   reduction; efficiency rate 20.00%;              year 2023 and penetration rate
01.87%      Water      reduction; efficiency rate 10.00%;              year 2023 and penetration rate
04.37%   Electricity   reduction; efficiency rate 22.86%;              year 2024 and penetration rate
02.37%      Water      reduction; efficiency rate 11.43%;              year 2024 and penetration rate
05.50%   Electricity   reduction; efficiency rate 25.71%;              year 2025 and penetration rate
03.00%      Water      reduction; efficiency rate 12.86%;              year 2025 and penetration rate
06.74%   Electricity   reduction; efficiency rate 28.57%;              year 2026 and penetration rate
03.64%      Water      reduction; efficiency rate 14.29%;              year 2026 and penetration rate
08.10%   Electricity   reduction; efficiency rate 31.43%;              year 2027 and penetration rate
04.36%      Water      reduction; efficiency rate 15.71%;              year 2027 and penetration rate
09.55%   Electricity   reduction; efficiency rate 34.29%;              year 2028 and penetration rate
05.09%      Water      reduction; efficiency rate 17.14%;              year 2028 and penetration rate
11.14%   Electricity   reduction; efficiency rate 37.14%;              year 2029 and penetration rate
05.92%      Water      reduction; efficiency rate 18.57%;              year 2029 and penetration rate
12.81%   Electricity   reduction; efficiency rate 40.00%;              year 2030 and penetration rate
06.76%      Water      reduction; efficiency rate 20.00%;              year 2030 and penetration rate
```

```
In [10]: Elec  = transition_rate(0.0, 0.5, start=2016)
         Water = transition_rate(0.0, 0.3, start=2016)
         pr    = transition_rate(0.0, 0.6, start=2016)
```

By modifying the growht rates we can create different scenarios.

```
In [11]: plot_transition_rate(
             {"Technology penetration rate": pr,
              "Electricity efficiency increase rate": Elec,
              "Water efficiency increase rate": Water},
              "scenario 2")
```




```
In [12]: for y, p, elec, water in zip(range(2010, 2031), pr, Elec, Water):
             _ = reduce_consumption(
                 file_name,
                 y, p, sampling_rules,
                 {'Electricity':elec, 'Water':water},
                 scenario_name = "scenario 2")
```

```
00.00%      both       reduction; efficiency rate 00.00%;              year 2010 and penetration rate
00.00%      both       reduction; efficiency rate 00.00%;              year 2011 and penetration rate
00.00%      both       reduction; efficiency rate 00.00%;              year 2012 and penetration rate
00.00%      both       reduction; efficiency rate 00.00%;              year 2013 and penetration rate
00.00%      both       reduction; efficiency rate 00.00%;              year 2014 and penetration rate
00.00%      both       reduction; efficiency rate 00.00%;              year 2015 and penetration rate
00.00%      both       reduction; efficiency rate 00.00%;              year 2016 and penetration rate
00.19%   Electricity   reduction; efficiency rate 03.57%;              year 2017 and penetration rate
```

```
00.13%      Water      reduction; efficiency rate 02.14%;              year 2017 and penetration rate
00.74%   Electricity   reduction; efficiency rate 07.14%;              year 2018 and penetration rate
00.48%      Water      reduction; efficiency rate 04.29%;              year 2018 and penetration rate
01.63%   Electricity   reduction; efficiency rate 10.71%;              year 2019 and penetration rate
01.07%      Water      reduction; efficiency rate 06.43%;              year 2019 and penetration rate
02.87%   Electricity   reduction; efficiency rate 14.29%;              year 2020 and penetration rate
01.87%      Water      reduction; efficiency rate 08.57%;              year 2020 and penetration rate
04.44%   Electricity   reduction; efficiency rate 17.86%;              year 2021 and penetration rate
02.88%      Water      reduction; efficiency rate 10.71%;              year 2021 and penetration rate
06.22%   Electricity   reduction; efficiency rate 21.43%;              year 2022 and penetration rate
04.06%      Water      reduction; efficiency rate 12.86%;              year 2022 and penetration rate
08.43%   Electricity   reduction; efficiency rate 25.00%;              year 2023 and penetration rate
05.55%      Water      reduction; efficiency rate 15.00%;              year 2023 and penetration rate
10.90%   Electricity   reduction; efficiency rate 28.57%;              year 2024 and penetration rate
07.08%      Water      reduction; efficiency rate 17.14%;              year 2024 and penetration rate
13.65%   Electricity   reduction; efficiency rate 32.14%;              year 2025 and penetration rate
08.88%      Water      reduction; efficiency rate 19.29%;              year 2025 and penetration rate
16.66%   Electricity   reduction; efficiency rate 35.71%;              year 2026 and penetration rate
10.71%      Water      reduction; efficiency rate 21.43%;              year 2026 and penetration rate
20.00%   Electricity   reduction; efficiency rate 39.29%;              year 2027 and penetration rate
12.80%      Water      reduction; efficiency rate 23.57%;              year 2027 and penetration rate
23.72%   Electricity   reduction; efficiency rate 42.86%;              year 2028 and penetration rate
15.10%      Water      reduction; efficiency rate 25.71%;              year 2028 and penetration rate
27.54%   Electricity   reduction; efficiency rate 46.43%;              year 2029 and penetration rate
17.42%      Water      reduction; efficiency rate 27.86%;              year 2029 and penetration rate
31.73%   Electricity   reduction; efficiency rate 50.00%;              year 2030 and penetration rate
19.95%      Water      reduction; efficiency rate 30.00%;              year 2030 and penetration rate
```

Sorsogon. Step 3.b. Visualize transition scenarios

```
In [1]: import datetime; print(datetime.datetime.now())
```

```
2018-04-09 11:47:05.326151
```

Notebook Abstract:

The following notebook **visualize** the the simple transition scenarios by plotting the total consumption over all simulation years and the per-capita consumption rate. Depending on the define scenarios the per-capita consumption rate can be maintained constant. The per-capita consumption value is computed as total consumption divided by population size.

Import libraries

```
In [2]: from smum.microsim.util_plot import plot_data_projection
```

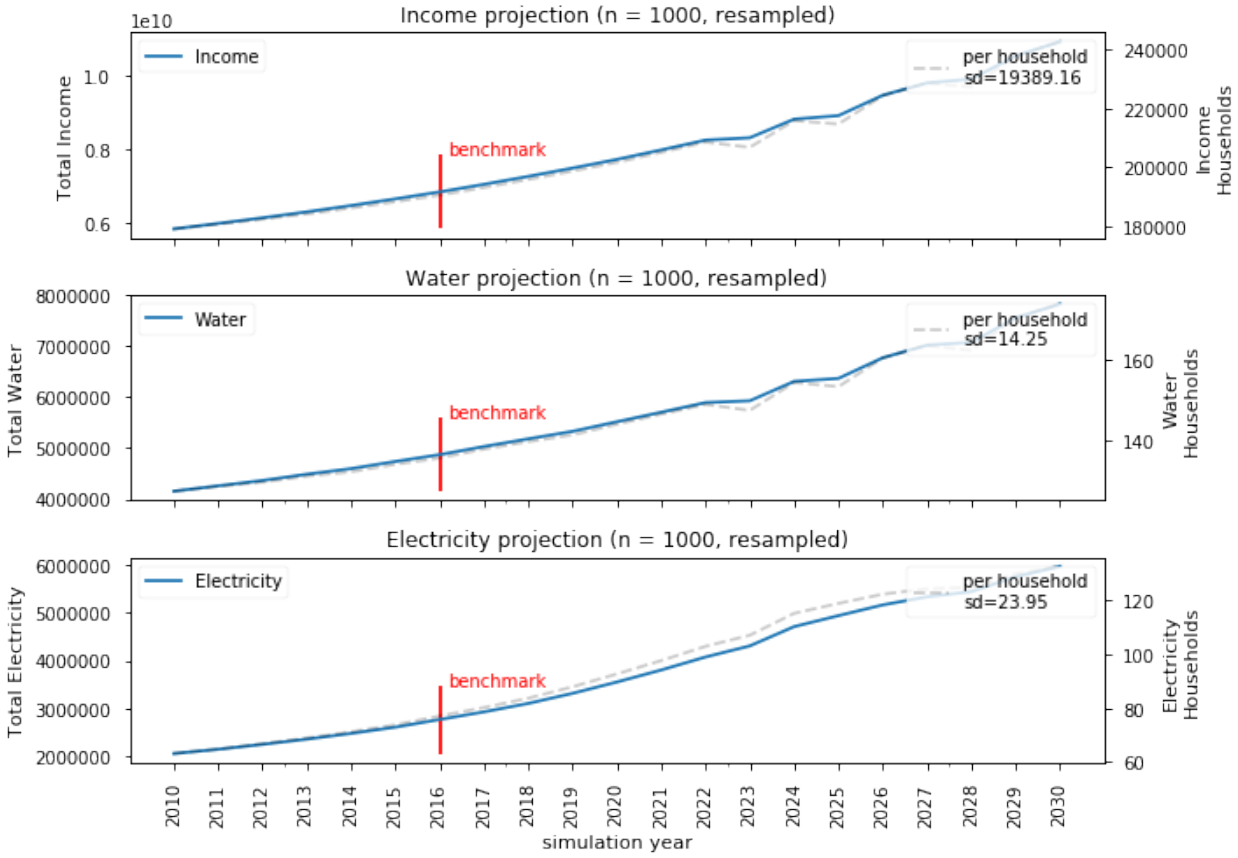
The visualization is performed with help of the module function `plot_data_projection()`.

Global variables

```
In [3]: iterations = 1000
        typ = 'resampled'
        model_name = 'Sorsogon_Electricity_Water_wbias_projected_dynamic_{}'.format(typ)
        reweighted_survey = 'data/survey_{}_{}'.format(model_name, iterations)
```

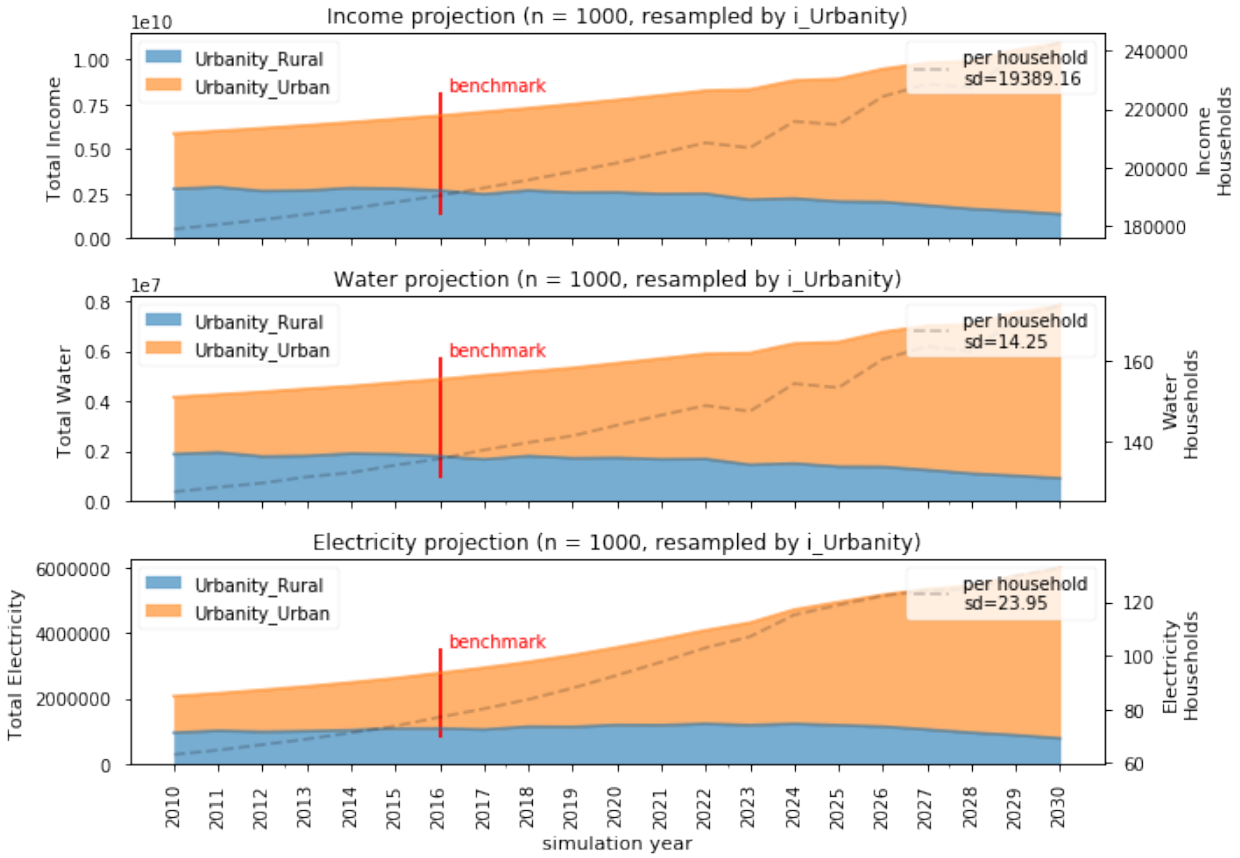
Base scenario

```
In [4]: var = ['Income', 'Water', 'Electricity']
        data = plot_data_projection(
            reweighted_survey, var, "{}, {}".format(iterations, typ),
            benchmark_year=2016,
        )
```



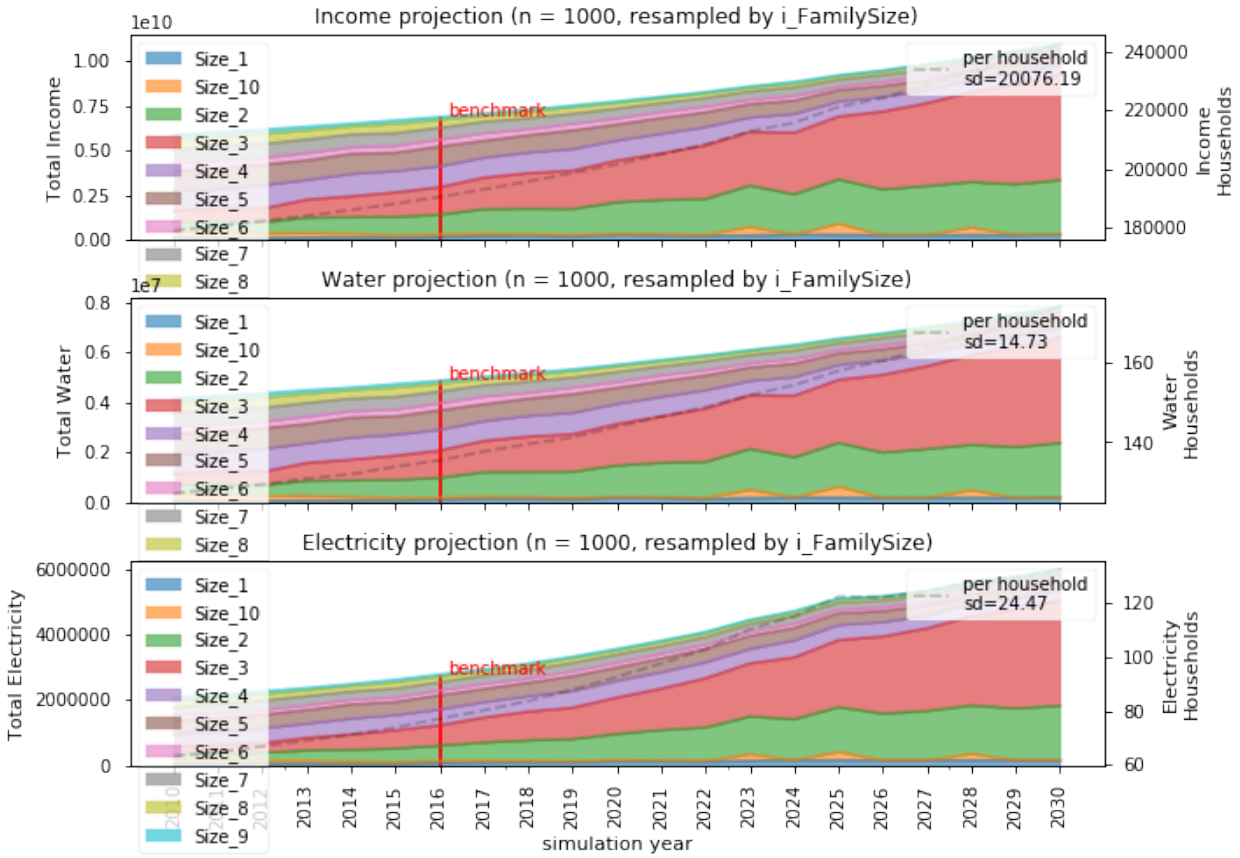
Base scenario grouped by Urban-Rural households

```
In [5]: var = ['Income', 'Water', 'Electricity']
        groupby = 'i_Urbanity'
        data = plot_data_projection(
            reweighted_survey, var, "{}, {} by {}".format(iterations, typ, groupby),
            benchmark_year = 2016,
            groupby = groupby
        )
```



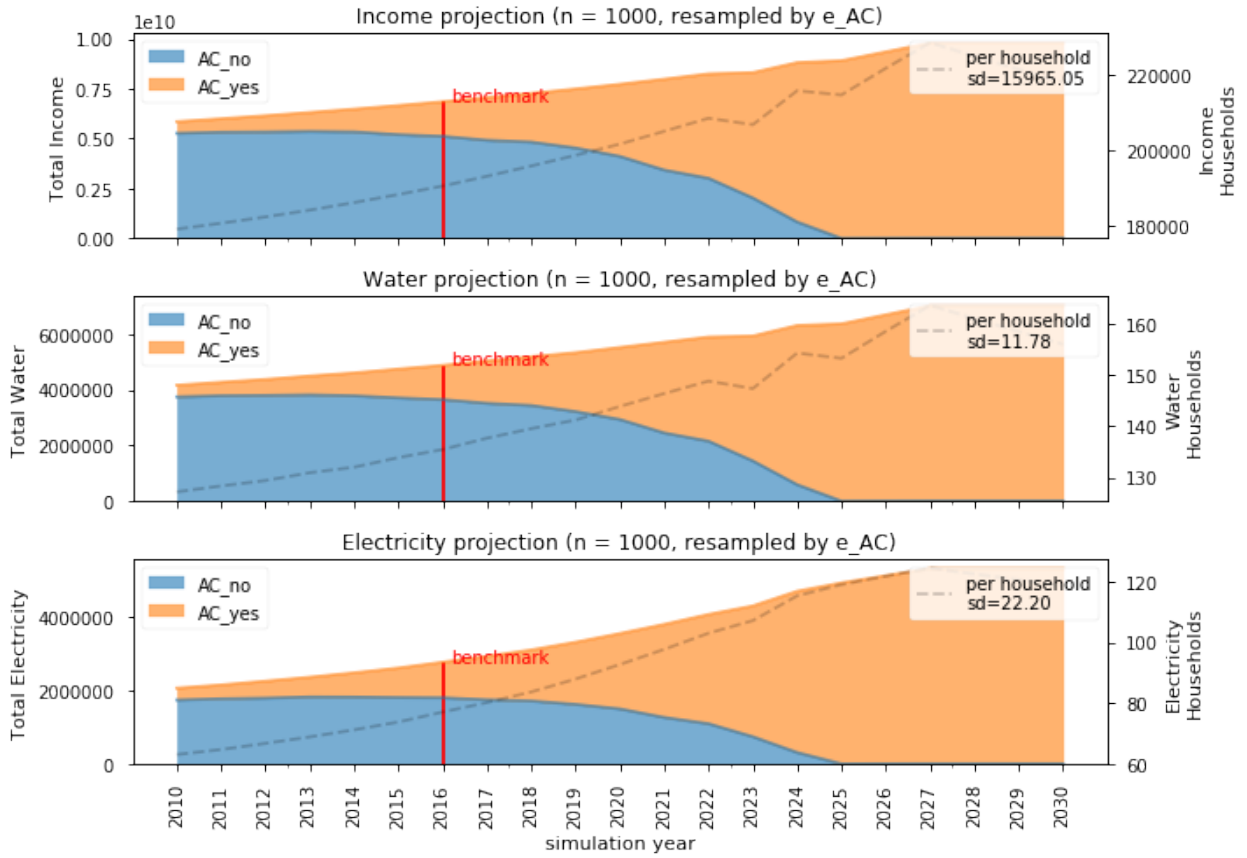
Base scenario grouped by family size

```
In [6]: var = ['Income', 'Water', 'Electricity']
        groupby = 'i_FamilySize'
        data = plot_data_projection(
            reweighted_survey, var, "{}, {} by {}".format(iterations, typ, groupby),
            benchmark_year = 2016,
            groupby = groupby
        )
```



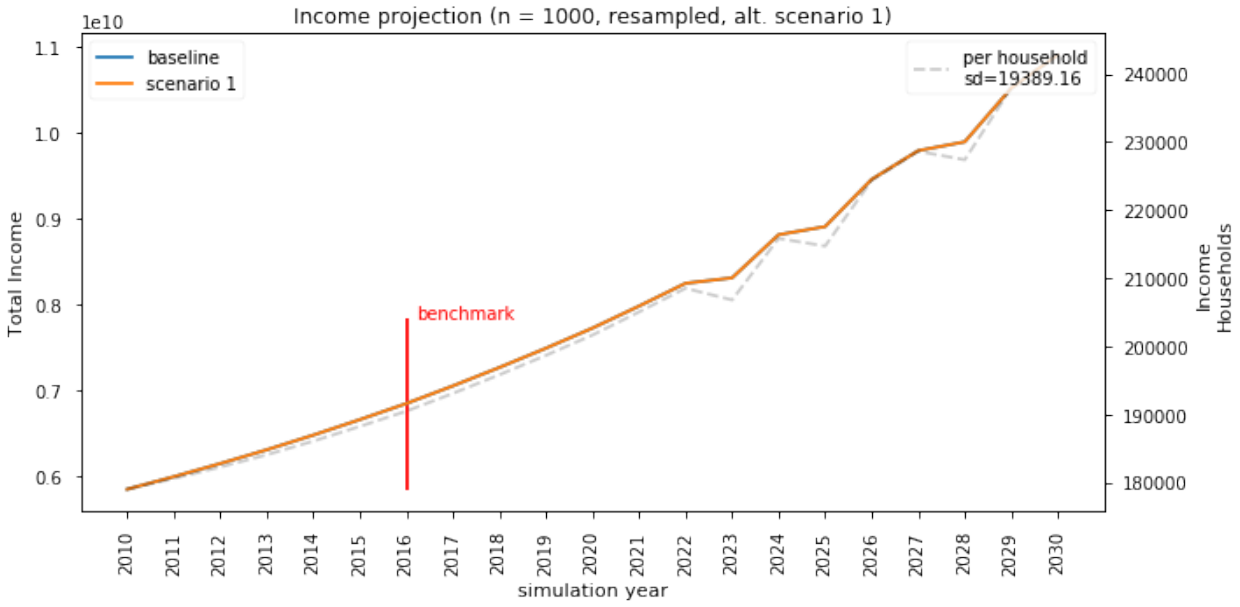
Base scenario grouped by AC ownership

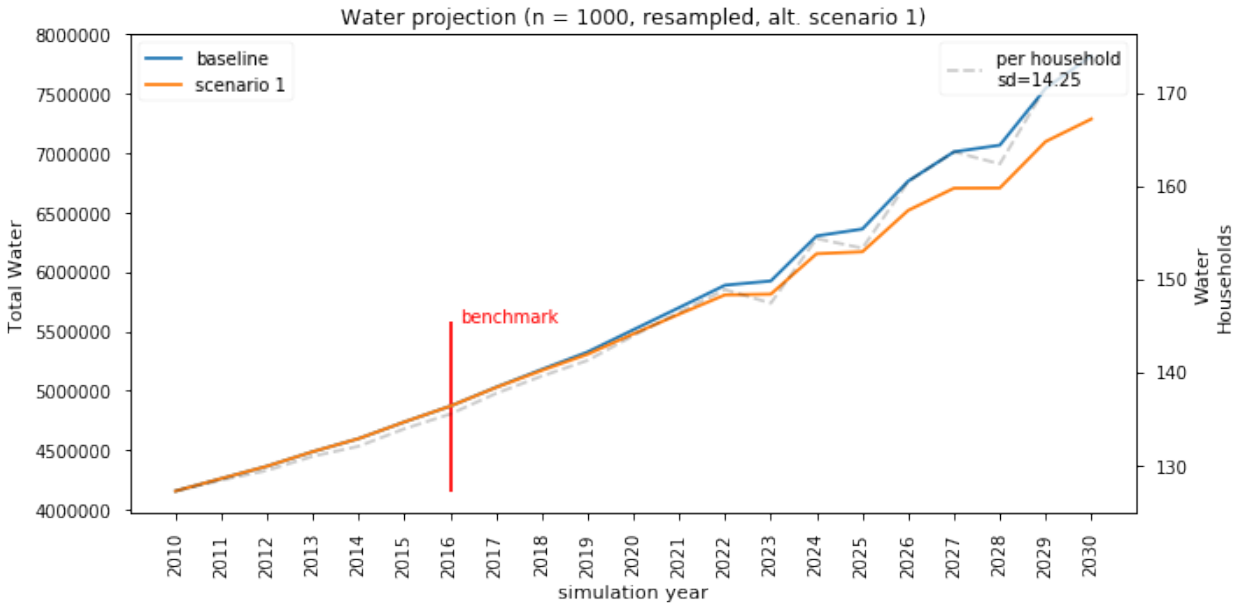
```
In [7]: var = ['Income', 'Water', 'Electricity']
        groupby = 'e_AC'
        data = plot_data_projection(
            reweighted_survey, var, "{}, {} by {}".format(iterations, typ, groupby),
            benchmark_year = 2016,
            verbose = False,
            groupby = groupby
        )
```

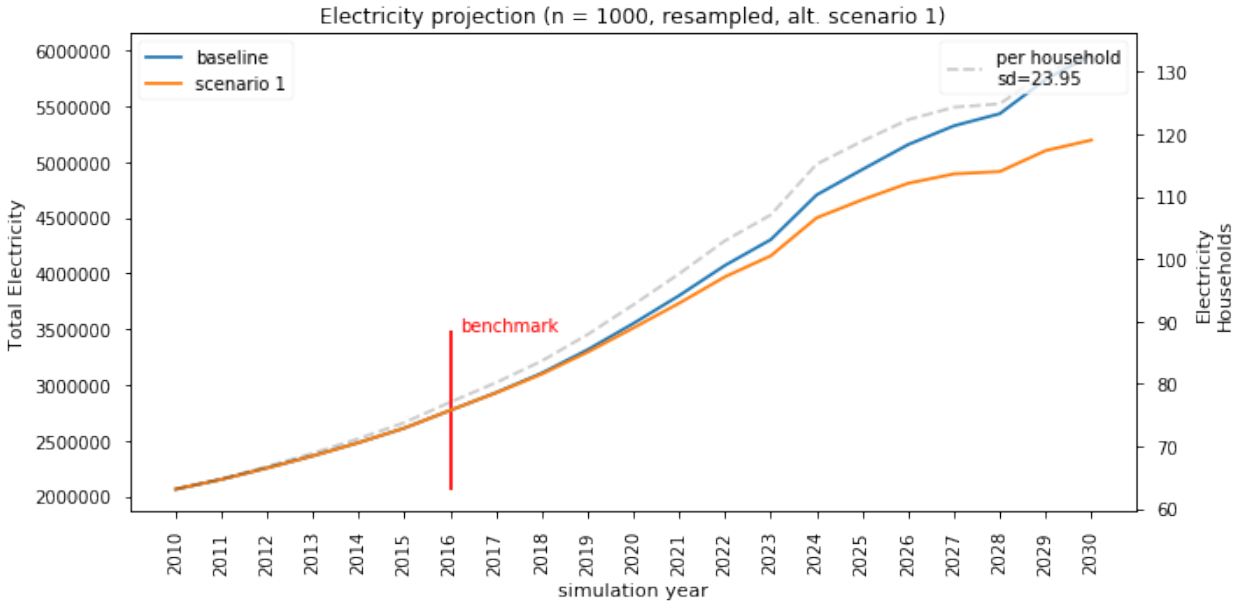


Scenario 1 compared to base scenario

```
In [8]: import numpy as np
        pr = [i for i in np.linspace(0, 0.3, num=15)]
        pr = [0]*6 + pr
        scenario_name = 'scenario 1'

In [9]: reweighted_survey

Out[9]: 'data/survey_Sorsogon_Electricity_Water_wbias_projected_dynamic_resampled_1000'

In [10]: variables = ['Income', 'Water', 'Electricity']
         for var in variables:
             var = [var]
             data = plot_data_projection(
                 reweighted_survey, var, "{}, {}, alt. scenario 1".format(iterations, typ),
                 benchmark_year=2016, pr = pr, scenario_name = scenario_name,
                 aspect_ratio = 2,
             )
```

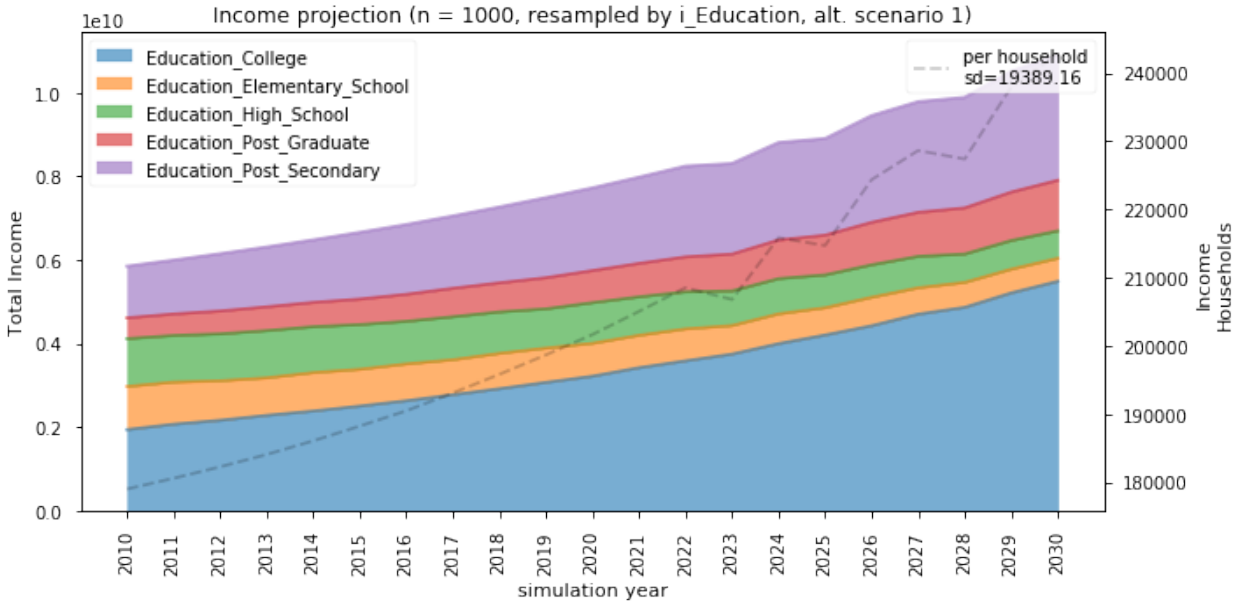

Income projection (n = 1000, resampled, alt. scenario 1)
1e10
baseline
scenario 1
per household
sd=19389.16
benchmark
Total Income
Income
Households
simulation year

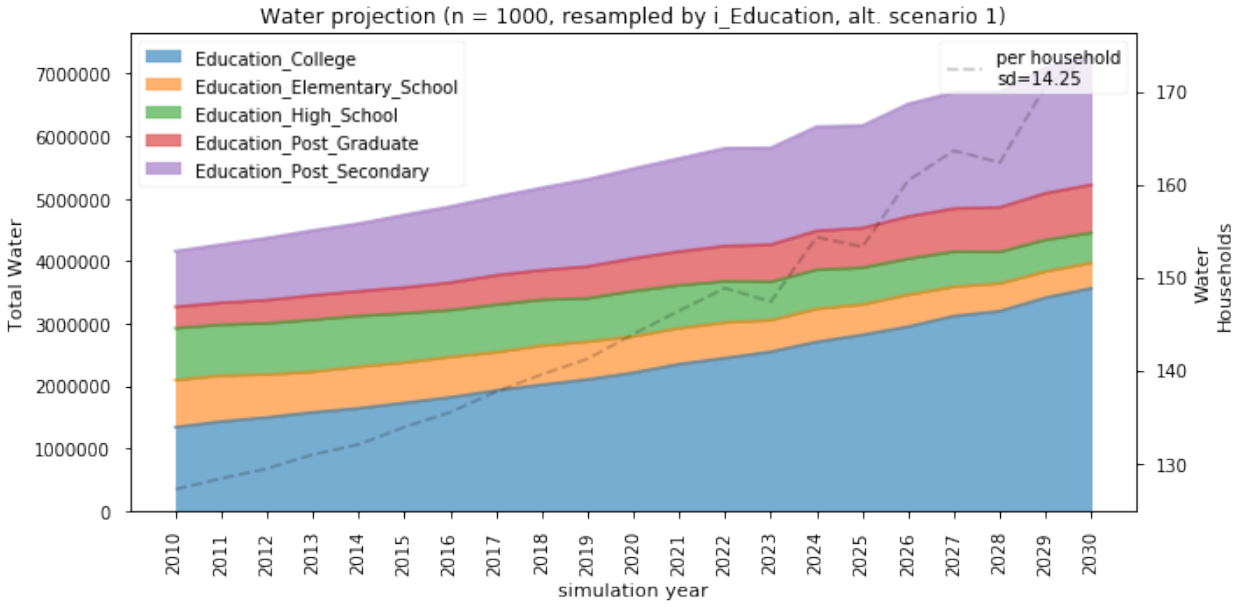

Water projection (n = 1000, resampled, alt. scenario 1)
baseline
scenario 1
per household
sd=14.25
benchmark
Total Water
Water
Households
simulation year

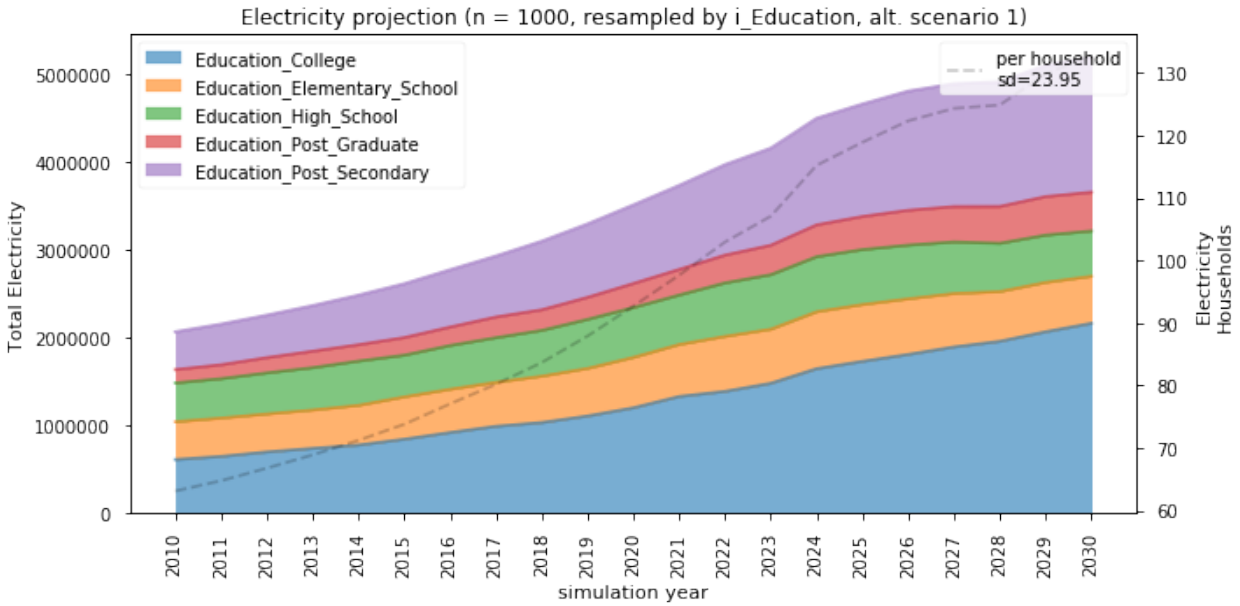


Scenario 1 grouped by education

```
In [11]: variables = ['Income', 'Water', 'Electricity']
         groupby = 'i_Education'
         for var in variables:
             var = [var]
             data = plot_data_projection(
                 reweighted_survey, var, "{}, {} by {}, alt. scenario 1".format(iterations, typ, grou
                 #benchmark_year=2016,
                 pr = pr, scenario_name = scenario_name,
                 groupby = groupby,
                 aspect_ratio = 2,
             )
```





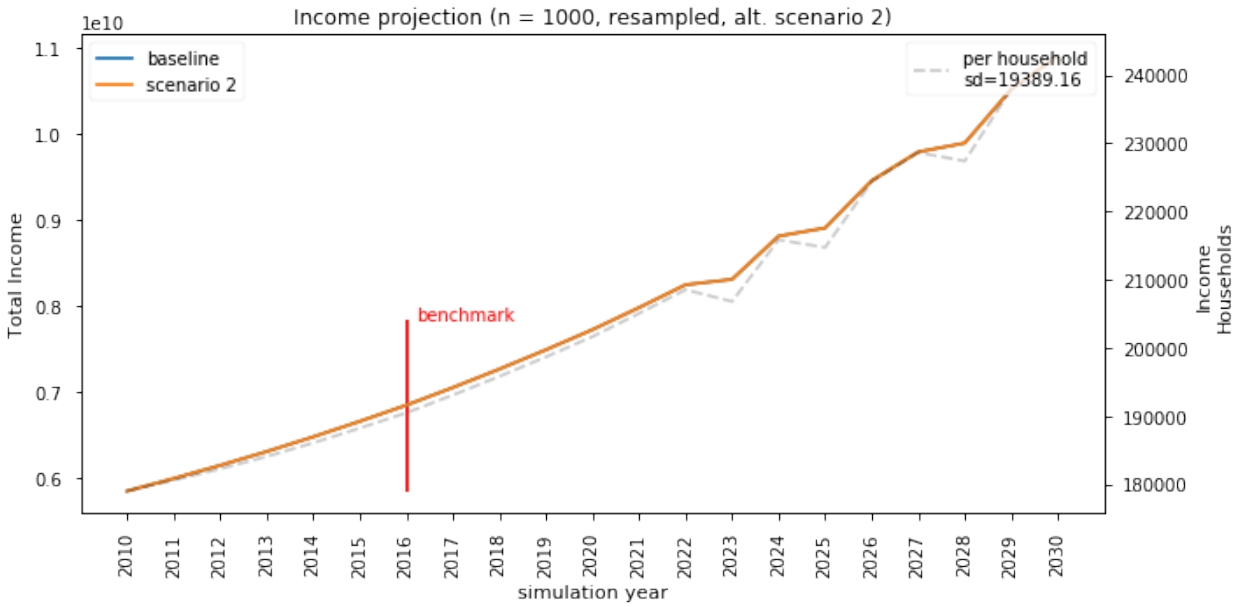


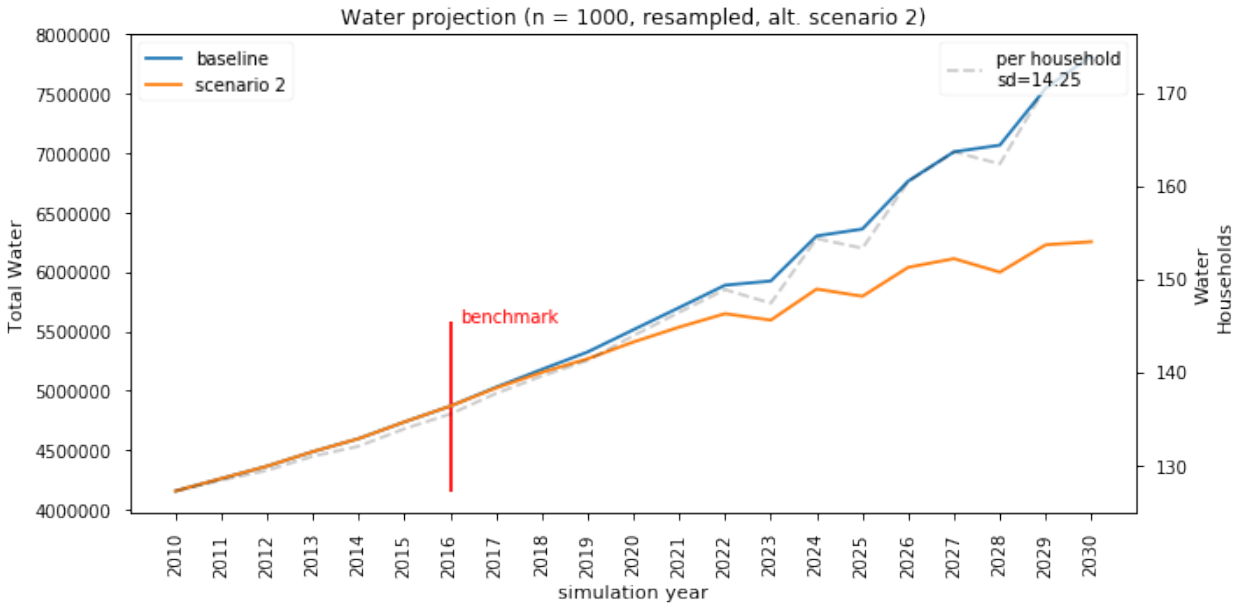

```
In [12]: import numpy as np
         pr = [i for i in np.linspace(0, 0.6, num=15)]
         pr = [0]*6 + pr
         scenario_name = 'scenario 2'
```

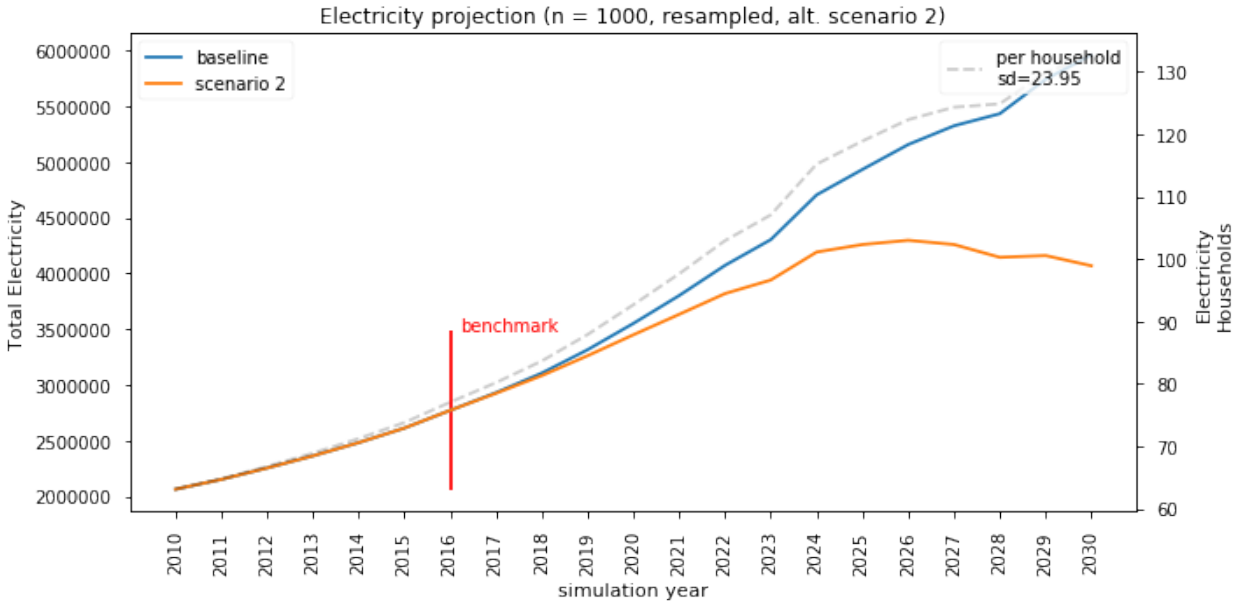
Scenario 2 compared to base scenario

```
In [13]: variables = ['Income', 'Water', 'Electricity']
         for var in variables:
             var = [var]
             data = plot_data_projection(
                 reweighted_survey, var, "{}, {}, alt. scenario 2".format(iterations, typ),
                 benchmark_year=2016, pr = pr, scenario_name = scenario_name,
                 aspect_ratio = 2,
```

)

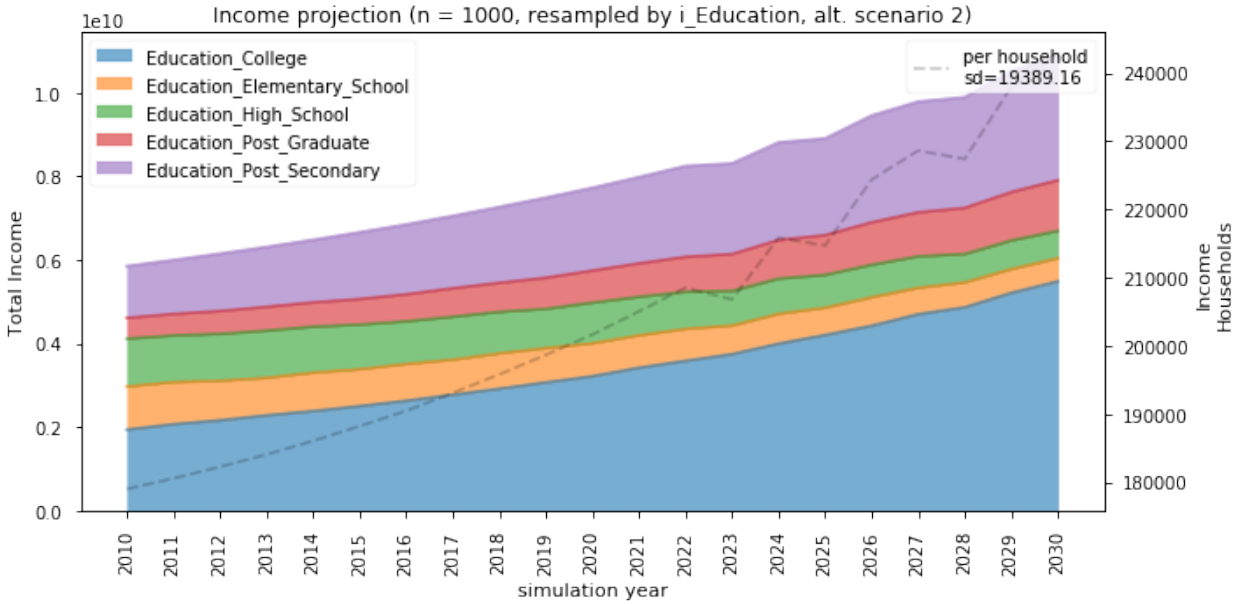

Income projection (n = 1000, resampled, alt. scenario 2)
1e10
baseline
scenario 2
per household
sd=19389.16
benchmark
Total Income
Income Households
simulation year

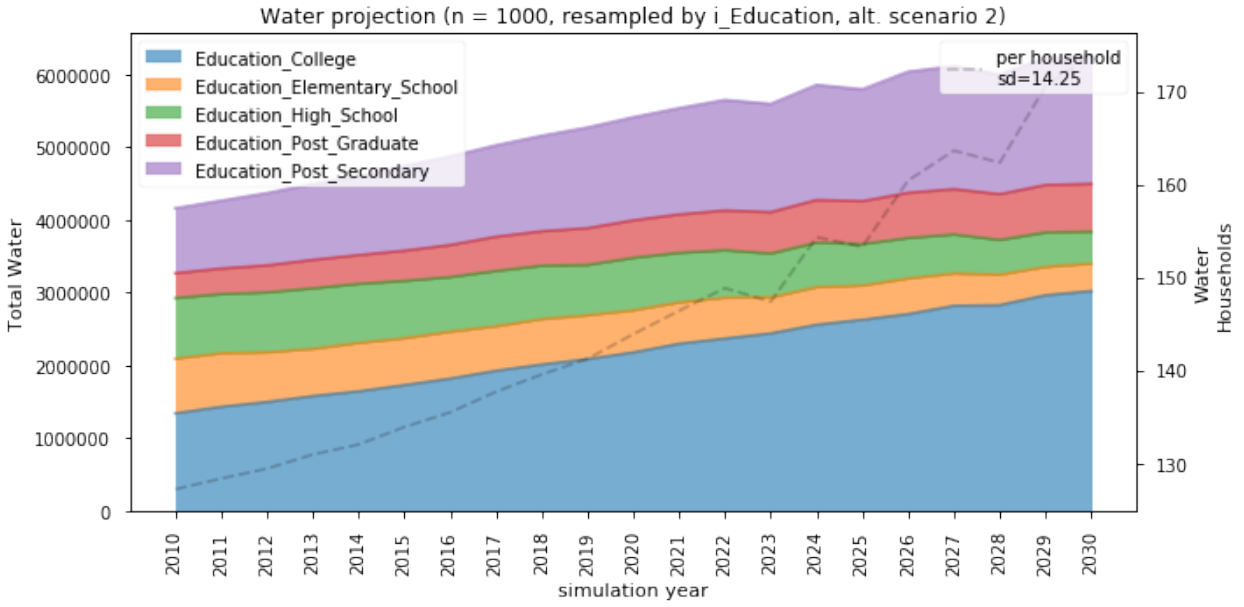

Water projection (n = 1000, resampled, alt. scenario 2)
baseline
scenario 2
per household
sd=14.25
benchmark
Total Water
Water Households
simulation year

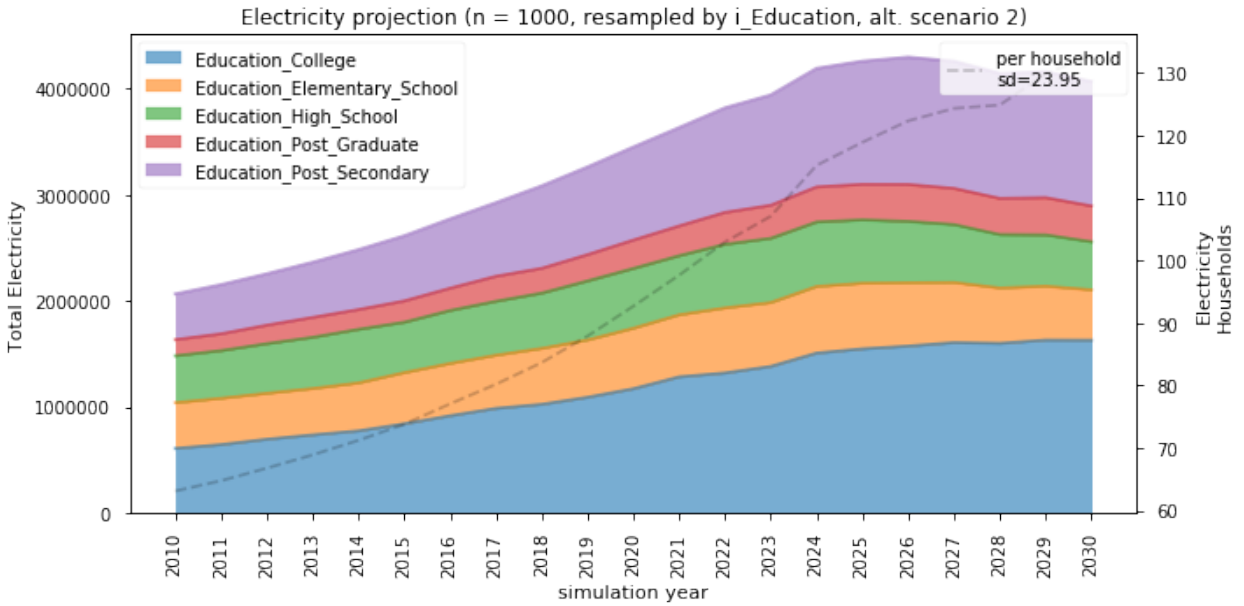


Scenario 2 grouped by education

```
In [14]: variables = ['Income', 'Water', 'Electricity']
         groupby = 'i_Education'
         for var in variables:
             var = [var]
             data = plot_data_projection(
                 reweighted_survey, var, "{}, {} by {}, alt. scenario 2".format(iterations, typ, grou
                 #benchmark_year=2016,
                 pr = pr, scenario_name = scenario_name,
                 groupby = groupby,
                 aspect_ratio = 2,
             )
```





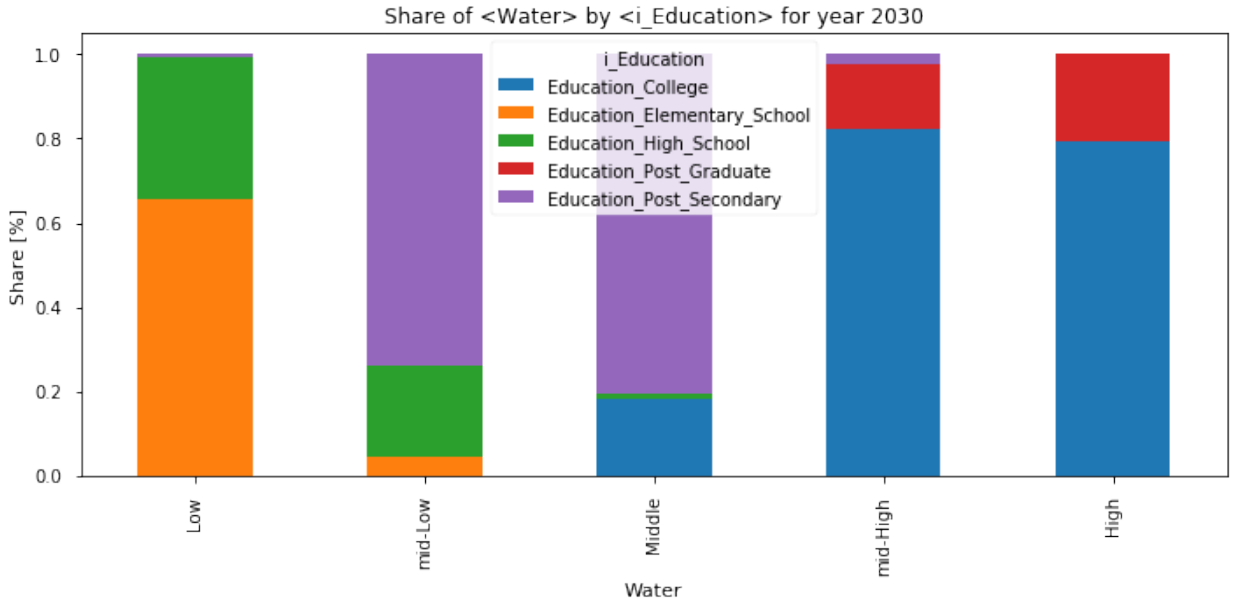


Scenario 2 grouped by education, cross tabulation

```
In [15]: from smum.microsim.util_plot import cross_tab
```

```
In [16]: a = 'Water'
         b = 'i_Education'
         ct = cross_tab(a, b, 2030, reweighted_survey + "_{}_scenario 2_0.60.csv", split_a = True)
```

```
data saved as: data/Water_i_Education_2030.xlsx
```



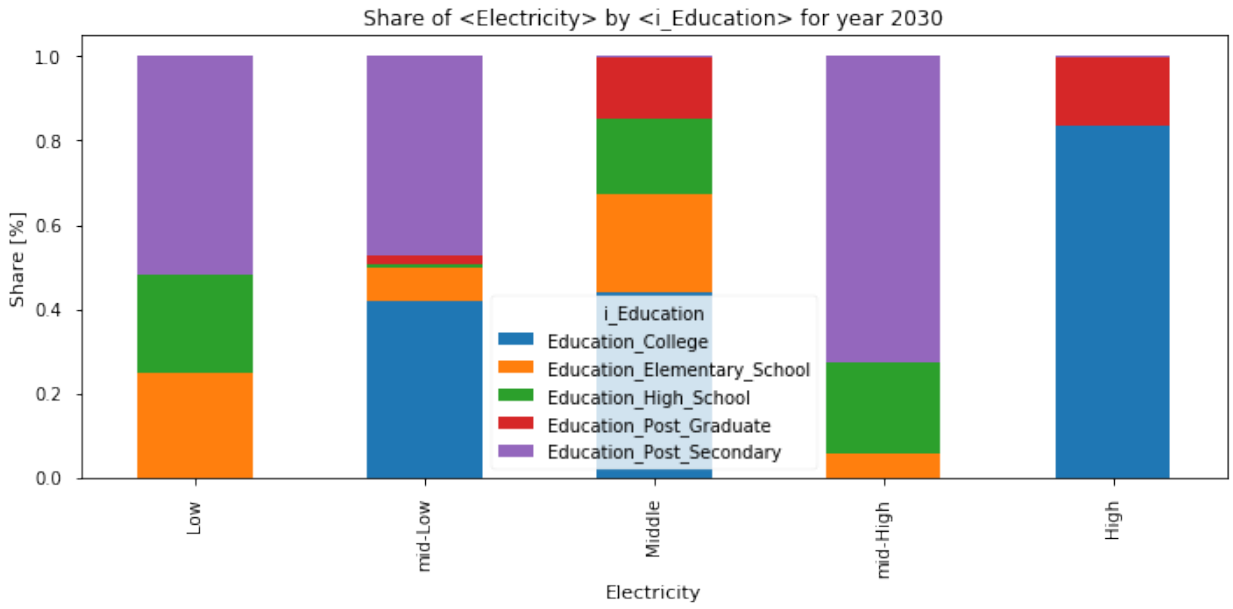

```
In [17]: ct
```

```
Out[17]: i_Education  Education_College  Education_Elementary_School  \
         Water
         Low                       NaN                      5764.04
         mid-Low                   NaN                       560.96
         Middle                1212.00                          NaN
         mid-High              9647.67                          NaN
         High                  4434.33                          NaN

         i_Education  Education_High_School  Education_Post_Graduate  \
         Water
         Low                       2933.70                      NaN
         mid-Low                   2584.57                      NaN
         Middle                      94.72                      NaN
         mid-High                      NaN                   1807.0
         High                          NaN                   1142.0

         i_Education  Education_Post_Secondary
         Water
         Low                            70.00
         mid-Low                      9002.94
         Middle                       5369.71
         mid-High                      282.68
         High                             NaN
```

```
In [18]: iterations = 1000
         typ = 'resampled'
         model_name = 'Sorsogon_Electricity_Water_wbias_projected_dynamic_{}'.format(typ)
         reweighted_survey = 'data/survey_{}_{}'.format(model_name, iterations)
```

```
In [19]: a = 'Electricity'
         b = 'i_Education'
         ct = cross_tab(a, b, 2030, reweighted_survey + "_{}_scenario 2_0.60.csv", split_a = True)
```

```
data saved as: data/Electricity_i_Education_2030.xlsx
```




```
In [20]: ct
```

```
Out[20]: i_Education  Education_College  Education_Elementary_School  \
         Electricity
         Low                       NaN                      2033.00
         mid-Low                5259.0                      1009.25
         Middle                 5559.0                      2934.69
         mid-High                  NaN                       348.06
         High                   4476.0                          NaN

         i_Education  Education_High_School  Education_Post_Graduate  \
         Electricity
         Low                        1886.00                      NaN
         mid-Low                      96.89                    283.0
         Middle                     2283.14                   1797.0
         mid-High                   1346.96                      NaN
         High                           NaN                    869.0

         i_Education  Education_Post_Secondary
         Electricity
         Low                          4222.00
         mid-Low                      5919.00
         Middle                         56.18
         mid-High                     4509.38
         High                           18.77
```

4.2.3 Advanced Example 2: Brussels

Brussels, Belgium

Abstract:

This is a simple implementation example of the developed **Spatial Microsimulation Urban Metabolism Model (SMUM)**.

The aim of this model is to identify and quantify the impact of transition pathways to a circular economy.

Two main algorithms implemented in the model, giving it it's name, are:

1. A Spatial Microsimulation, used for the construction of a synthetic population; and
2. An Urban Metabolism approach, used to benchmark consumption level at a city-level or neighbourhood level (making it spatial).

(Step 1) Constructing a synthetic population

On this section the example shows how to construct representative samples given the **distributions of aggregated variables**. The algorithm constructs the samples either by constructing a new sample for each simulation year (**resample method**) or by reweighting an initial sample for each simulation year (**reweighting method**).

In order to construct the samples the model needs input on: (a) the distribution functions of aggregate variables and (b) the changes over time. This means that the model required the **projected aggregated values**. The model provides some function to facilitate and visualize the projection of aggregated values.

For a spatial microsimulation, the model requires also values for each simulation area. In this case a combination of the resample and reweighting methodologies is implemented. The algorithm will construct a new sample for each simulation year at an aggregated level (e.g. city-level) and reweight this sample for each area (e.g. statistical census areas).

For the computation of resource consumption the model requires a **consumption model**. This model defines how the resource values are computed. This can be anything from a simple linear model to the implementation of external libraries.

Aggregate level benchmarks

(Step 1.a) Projecting demographic variables

Micro level consumption models

(Step 1.c) Micro-level Electricity demand model

(Step 1.d) Micro-level Water demand model

(Step 1.e) Micro-level Non-Residential model

(Step 2) Sampling and reweighting

This section takes care of the actual **sampling procedure** and subsequent reweighting of the proxy data. The sample will be constructed with help of an MCMC algorithm and reweighted with help of the GREGWT algorithm.

This section also presents the internal validation of the sampling procedure.

Dynamic samplic models

(Step 2.a) Dynamic Sampling Model and GREGWT

(Step 2.b) Non-Residential Model

Model internal validation

(Step 2.c) Model Internal Validation

(Step 3) Constructing scenarions:

The construction of scenarios can be define at **different steps** of the model. In a sense, the definition of scenarios start by the projection of aggregated values (see section 1). This section defines scenarios by defining technology adoption rates and changes in technology efficiency.

(Step 3.a.1) Define Transition Scenarions

(Step 3.b.2) Visualize transition scenarios

(Step 3.a.2) Define Transition Scenarions, Non-Residential

(Step 3.b.2) Visualize transition scenarios, Non-Residential

Brussels. Step 1.a Projecting demographic variables

```
In [ ]: import datetime; print(datetime.datetime.now())
```

```
2018-03-26 02:07:05.468122
```

Notebook Abstract:

This notebook shows an example on how to prepare projected aggregated benchmarks, as well as how to introduce **bias** (a proxy for simulating development scenarios at an aggregated level) to the benchmarks.

This step is not a requirement for running the simulation, the data containing projections at an aggregated level are a **requirement** for running the simulation.

A spatial model requires projected aggregated values for each simulation area. Depending on the simulation model the estimation of such values might be difficult.

Aggregated census data

```
In [2]: import pandas as pd
        from smum._scripts.aggregates import print_all
```

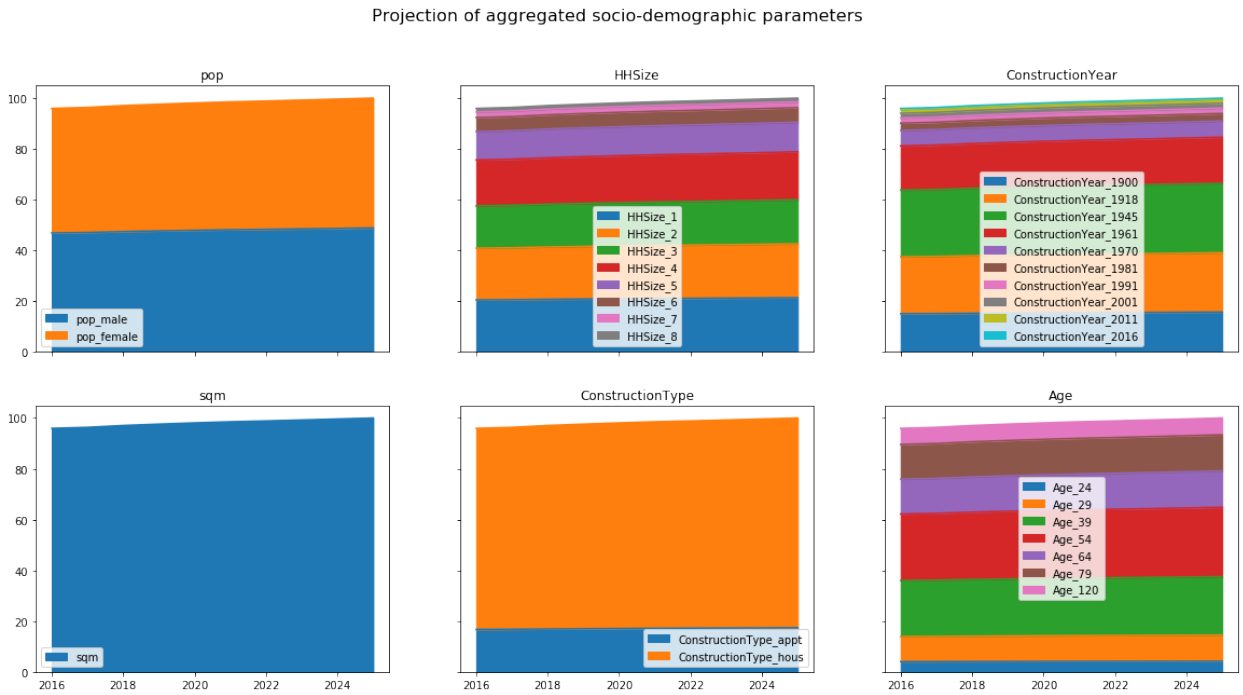
```
In [3]: census = pd.read_csv('data/benchmarks_be_year.csv', index_col=0)
        skip = ['pop', 'Income', 'Electricity', 'Water']
```

```
In [4]: census.columns = [
            'pop', 'pop_male', 'pop_female',
            'HHSize_1', 'HHSize_2', 'HHSize_3', 'HHSize_4',
            'HHSize_5', 'HHSize_6', 'HHSize_7', 'HHSize_8',
            'Income',
            'ConstructionYear_1900', 'ConstructionYear_1918',
            'ConstructionYear_1945', 'ConstructionYear_1961',
            'ConstructionYear_1970', 'ConstructionYear_1981',
            'ConstructionYear_1991', 'ConstructionYear_2001',
            'ConstructionYear_2011', 'ConstructionYear_2016',
            'sqm',
            'ConstructionType_appt', 'ConstructionType_hous',
            'Age_24', 'Age_29', 'Age_39',
            'Age_54', 'Age_64', 'Age_79', 'Age_120',
            'Electricity', 'Water']
```

We read a csv file containing the projected data with help of pandas. The `print_all` function is used to print the aggregated data.

A list of strings is constructed to tell the function which columns to ignore during plotting.

```
In [5]: _ = print_all(
            census, 'year',
            skip = skip,
            start_year = 2016, end_year = 2025,
            total_pop = census.loc[:, 'pop'],
            title="Projection of aggregated socio-demographic parameters"
                  )
```



We pass the following variables to the `print_all` function:

1. A pandas DataFrame containing the projected aggregated data.
2. A string defining the sufix used to save the file on disk.
3. `var` = defines the type of plot to use.
4. `skip` = defines the list of columns to skip in the plot.
5. `total_pop` = defines a column to use for data normalization.
6. `title` = defines the plot title.

Introduce bias to census data

The manipulations are expressed as **growth factors**. This factors increase the share (> 1) or decrease the share (< 1) of specific categories. In addition we can define gradual changes on these growth factors. If no starting point is given the function assumes initial simulation year.

The define growth rates are passed as a python dictionary to the function (`{'key'=value}`). Where `key` is the variable category to be modified and `value` is either a single number (assumed initial year to be initial simulation year) or a dictionary attributing a growth rate to sequential steps.

In this case we create three new categories in order to allocate values to:

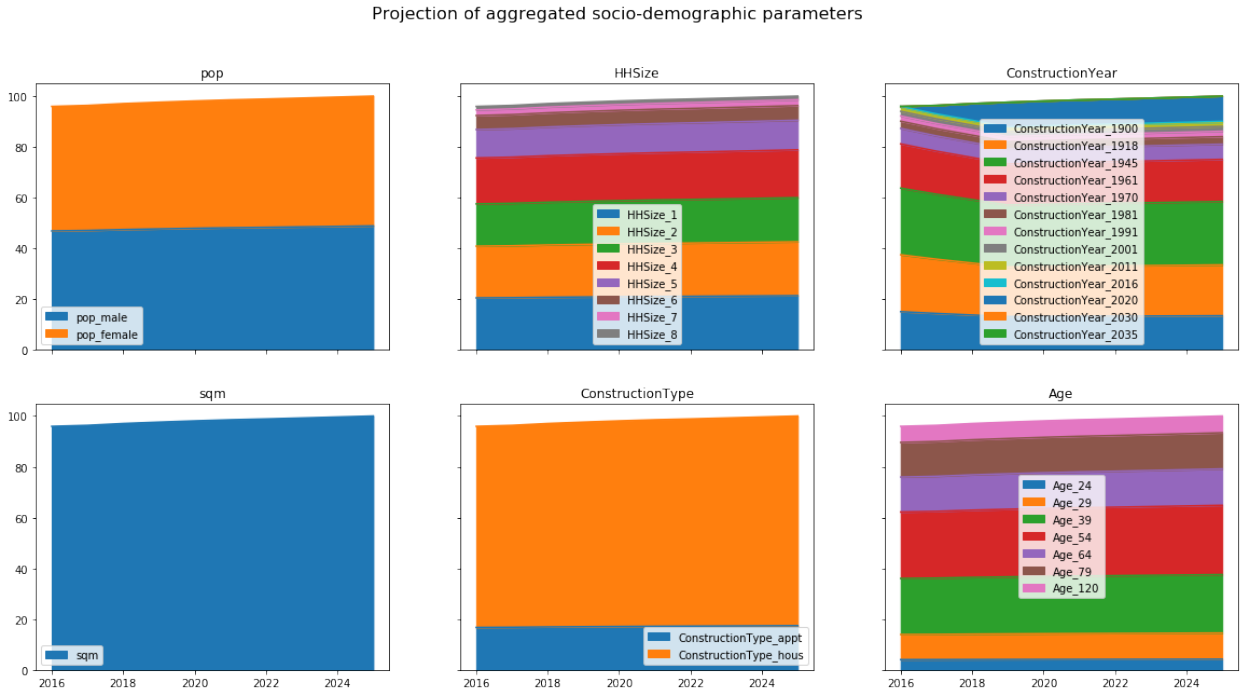
1. `ConstructionYear_2020`
2. `ConstructionYear_2030`
3. `ConstructionYear_2035`

```
In [6]: start = [0 for _ in census.index]
        census.insert(22, 'ConstructionYear_2020', start)
        census.insert(23, 'ConstructionYear_2030', start)
        census.insert(24, 'ConstructionYear_2035', start)
```

Iteration (1)

```
In [7]: bias = {
            'ConstructionYear_1900':   {2016:0.95, 2020:1,
                                'allocate_to': 'ConstructionYear_2020'},
            'ConstructionYear_1918':{2016:0.95, 2020:1,
                                'allocate_to': 'ConstructionYear_2020'},
            'ConstructionYear_1945':{2016:0.97, 2020:1,
                                'allocate_to': 'ConstructionYear_2020'},
            'ConstructionYear_1961':{2016:0.97, 2020:1,
                                'allocate_to': 'ConstructionYear_2020'},
            'ConstructionYear_1970':{2016:0.98, 2020:1,
                                'allocate_to': 'ConstructionYear_2020'},
            'ConstructionYear_1981':{2016:1,
                                'allocate_to': 'ConstructionYear_2020'},
            'ConstructionYear_1991':{2016:1,
                                'allocate_to': 'ConstructionYear_2020'},
            'ConstructionYear_2001':{2016:1,
                                'allocate_to': 'ConstructionYear_2020'},
            'ConstructionYear_2011':{2016:1,
                                'allocate_to': 'ConstructionYear_2020'},
        }
```

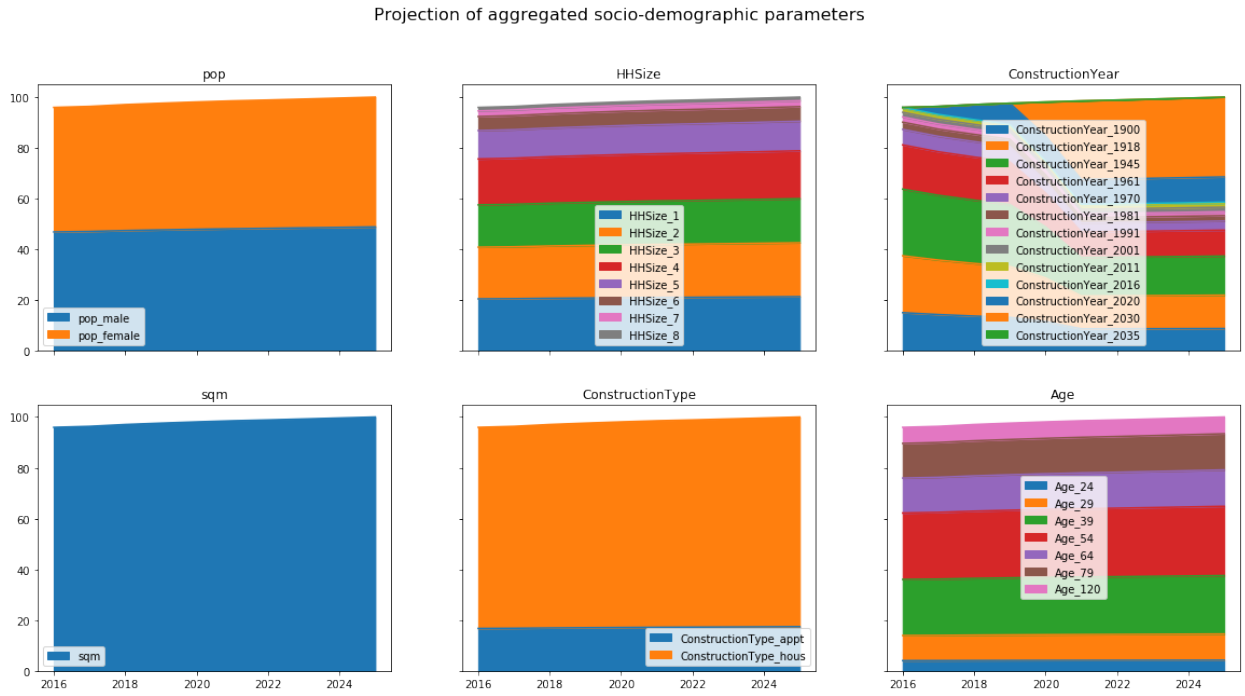
```
In [8]: census = print_all(
            census, 'year_bias_be',
            skip = skip,
            bias = bias,
            start_year = 2016, end_year = 2025,
            total_pop = census.loc[:, 'pop'],
            title="Projection of aggregated socio-demographic parameters",
            save_data = 'data/benchmarks_be_year_bias.csv'
                  )
```



Iteration (2)

```
In [9]: bias2 = {
            'ConstructionYear_1900'   :{2016:1, 2020:0.75, 2022:1,
                                'allocate_to': 'ConstructionYear_2030'
                               },
            'ConstructionYear_1918':{2016:1, 2020:0.75, 2022:1,
                                'allocate_to': 'ConstructionYear_2030'
                               },
            'ConstructionYear_1945':{2016:1, 2020:0.75, 2022:1,
                                'allocate_to': 'ConstructionYear_2030'
                               },
            'ConstructionYear_1961':{2016:1, 2020:0.75, 2022:1,
                                'allocate_to': 'ConstructionYear_2030'
                               },
            'ConstructionYear_1970':{2016:1, 2020:0.75, 2022:1,
                                'allocate_to': 'ConstructionYear_2030'
                               },
            'ConstructionYear_1981':{2016:1, 2020:0.85, 2022:1,
                                'allocate_to': 'ConstructionYear_2030'
                               },
            'ConstructionYear_1991':{2016:1, 2020:0.85, 2022:1,
                                'allocate_to': 'ConstructionYear_2030'
                               },
            'ConstructionYear_2001':{2016:1, 2020:0.96, 2022:1,
                                'allocate_to': 'ConstructionYear_2030'
                               },
            'ConstructionYear_2011':{2016:1,
                                'allocate_to': 'ConstructionYear_2030'
                               },
        }

In [10]: census = pd.read_csv('data/benchmarks_be_year_bias.csv', index_col=0)
```

```
_ = print_all(
    census, 'year_bias_be2',
    skip = skip,
    bias = bias2,
    start_year = 2016, end_year = 2025,
    total_pop = census.loc[:, 'pop'],
    title="Projection of aggregated socio-demographic parametsers",
    save_data = 'data/benchmarks_be_year_bias2.csv'
          )
```

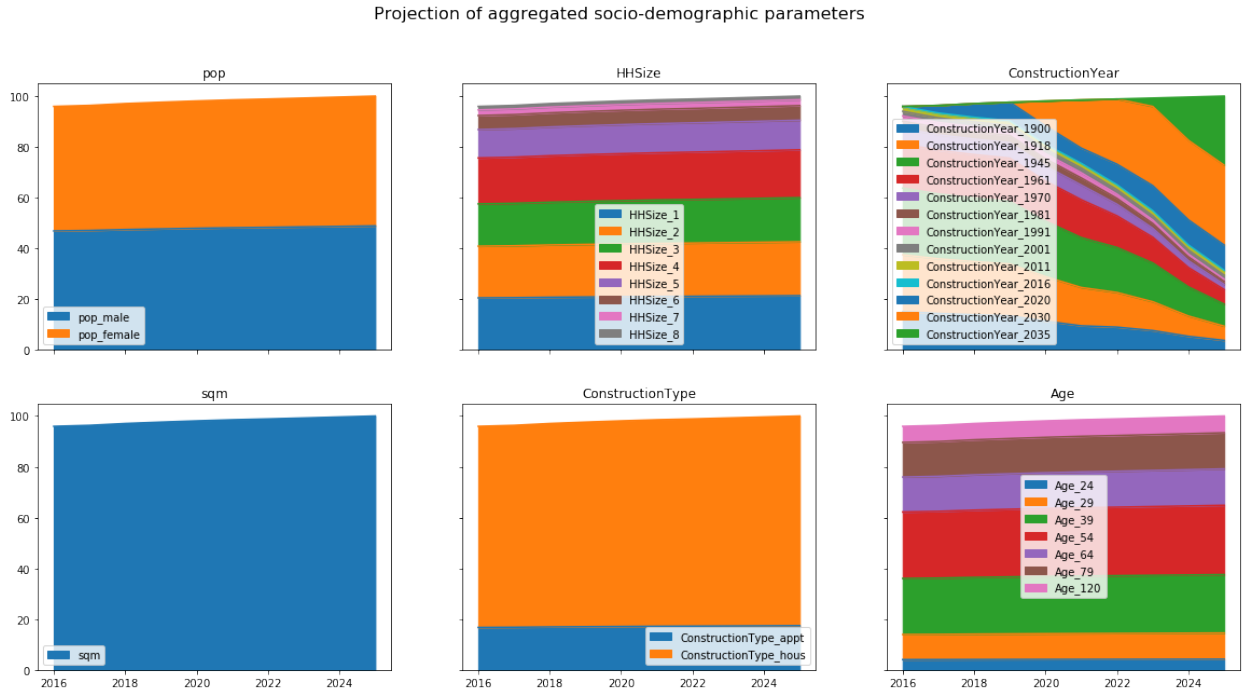


Iteration (3)

```
In [11]: bias3 = {
            'ConstructionYear_1900'   :{2016:1, 2020:1, 2022:0.70,
                                        'allocate_to': 'ConstructionYear_2035'
                                       },
            'ConstructionYear_1918':{2016:1, 2020:1, 2022:0.70,
                                        'allocate_to': 'ConstructionYear_2035'
                                       },
            'ConstructionYear_1945':{2016:1, 2020:1, 2022:0.75,
                                        'allocate_to': 'ConstructionYear_2035'
                                       },
            'ConstructionYear_1961':{2016:1, 2020:1, 2022:0.75,
                                        'allocate_to': 'ConstructionYear_2035'
                                       },
            'ConstructionYear_1970':{2016:1, 2020:1, 2022:0.75,
                                        'allocate_to': 'ConstructionYear_2035'
                                       },
            'ConstructionYear_1981':{2016:1, 2020:1, 2022:0.80,
                                        'allocate_to': 'ConstructionYear_2035'
                                       },
            'ConstructionYear_1991':{2016:1, 2020:1, 2022:0.80,
```

```
                                'allocate_to': 'ConstructionYear_2035'
                               },
                'ConstructionYear_2001':{2016:1, 2020:1, 2022:0.95,
                                'allocate_to': 'ConstructionYear_2035'
                               },
                'ConstructionYear_2011':{2016:1, 2020:1, 2022:0.99,
                                'allocate_to': 'ConstructionYear_2035'
                               },
        }
```

```
In [12]: census = pd.read_csv('data/benchmarks_be_year_bias2.csv', index_col=0)

        _ = print_all(
            census, 'year_bias_be3',
            skip = skip,
            bias = bias3,
            start_year = 2016, end_year = 2025,
            total_pop = census.loc[:, 'pop'],
            title="Projection of aggregated socio-demographic parameters",
            save_data = 'data/benchmarks_be_year_bias3.csv'
                  )
```



Brussels. Step 1.c Micro-level Electricity demand model

```
In [1]: import datetime; print(datetime.datetime.now())
```

```
2018-03-26 02:08:35.436887
```

Notebook Abstract:

A simple micro-level electricity demand model. The electricity demand model used available micro level data for the estimation of regression coefficients. This regression coefficients are used to define a table model. The electricity table model is used for the construction of a proxy micro level sample data set.

Prior electricity demand model

```
In [2]: import statsmodels.api as sm
        import pandas as pd
        import numpy as np
        from smum._scripts.micro import compute_categories, change_index
```

```
/usr/lib/python3.6/site-packages/statsmodels-0.8.0-py3.6-linux-x86_64.egg/statsmodels/compat/pandas.p
  from pandas.core import datetools
```

```
In [3]: electricity_data = pd.read_csv('data/elec_data_be.csv', index_col=0)
        #formula = "KWH ~ TOTSQMT_EN + CDD30YR + HDD30YR + MONEYPY + NHSLDMEM +\
        # C(TYPEHUQ, Treatment(reference='House')) + YEARMADE + ELWARM + ELWATER + ELFOOD"
        formula = "KWH ~ TOTSQMT_EN + CDD30YR + HDD30YR + MONEYPY + NHSLDMEM +\
         C(TYPEHUQ) + YEARMADE + ELWARM + ELWATER + ELFOOD"
```

```
In [4]: electricity_data.head()
```

Out[4]:

DOEID	TYPEHUQ	NWEIGHT	HDD65	CDD65	HDD30YR	CDD30YR	\
1	House	2471.68	4742.0	1080.0	4953.0	1271.0	
2	House	8599.17	2662.0	199.0	2688.0	143.0	
3	Apartment	8969.92	6233.0	505.0	5741.0	829.0	
4	House	18003.64	6034.0	672.0	5781.0	868.0	
5	House	5999.61	5388.0	702.0	5313.0	797.0	

DOEID	Climate_Region_Pub	UR	KOWNRENT	YEARMADE	...	DOLLARLP	\
1	4	U	Owned	2004	...	0.0	
2	5	U	Rented	1998	...	0.0	
3	1	U	Rented	1965	...	0.0	
4	1	U	Owned	1985	...	0.0	
5	1	U	Owned	1983	...	0.0	

DOEID	GALLONFO	DOLLARFO	GALLONKER	BTUKER	DOLLARKER	BTUWOOD	TOTALBTU	\
1	0.0	0.0	0.0	0.0	0.0	0.0	63006.0	
2	0.0	0.0	0.0	0.0	0.0	20000.0	103460.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	58716.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	76401.0	
5	0.0	0.0	0.0	0.0	0.0	0.0	59809.0	

DOEID	TOTALDOL	TOTSQMT_EN
1	1315.0	434.775
2	1293.0	254.448
3	1327.0	49.104
4	1398.0	150.939
5	1558.0	177.816

[5 rows x 51 columns]

```
In [5]: model = sm.WLS.from_formula(formula, electricity_data, weights=electricity_data.NWEIGHT)
        model_results = model.fit()
        #params = model_results.params
        #bse = model_results.bse
        params = change_index(model_results.params)
        bse = change_index(model_results.bse)
```

```
In [6]: model_results.summary()
```

Out[6]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            WLS Regression Results
==============================================================================
Dep. Variable:                    KWH   R-squared:                       0.451
Model:                            WLS   Adj. R-squared:                  0.450
Method:                 Least Squares   F-statistic:                     946.8
Date:                Mon, 26 Mar 2018   Prob (F-statistic):               0.00
Time:                        02:08:38   Log-Likelihood:            -1.1684e+05
No. Observations:               11542   AIC:                         2.337e+05
Df Residuals:                   11531   BIC:                         2.338e+05
Df Model:                          10
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
Intercept                    -2.996e+04   4347.551     -6.891      0.000   -3.85e+04   -2.14e+04
C(TYPEHUQ)[T.House]           2752.5037    138.890     19.818      0.000    2480.256    3024.751
TOTSQMT_EN                      16.7486      0.524     31.943      0.000      15.721      17.776
CDD30YR                          1.7152      0.089     19.301      0.000       1.541       1.889
HDD30YR                          0.1985      0.041      4.886      0.000       0.119       0.278
MONEYPY                          0.0323      0.003     12.110      0.000       0.027       0.038
NHSLDMEM                      1037.7339     36.334     28.561      0.000     966.512    1108.955
YEARMADE                        12.9729      2.215      5.858      0.000       8.632      17.314
ELWARM                        2296.3175    119.387     19.234      0.000    2062.298    2530.337
ELWATER                       3660.4636    130.959     27.951      0.000    3403.761    3917.166
ELFOOD                        1396.9598    123.022     11.355      0.000    1155.816    1638.104
==============================================================================
Omnibus:                     8161.961   Durbin-Watson:                   1.969
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           474206.898
Skew:                           2.788   Prob(JB):                         0.00
Kurtosis:                      33.902   Cond. No.                     3.26e+06
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.26e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

In [7]:
```
elec = pd.concat([params, bse], axis=1)
elec.columns = ['co_mu', 'co_sd']
```

In [8]:
```
elec.index
```

Out[8]:
```
Index(['Intercept', 'TYPEHUQ_House', 'TOTSQMT_EN', 'CDD30YR', 'HDD30YR',
       'MONEYPY', 'NHSLDMEM', 'YEARMADE', 'ELWARM', 'ELWATER', 'ELFOOD'],
      dtype='object')
```

In [9]:
```
elec.loc[:, 'p'] = np.nan
elec.loc[:,'mu'] = np.nan
elec.loc[:,'sd'] = np.nan
elec.loc[:, 'dis'] = 'Normal'
elec.loc[['CDD30YR','HDD30YR'], 'dis'] = 'Deterministic'
elec.loc['MONEYPY', 'dis'] = 'None'
#elec.loc[["TYPEHUQ, Treatmentreference='House'_Apartment", 'ELWARM', 'ELWATER', 'ELFOOD'],
elec.loc[["TYPEHUQ_House", 'ELWARM', 'ELWATER', 'ELFOOD'], 'dis'] = 'None'
elec.loc[['NHSLDMEM', 'YEARMADE'], 'dis'] = 'None'
elec.loc[:,'ub'] = np.inf
elec.loc[:,'lb'] = 0
elec.loc['NHSLDMEM', 'lb'] = 1
```

```
        elec.loc['NHSLDMEM', 'ub'] = 8
        elec.loc['YEARMADE', 'lb'] = 1800
        elec.loc['YEARMADE', 'ub'] = 2035
        elec.loc['Intercept', 'lb'] = -np.inf
        elec.loc['Intercept', 'p'] = elec.loc['Intercept', 'co_mu']
        elec.loc['Intercept', 'dis'] = 'Deterministic'
        elec.loc['Intercept', 'co_mu'] = np.nan
        elec.loc['Intercept', 'co_sd'] = np.nan
```

In [10]:
```
elec.index = [
    'e_Intercept',
    'e_ConstructionType',
    'e_sqm',
    'e_CDD',
    'e_HDD',
    'e_Income',
    'e_HHSize',
    'e_ConstructionYear',
    'e_ELWARM',
    'e_ELWATER',
    'e_ELFOOD'
]
```

In [11]:
```
skip = [
    'e_ELWARM',
    'e_ELWATER',
    'e_ELFOOD'
]
elec = elec.loc[[i for i in elec.index if i not in skip]]
```

In [12]:
```
elec.to_csv('data/table_elec.csv')
```

In [13]:
```
elec
```

Out[13]:

	co_mu	co_sd	p	mu	sd	\
e_Intercept			NaN	NaN	-29960.626896	NaN NaN
e_ConstructionType	2752.503665	138.889881			NaN	NaN NaN
e_sqm	16.748580	0.524321			NaN	NaN NaN
e_CDD	1.715248	0.088868			NaN	NaN NaN
e_HDD	0.198506	0.040625			NaN	NaN NaN
e_Income	0.032290	0.002666			NaN	NaN NaN
e_HHSize	1037.733889	36.334410			NaN	NaN NaN
e_ConstructionYear	12.972898	2.214585			NaN	NaN NaN

	dis	ub	lb
e_Intercept	Deterministic	inf	-inf
e_ConstructionType	None	inf	0.000000
e_sqm	Normal	inf	0.000000
e_CDD	Deterministic	inf	0.000000
e_HDD	Deterministic	inf	0.000000
e_Income	None	inf	0.000000
e_HHSize	None	8.000000	1.000000
e_ConstructionYear	None	2035.000000	1800.000000

Brussels. Step 1.d Micro-level Water demand model

In [1]:
```
import datetime; print(datetime.datetime.now())
```

```
2018-03-26 02:13:23.269759
```

Notebook abstract

A simple micro-level water demand model.

A simple micro-level water demand model. Similar to the electricity demand model, the water demand model uses micro level consumption demand data for the construction of a table model. The table model describes simple rules for the construction of the proxy micro level sample data.

Prior water demand model

```
In [2]: import statsmodels.api as sm
        import pandas as pd
        import numpy as np
        from smum._scripts.micro import compute_categories, change_index
```

```
/usr/lib/python3.6/site-packages/statsmodels-0.8.0-py3.6-linux-x86_64.egg/statsmodels/compat/pandas.p
  from pandas.core import datetools
```

Dataset

The data used for the construction of the household water demand model is downscale from municipal-level data-sets from the Wallonie region.

This data to construct a water demand model a function of: 1. Building construction type (classified as house or apartment) 2. Building construction year 3. Head of household gender 4. Household size 5. Household income level

```
In [3]: water_data = pd.read_csv('data/downscaled_data_be.csv', index_col=0)
        water_data.loc[
            water_data.ConstructionType != 'appartements',
            'ConstructionType'] = 'House'

        formula = "Water ~ Age + ConstructionType + ConstructionYear + HHSize + Income"
```

```
In [4]: water_data.head()
```

Out[4]:

	Age	ConstructionType	ConstructionYear	Energy	Gender	HHSize	Income	Waste	Water	w
0	70	House	1900	49	femme	2.61	28150.0	147.0	79	71.45
1	22	House	1977	49	femme	2.61	28150.0	147.0	79	71.45
2	28	House	1972	49	femme	2.61	28150.0	147.0	79	71.45
3	7	House	1900	49	femme	2.61	28150.0	147.0	79	71.45
4	33	House	1962	49	homme	2.61	28150.0	147.0	79	71.45

Regression model

A weighted multiple regression model is defined and run with help of the 'statsmodels' python library.

The model regression coefficients and their standard error are used as the water demand model.

```
In [5]: model_water = sm.WLS.from_formula(formula, water_data, weights=water_data.w)
        model_results_water = model_water.fit()
```

```
In [6]: model_results_water.summary()
```

Out[6]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            WLS Regression Results
==============================================================================
Dep. Variable:                  Water   R-squared:                       0.360
Model:                            WLS   Adj. R-squared:                  0.360
Method:                 Least Squares   F-statistic:                     2519.
Date:                Mon, 26 Mar 2018   Prob (F-statistic):               0.00
Time:                        02:13:26   Log-Likelihood:                -75359.
No. Observations:               22412   AIC:                         1.507e+05
Df Residuals:                   22406   BIC:                         1.508e+05
Df Model:                           5
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.025
------------------------------------------------------------------------------------------------
Intercept                       -5.2824      2.273     -2.324      0.020      -9.738
ConstructionType[T.appartements]     1.1517      0.112     10.307      0.000       0.933
Age                              0.0005      0.002      0.305      0.760      -0.003
ConstructionYear                 0.0157      0.001     13.527      0.000       0.013
HHSize                          10.3606      0.273     37.966      0.000       9.826
Income                           0.0010   1.27e-05     75.207      0.000       0.001
==============================================================================
Omnibus:                     2964.935   Durbin-Watson:                   0.038
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             7638.192
Skew:                           0.748   Prob(JB):                         0.00
Kurtosis:                       5.438   Cond. No.                     1.28e+06
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.28e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

In [7]:
```
params_water = change_index(model_results_water.params)
bse_water = change_index(model_results_water.bse)
water = pd.concat([params_water, bse_water], axis=1)
water.columns = ['co_mu', 'co_sd']
```

In [8]:
```
water.index = [
    'Intercept', 'ConstructionType', 'Age',
    'ConstructionYear', 'HHSize', 'Income',
]
```

In [9]:
```
water.loc[:, 'p'] = np.nan
water.loc[:,'mu'] = np.nan
water.loc[:,'sd'] = np.nan

water.loc['ConstructionType', 'dis'] = 'Bernoulli'
water.loc['Age', 'dis'] = 'Normal'
water.loc['ConstructionYear', 'dis'] = 'Poisson'
water.loc['HHSize', 'dis'] = 'Poisson'
water.loc['Income', 'dis'] = 'Gamma'

water.loc[:, 'ub'] = np.inf
water.loc[:, 'lb'] = 0
water.loc['Intercept', 'lb'] = -np.inf
water.loc['Age', 'ub'] = 85
water.loc['Age', 'lb'] = 20
```

```
water.loc['HHSize', 'ub'] = 8
water.loc['HHSize', 'lb'] = 1
water.loc['ConstructionYear', 'ub'] = 2035
water.loc['ConstructionYear', 'lb'] = 1800

water.loc['Intercept', 'p'] = water.loc['Intercept', 'co_mu']
water.loc['Intercept', 'dis'] = 'Deterministic'
water.loc['Intercept', 'co_mu'] = np.nan
water.loc['Intercept', 'co_sd'] = np.nan
```

```
In [10]: water.index = ['w_'+i for i in water.index]
```

```
In [11]: water.to_csv('data/table_water.csv')
```

```
In [12]: water
```

```
Out[12]: co_mu     co_sd         p  mu  sd            dis  \
         w_Intercept               NaN       NaN -5.282409 NaN NaN  Deterministic
         w_ConstructionType   1.151750  0.111739       NaN NaN NaN      Bernoulli
         w_Age                0.000516  0.001691       NaN NaN NaN         Normal
         w_ConstructionYear   0.015676  0.001159       NaN NaN NaN        Poisson
         w_HHSize            10.360614  0.272889       NaN NaN NaN        Poisson
         w_Income             0.000957  0.000013       NaN NaN NaN          Gamma

                                    ub           lb
         w_Intercept               inf         -inf
         w_ConstructionType        inf     0.000000
         w_Age                85.000000    20.000000
         w_ConstructionYear 2035.000000  1800.000000
         w_HHSize              8.000000     1.000000
         w_Income                  inf     0.000000
```

Brussels. Step 1.e Micro-level Non-Residential model

```
In [1]: import datetime; print(datetime.datetime.now())
```

```
2018-03-26 02:14:48.583647
```

Notebook abstract

A simple micro-level building stock model. The consumption model defined for the building stock works in theory exactly like the other micro level consumption model. The difference between this model and the income, electricity and water demand models is that we don't have a micro-level consumption data set in order to extract regression coefficients. In order to define a consumption model we use predefine building typologies.

Prior non-residential model

```
In [2]: import pandas as pd
```

```
In [3]: TypesB = pd.read_csv('data/TypesB.csv', index_col=[0,1])
```

```
In [4]: TypesB
```

```
Out[4]: heat         cool        elec  \
        group typ
        comm  Services                              114.646120    42.900227   85.453653
              Food services                         820.440000   125.188571  298.847619
              Lodging                                79.161081    23.435315   74.837117
              Office building                        55.291667    31.980083   92.729167
              Warehous/Storage                      101.099570     1.860645   41.330108
        indu  Chemical - Pharmaceutical/Medical    183.492971     9.645261  198.687800
```

```
             Fabricated metal products           33.160702     1.528521  228.669574
             Food                               483.129187   275.699073  255.629133
             Furniture                           39.597494     2.899449  172.696441
             Machinery                           27.536797    24.687100  230.904416
             Plastic/Rubber products             31.946667  1396.809143  516.179810
             Textiles - Apparel                  82.170370    16.185185  243.263426

                                                    sqm  entities         p
       group typ
       comm  Services                            3600.0      7610  0.383472
             Food services                        525.0      3020  0.152179
             Lodging                            11100.0       259  0.013051
             Office building                    12000.0      8099  0.408113
             Warehous/Storage                    4650.0       130  0.006551
       indu  Chemical - Pharmaceutical/Medical    105.0        40  0.002016
             Fabricated metal products            105.0        95  0.004787
             Food                                  39.0       362  0.018241
             Furniture                            105.0        45  0.002268
             Machinery                            105.0        78  0.003930
             Plastic/Rubber products              105.0        18  0.000907
             Textiles - Apparel                    36.0        89  0.004485
```

```
In [5]: sub_typ = TypesB.loc[:, ['heat', 'cool', 'sqm', 'entities']]

In [6]: sub_typ_sum = sub_typ.loc[:, ['heat', 'cool']].mul(sub_typ.sqm.mul(sub_typ.entities), axis=0)

In [7]: sub_typ_sum.div(sub_typ_sum.sum())

Out[7]: heat    0.689349
        cool    0.310651
        dtype: float64

In [8]: input_sd = 0.01

In [9]: nrb_elec = pd.DataFrame(columns=['co_mu', 'co_sd', 'p', 'dis', 'lb', 'ub'])

In [10]: nrb_elec.loc['BuildingSqm', 'co_mu'] = ",".join([str(i) for i in TypesB.loc[:, 'sqm']])
         nrb_elec.loc['BuildingSqm', 'co_sd'] = ",".join([str(i * input_sd) for i in TypesB.loc[:, 's
         nrb_elec.loc['BuildingSqm', 'dis'] = "Deterministic;n;Categorical"

         nrb_elec.loc['BuildingHeat', 'co_mu'] = ",".join([str(i) for i in TypesB.loc[:, 'heat']])
         nrb_elec.loc['BuildingHeat', 'co_sd'] = ",".join([str(i * input_sd) for i in TypesB.loc[:,
         nrb_elec.loc['BuildingHeat', 'dis'] = "Deterministic;BuildingSqm;Categorical"

         nrb_elec.loc['BuildingCool', 'co_mu'] = ",".join([str(i) for i in TypesB.loc[:, 'cool']])
         nrb_elec.loc['BuildingCool', 'co_sd'] = ",".join([str(i * input_sd) for i in TypesB.loc[:,
         nrb_elec.loc['BuildingCool', 'dis'] = "Deterministic;BuildingSqm;Categorical"

         nrb_elec.loc['BuildingElec', 'co_mu'] = ",".join([str(i) for i in TypesB.loc[:, 'elec']])
         nrb_elec.loc['BuildingElec', 'co_sd'] = ",".join([str(i * input_sd) for i in TypesB.loc[:,
         nrb_elec.loc['BuildingElec', 'dis'] = "Deterministic;BuildingSqm;Categorical"

         nrb_elec.loc[:, 'p'] = ",".join([str(i * input_sd) for i in TypesB.loc[:, 'p']])
         nrb_elec.loc[:, 'lb'] = 0

In [11]: nrb_elec.to_csv('data/table_elec_nr.csv')

In [12]: nrb_elec

Out[12]: co_mu  \
         BuildingSqm   3600.0,525.0,11100.0,12000.0,4650.0,105.0,105....
         BuildingHeat  114.64612041392596,820.44,79.16108108108108,55...
         BuildingCool  42.90022677542567,125.18857142857142,23.435315...
```

```
             BuildingElec  85.45365281064835,298.84761904761905,74.837117...

                                                                    co_sd  \
             BuildingSqm   36.0,5.25,111.0,120.0,46.5,1.05,1.05,0.39,1.05...
             BuildingHeat  1.1464612041392597,8.204400000000001,0.7916108...
             BuildingCool  0.4290022677542567,1.2518857142857143,0.234353...
             BuildingElec  0.8545365281064835,2.9884761904761907,0.748371...

                                                                       p  \
             BuildingSqm   0.00383471907281,0.00152179390275,0.0001305114...
             BuildingHeat  0.00383471907281,0.00152179390275,0.0001305114...
             BuildingCool  0.00383471907281,0.00152179390275,0.0001305114...
             BuildingElec  0.00383471907281,0.00152179390275,0.0001305114...

                                                      dis  lb   ub
             BuildingSqm                Deterministic;n;Categorical   0  NaN
             BuildingHeat  Deterministic;BuildingSqm;Categorical   0  NaN
             BuildingCool  Deterministic;BuildingSqm;Categorical   0  NaN
             BuildingElec  Deterministic;BuildingSqm;Categorical   0  NaN
```

Brussels. Step 2.a Dynamic Sampling Model and GREGWT

```
In [1]: import datetime; print(datetime.datetime.now())
```

```
2018-04-03 15:01:31.465870
```

Notebook abstract

This notebook shows the main sampling and reweighting algorithm.

Import libraries

```
In [2]: from smum.microsim.run import run_calibrated_model
        from smum.microsim.table import TableModel
```

```
/usr/lib/python3.6/site-packages/h5py-2.7.1-py3.6-linux-x86_64.egg/h5py/__init__.py:36: FutureWarning
  from ._conv import register_converters as _register_converters
```

Global variables

```
In [3]: iterations = 1000
        year = 2016
        census_file = 'data/benchmarks_be_year_bias3_climate.csv'
        typ = 'resampled'
        model_name = 'Brussels_Electricity_Water_projected_dynamic_{}_bias'.format(typ)
        verbose = False
        #The number of chains to run in parallel.
        njobs = 4
```

Define Table model

```
In [4]: tm = TableModel(census_file = census_file, verbose=verbose)
```

Water model

```
In [5]: tm.add_model('data/table_water.csv', 'Water')

        tm.update_dynamic_model(
            'Water', specific_col = 'ConstructionType', select = 1)
        tm.update_dynamic_model(
            'Water', specific_col = 'Age', val = 'mu', compute_average = 0)
        tm.update_dynamic_model(
            'Water', specific_col = 'ConstructionYear', val = 'mu')
        tm.update_dynamic_model(
            'Water', specific_col = 'HHSize', val = 'mu')
        tm.update_dynamic_model(
            'Water', specific_col = 'Income', val = 'mu',
            compute_average = False)

In [6]: tm.models['Water'].loc[2020]

Out[6]: co_mu        co_sd         p       mu       sd  \
        w_Intercept                 NaN          NaN  -5.28241      NaN      NaN
        w_ConstructionType      1.15175     0.111739  0.825655      NaN      NaN
        w_Age                0.000515922   0.00169118       NaN  56.0519  23.2562
        w_ConstructionYear     0.0156761   0.00115888       NaN   1957.4  40.6553
        w_HHSize                 10.3606     0.272889       NaN  3.11831  1.71888
        w_Income              0.000956665  1.27204e-05       NaN  13648.4  136.484

                                    dis    ub    lb
        w_Intercept         Deterministic   inf  -inf
        w_ConstructionType      Bernoulli   inf     0
        w_Age                      Normal    85    20
        w_ConstructionYear        Poisson  2035  1800
        w_HHSize                  Poisson     8     1
        w_Income                    Gamma   inf     0

In [7]: formula_water = "+".join(
            ["c_{0}*{0}".format(e) for e in tm.models['Water'][year].index if\
             (e not in  ['w_Intercept'])
            ])
        tm.add_formula(formula_water, 'Water')

In [8]: tm.add_formula(formula_water, 'Water')

In [9]: tm.print_formula('Water')

Water =
         c_w_ConstructionType*w_ConstructionType +
         c_w_Age*w_Age +
         c_w_ConstructionYear*w_ConstructionYear +
         c_w_HHSize*w_HHSize +
         c_w_Income*w_Income +
```

Electricity model

```
In [10]: tm.add_model('data/table_elec.csv',  'Electricity',
                      skip_cols = [
                          'ConstructionType',
                          'Income',
                          'HHSize',
                          'ConstructionYear',
                          'ELWARM',
```

```
                    'ELWATER',
                    'ELFOOD'])
        tm.update_dynamic_model(
           'Electricity', specific_col = 'sqm', val = 'mu',
            compute_average = False)
        tm.update_dynamic_model(
            'Electricity', specific_col = 'CDD',
            static = True,
            compute_average = False)
        tm.update_dynamic_model(
            'Electricity', specific_col = 'HDD',
            static = True,
            compute_average = False)
```

In [11]:
```
tm.models['Electricity'].loc[2016]
```

Out[11]:
```
co_mu       co_sd        p       mu         sd  \
e_Intercept                NaN         NaN -29960.6       NaN       NaN
e_ConstructionType      2752.5      138.89       NaN       NaN       NaN
e_sqm                  16.7486    0.524321       NaN  73.0045  0.730045
e_CDD                  1.71525   0.0888685    833.3       NaN       NaN
e_HDD                 0.198506   0.0406247   3006.5       NaN       NaN
e_Income             0.0322898  0.00266627       NaN       NaN       NaN
e_HHSize               1037.73     36.3344       NaN       NaN       NaN
e_ConstructionYear     12.9729     2.21459       NaN       NaN       NaN

                                  dis    ub    lb
e_Intercept         Deterministic   inf  -inf
e_ConstructionType           None   inf     0
e_sqm                      Normal   inf     0
e_CDD               Deterministic   inf     0
e_HDD               Deterministic   inf     0
e_Income                     None   inf     0
e_HHSize                     None     8     1
e_ConstructionYear           None  2035  1800
```

In [12]:
```
skip_elec = [
    'e_Intercept', 'e_ConstructionType', 'e_Income', 'e_HHSize', 'e_ConstructionYear',
    'e_CDD', 'e_HDD',
]
formula_elec = "+".join(
    ["c_{0}*{0}".format(e) for e in tm.models['Electricity'][year].index \
     if (e not in skip_elec)
    ])
formula_elec += '+c_e_ConstructionType*w_ConstructionType +\
c_e_Income*w_Income +\
c_e_HHSize*w_HHSize +\
c_e_ConstructionYear*w_ConstructionYear+\
e_CDD +\
e_HDD'
```

In [13]:
```
tm.add_formula(formula_elec, 'Electricity')
```

In [14]:
```
tm.print_formula('Electricity')
```

```
Electricity =
        c_e_sqm*e_sqm +
        c_e_ConstructionType*w_ConstructionType  +
        c_e_Income*w_Income  +
        c_e_HHSize*w_HHSize  +
        c_e_ConstructionYear*w_ConstructionYear +
```

```
        e_CDD  +
        e_HDD +
```

Make model and save it to excel

```
In [15]: table_model = tm.make_model()

In [16]: tm.to_excel(sufix = "_be")

creating data/tableModel_Water_be.xlsx
creating data/tableModel_Electricity_be.xlsx
```

Define model variables

```
In [17]: labels_age = [
             'Age_24', 'Age_29', 'Age_39',#3
             'Age_54', 'Age_64', 'Age_79',#6
             'Age_120']
         cut_age = [17,
                24, 29, 39,
                54, 64, 79,
                120]

         labels_cy = [
             'ConstructionYear_1900', 'ConstructionYear_1918',
             'ConstructionYear_1945', 'ConstructionYear_1961',
             'ConstructionYear_1970', 'ConstructionYear_1981',
             'ConstructionYear_1991', 'ConstructionYear_2001',
             'ConstructionYear_2011', 'ConstructionYear_2016',
             'ConstructionYear_2020', 'ConstructionYear_2030',
             'ConstructionYear_2035']
         cut_cy = [0,
                   1900, 1918,
                   1945, 1961,
                   1970, 1981,
                   1991, 2001,
                   2011, 2016,
                   2020, 2030,
                   2100]

         to_cat = {
             'w_Age':[cut_age, labels_age],
             'w_ConstructionYear':[cut_cy, labels_cy],
                  }

         drop_col_survey = [
             'e_ConstructionType', 'e_Income', 'e_HHSize', 'e_ConstructionYear',
             'e_HDD', 'e_CDD'
         ]

In [ ]: fw = run_calibrated_model(
            table_model,
            verbose = verbose,
            project = typ,
            njobs = njobs,
            census_file = census_file,
            year = year,
```

```
            name = '{}_{}'.format(model_name, iterations),
            to_cat = to_cat,
            iterations = iterations,
            drop_col_survey = drop_col_survey)
```

Brussels. Step 2.b Dynamic Sampling Model and GREGWT, Non-Residential Model

```
In [1]: import datetime; print(datetime.datetime.now())
```

```
2018-04-03 15:12:00.614979
```

Notebook abstract

This notebook shows the main sampling and reweighting algorithm for the non-residential sector.

Import libraries

```
In [2]: from smum.microsim.run import run_calibrated_model
        from smum.microsim.util_plot import plot_data_projection
        from smum.microsim.table import TableModel
```

```
/usr/lib/python3.6/site-packages/h5py-2.7.1-py3.6-linux-x86_64.egg/h5py/__init__.py:36: FutureWarning
  from ._conv import register_converters as _register_converters
```

Global variables

```
In [3]: iterations = 10000
        benchmark_year = 2016
        census_file = 'data/benchmarks_nonresidential.csv'
        typ = 'resampled'
        model_name = 'Brussels_NonResidentialElectricity_wbias_projected_dynamic_{}'.format(typ)
        verbose = False
        drop_col_survey = ['h_BuildingHeat', 'c_BuldingCool', 'e_BuildingElec',
                           'h_BuildingSqm', 'c_BuildingSqm']
```

Define model

```
In [4]: tm = TableModel(census_file = census_file, verbose=verbose)
        table_model_name = 'data/table_elec_nr_{}.csv'
```

Electricity

```
In [5]: tm.add_model(table_model_name.format('e'), 'elec', static = True)
        tm.add_model(table_model_name.format('h'), 'heat', static = True)
        tm.add_model(table_model_name.format('c'), 'cool', static = True)
```

```
In [6]: tm.models['cool'].loc[2020]
```

```
Out[6]: co_mu  \
        c_BuildingSqm   3600.0,525.0,11100.0,12000.0,4650.0,105.0,105....
        c_BuildingCool  42.9002267754,125.188571429,23.4353153153,31.9...

                                                                co_sd  \
        c_BuildingSqm   36.0,5.25,111.0,120.0,46.5,1.05,1.05,0.39,1.05...
        c_BuildingCool  0.429002267754,1.25188571429,0.234353153153,0....
```

```
                                                                    p  \
c_BuildingSqm   0.00383471907281,0.00152179390275,0.0001305114...
c_BuildingCool  0.00383471907281,0.00152179390275,0.0001305114...

                                                    dis lb   ub
c_BuildingSqm                 Deterministic;c;Categorical  0  inf
c_BuildingCool  Deterministic;BuildingSqm;Categorical  0  inf
```

```
In [7]: formula_nrb = "c_e_BuildingElec * e_BuildingElec * c_e_BuildingSqm * e_BuildingSqm"
        tm.add_formula(formula_nrb, 'elec')
```

```
In [8]: formula_nrb = "c_h_BuildingHeat * h_BuildingHeat * c_e_BuildingSqm * e_BuildingSqm"
        tm.add_formula(formula_nrb, 'heat')
```

```
In [9]: formula_nrb = "c_c_BuildingCool * c_BuildingCool * c_e_BuildingSqm * e_BuildingSqm"
        tm.add_formula(formula_nrb, 'cool')
```

```
In [10]: table_model = tm.make_model()
         tm.to_excel()
```

```
creating data/tableModel_elec.xlsx
creating data/tableModel_heat.xlsx
creating data/tableModel_cool.xlsx
```

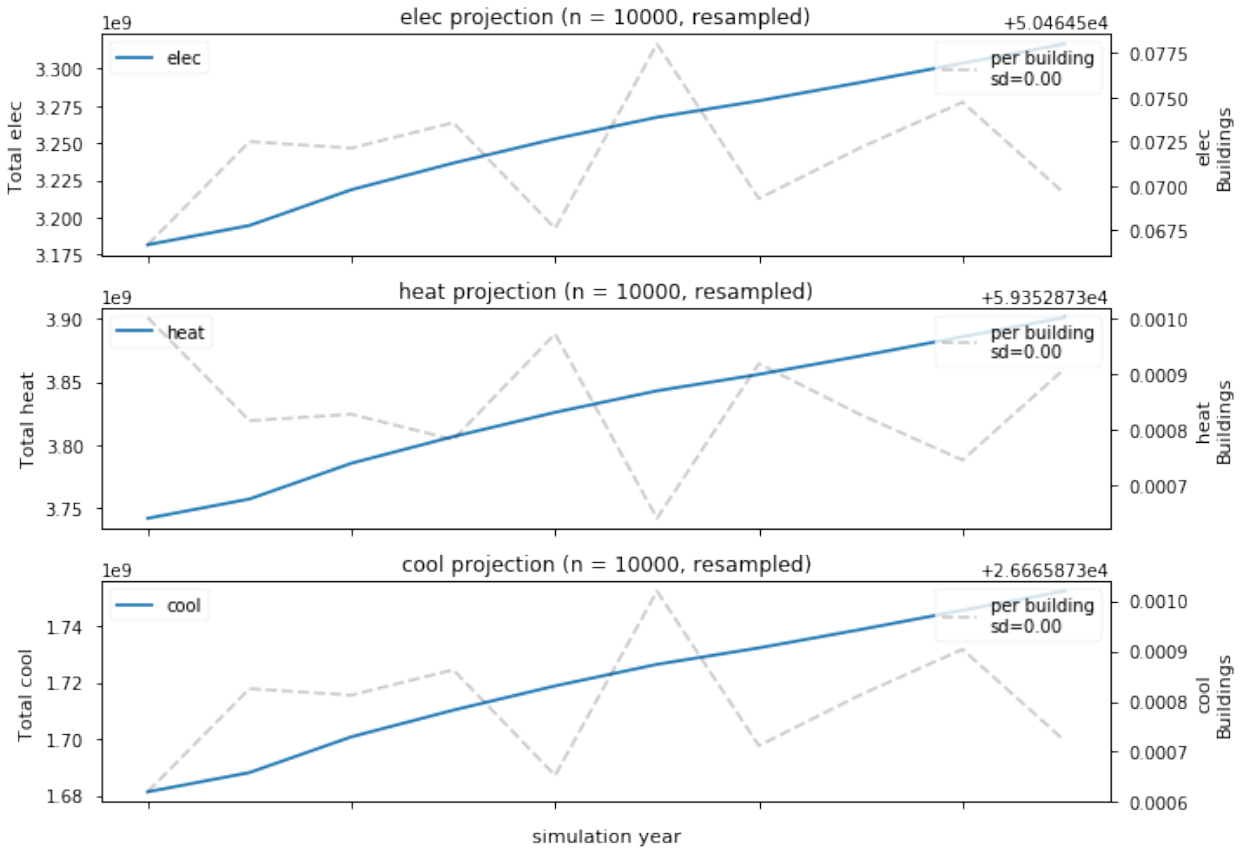
Run model

```
In [ ]: fw = run_calibrated_model(
            table_model,
            verbose = verbose,
            project = typ,
            census_file = census_file,
            year = benchmark_year,
            population_size = False,
            name = '{}_{}'.format(model_name, iterations),
            iterations = iterations,
            align_census = False,
            drop_col_survey = drop_col_survey
        )
```

Plot results

```
In [12]: reweighted_survey = 'data/survey_{}_{}'.format(model_name, iterations)
```

```
In [13]: data = plot_data_projection(
             reweighted_survey, ['elec','heat','cool'], "{}, {}".format(iterations, typ),
             start_year = 2016, end_year = 2025, aspect_ratio = 4,
             benchmark_year = False, unit = "building")
```



Brussels. Step 2.c Model Internal Validation

```
In [1]: import datetime; print(datetime.datetime.now())
```

```
2018-04-09 10:47:47.530334
```

Notebook abstract

Model internal validation.

```
In [2]: from smum.microsim.util_plot import plot_error
```

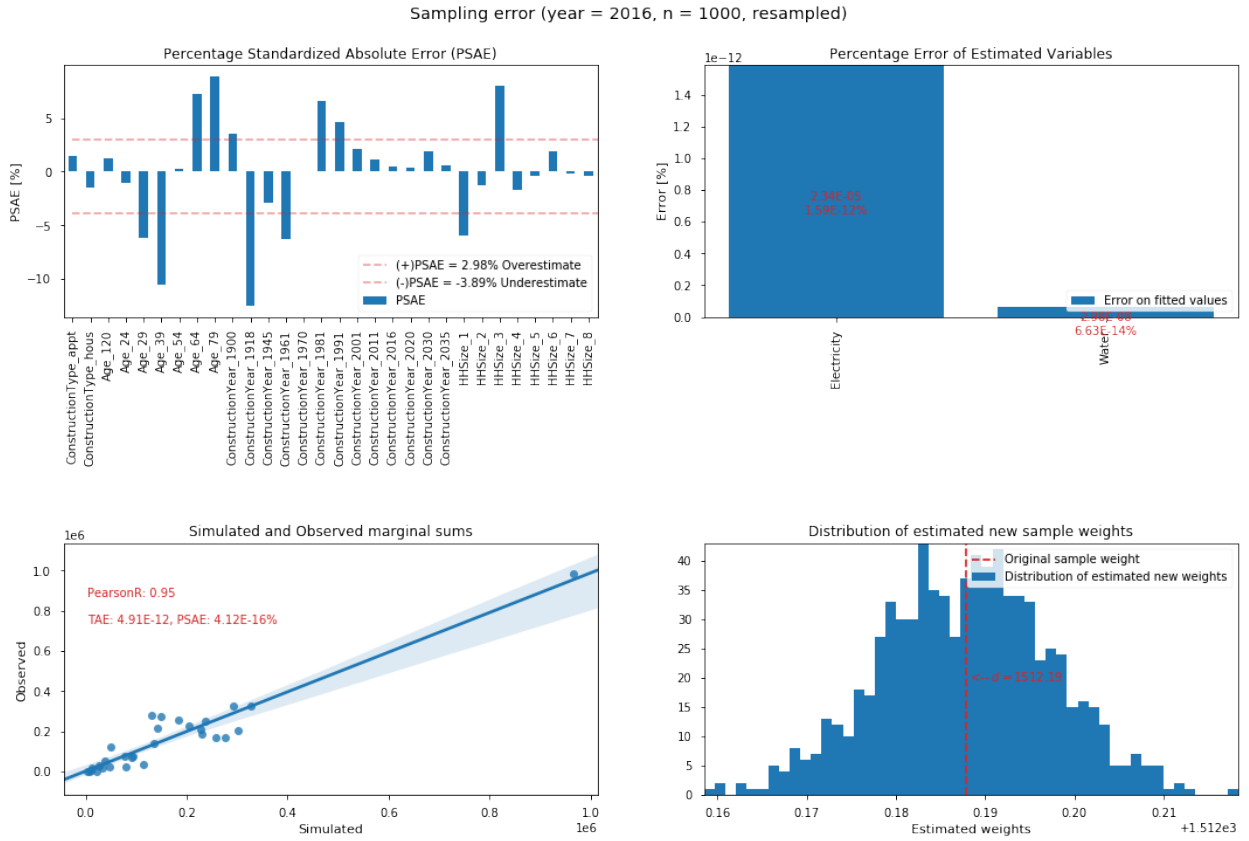
Residential Sector

```
In [3]: iterations = 1000
        benchmark_year = 2016
        census_file = 'data/benchmarks_be_year_bias3_climate.csv'
        typ = 'resampled'
        model_name = 'Brussels_Electricity_Water_projected_dynamic_{}_bias'.format(typ)
        survey_file = 'data/survey_{}_{}_{}.csv'.format(model_name, iterations, benchmark_year)
```

```
In [4]: import pandas as pd
        input_census = pd.read_csv(census_file, index_col = 0)
```

```
In [5]: sufix = 4
        census_file = './temp/calibrated_benchmarks_{}.csv'.format(sufix)
        REC = plot_error(
            survey_file,
```

```
        census_file,
        "{}, {}".format(iterations, typ),
        year = benchmark_year,
        #force_fit = False,
        pop = input_census.loc[benchmark_year, 'pop'],
        add_cols = input_census.loc[benchmark_year, ['Electricity', 'Water']],
        skip = ['e_sqm', 'w_Income', 'e_CDD', 'e_HDD'],
        fit_cols = ['Electricity', 'Water']
    )
```



Brussels. Step 3.a Defining simple transition scenarios

```
In [1]: import datetime; print(datetime.datetime.now())
```

```
2018-04-09 10:49:38.925958
```

Notebook Abstract:

The following notebook describes the process to construct simple **transition scenarios**.

The transition scenarios are define as **efficiency** rates induced by **technology development** or **behavioral** changes. These rates can be used as proxies for all types of efficiency improvements.

In order to define transition scenarios the model need the following information:

1. A technology **penetration rate**. This defines the share of the population *adopting* the technology. The model uses `sampling rules` for the selection of the population adopting this technology.
2. Development of **efficiency rates**. This define the actual technology development rate.

Import libraries

```
In [2]: from smum.microsim.run import transition_rate
        from smum.microsim.util_plot import plot_transition_rate
        from smum.microsim.run import reduce_consumption
```

```
/usr/lib/python3.6/site-packages/h5py-2.7.1-py3.6-linux-x86_64.egg/h5py/__init__.py:36: FutureWarning
  from ._conv import register_converters as _register_converters
```

In order to compute the transition scenarios we make use of three modules of the `urbanmetabolism` library:

1. `growth_rate`. This module will return a vector with linear **growth rates** given a starting and end rate.
2. `plot_growth_rate`. This a simple function to **visualize** the defined growth rates.
3. `reduce_consumption`. This function creates **new samples** with reduced consumption levels for the selected selection of the population.

Define simple population selection rules

```
In [3]: sampling_rules = {
            "w_ConstructionType == 'ConstructionType_appt'": 10,
            "e_sqm < 70": 10,
            "w_Income > 13650": 10,
        }
```

Part of the scenario development is to identified which section of the population will adopt the new technology. The model defined this by a **sampling probability**. This probability is initially define as a uniform distribution (i.e. each individual on the sample has equal probability of being selected). A scenario is defined by allocating new sampling probabilities to a section of the population, by defining sampling rules. The sampling rules are passes to the query function of a pandas `DataFrame`.

On the example above, all individuals with a `Income` larger of equal to 180 000 Philippine Pesos are 30 times more likely to adopt the technology that the rest of the population. The sampling probabilities will sum up. This means that an individual with `Income >= 180000` and living on an urban area `e_Urban == 'Urbanity_Urban'` is 60 times more likely to be selected (i.e. adopt a new technology) that other individuals.

Initial sample data

```
In [4]: import pandas as pd
        file_name = "data/survey_Brussels_Electricity_Water_projected_dynamic_resampled_bias_1000_{}
        sample_survey = pd.read_csv(file_name.format(2016), index_col=0)
        sample_survey.head()
```

Out[4]:

index	w_ConstructionType	w_Age	w_ConstructionYear	w_HHSize	\
0	0	ConstructionType_hous	Age_54	ConstructionYear_1961	HHSize_3
1	1	ConstructionType_hous	Age_79	ConstructionYear_1961	HHSize_1
2	2	ConstructionType_hous	Age_29	ConstructionYear_1961	HHSize_3
3	3	ConstructionType_appt	Age_39	ConstructionYear_1945	HHSize_6
4	4	ConstructionType_hous	Age_120	ConstructionYear_1981	HHSize_1

	e_sqm	w_Income	Water	Electricity	w	wf
0	73.029173	13563.144795	35.477657	1424.293271	1512.187817	1512.194014
1	73.020808	13651.074297	26.794691	1403.605964	1512.187817	1512.188162
2	73.291210	13789.696751	37.695896	1054.339670	1512.187817	1512.178937
3	73.366936	13721.455861	48.304569	1335.381382	1512.187817	1512.183478
4	72.718209	13649.570622	26.428328	950.361211	1512.187817	1512.188263

The `reduce_consumption` module will use as input the samples created by the MCMC algorithm and select specific sections of the population to reduce their consumption levels.

The input data is the constructed proxy sample data. Depending on the simulation type (reweight/resample) the input data is a single file containing the weights for each simulation year (reweight) or individual samples for each simulation year (resample).

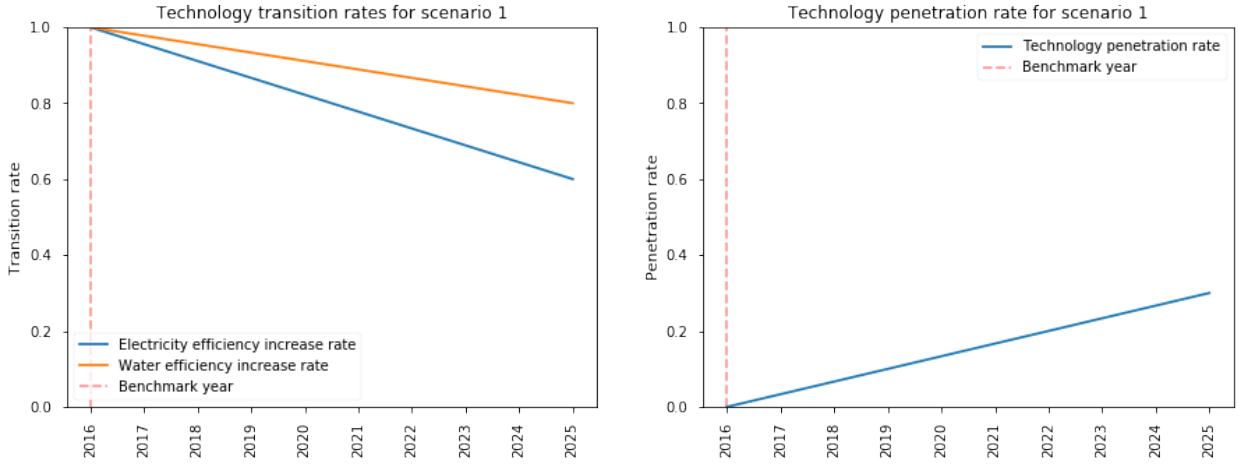
Most of the variables of the sample data is formatted as categorical data. The sample data shown above presents individual records with the predefine simulation variables as well as the computed **Electricity** and **Water** consumption levels. The sample also contains two sets of weights: **w** and **wf**. The `w` weights correspond to the weights assign by the MCMC algorithm (uniform distributed) while the `wf` (final weights) are the weights computed by the GREGWT algorithm. The `reduce_consumption` function will use the `wf` weights for the selection of individuals.

Define growth rates

```
In [5]: Elec  = transition_rate(0, 0.4, default_start=2016, default_end=2025)
        Water = transition_rate(0, 0.2, default_start=2016, default_end=2025)
        pr    = transition_rate(0, 0.3, default_start=2016, default_end=2025)
```

With help of the `growth_rate()` function we define the **efficiency growth rate** and the technology penetration rate. For the technology penetration rate we define the start year to be equal to the benchmark year (2016). The function will automatically include the necessary zeros at the beginning of the growth rate vector. The function `plot_growth_rate()` allow us to **visualize** the predefined efficiency growth rates and the technology growth rates.

```
In [6]: plot_transition_rate(
            {"Technology penetration rate": pr,
             "Electricity efficiency increase rate": Elec,
             "Water efficiency increase rate": Water},
             "scenario 1", start_year=2016, end_year=2025)
```



Reduce consumption

The actual modifications on the sample is performed by the `reduce_consumption()` function. The function requires as input the following parameters:

1. A base file name for the samples (in case of implementing the resample method).
2. The sample year (in this case generated within the loop via `range(2010, 2031)`.
3. The penetration rate for the sample year (iterated from vector `pr`).

4. The predefined `sampling_rules`.
5. A dictionary containing the efficiency rates for specific variables. (in this case for Electricity and Water).
6. A name for the scenario.

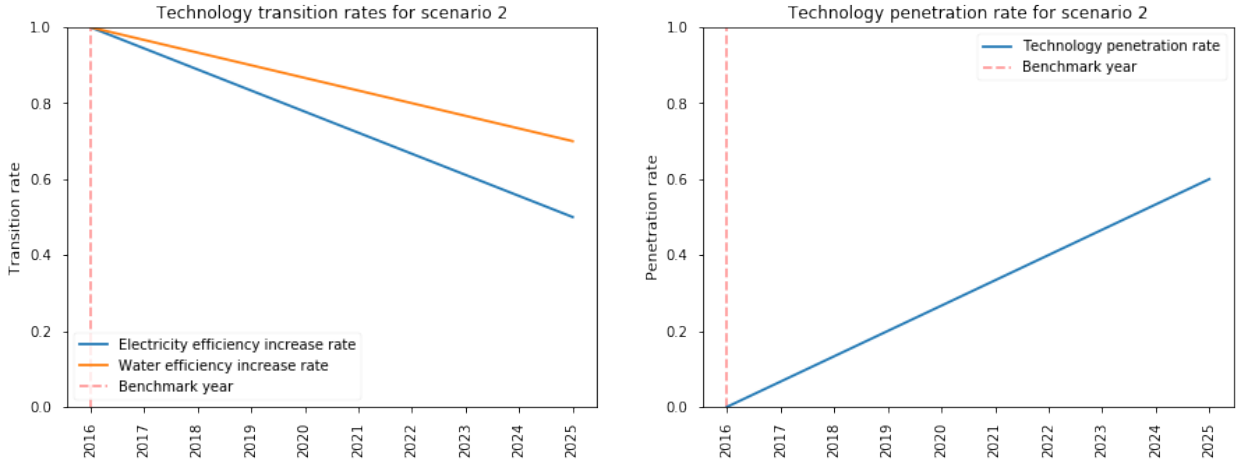
```
In [7]: for y, p, elec, water in zip(range(2016, 2026), pr, Elec, Water):
            _ = reduce_consumption(
                file_name,
                y, p, sampling_rules,
                {'Electricity':elec, 'Water':water},
                scenario_name = "scenario 1")
```

```
00.00%      both       reduction; efficiency rate 00.00%;              year 2016 and penetration rate
00.15%   Electricity   reduction; efficiency rate 04.44%;              year 2017 and penetration rate
00.07%      Water      reduction; efficiency rate 02.22%;              year 2017 and penetration rate
00.59%   Electricity   reduction; efficiency rate 08.89%;              year 2018 and penetration rate
00.30%      Water      reduction; efficiency rate 04.44%;              year 2018 and penetration rate
01.31%   Electricity   reduction; efficiency rate 13.33%;              year 2019 and penetration rate
00.67%      Water      reduction; efficiency rate 06.67%;              year 2019 and penetration rate
02.33%   Electricity   reduction; efficiency rate 17.78%;              year 2020 and penetration rate
01.18%      Water      reduction; efficiency rate 08.89%;              year 2020 and penetration rate
03.66%   Electricity   reduction; efficiency rate 22.22%;              year 2021 and penetration rate
01.83%      Water      reduction; efficiency rate 11.11%;              year 2021 and penetration rate
05.28%   Electricity   reduction; efficiency rate 26.67%;              year 2022 and penetration rate
02.65%      Water      reduction; efficiency rate 13.33%;              year 2022 and penetration rate
07.12%   Electricity   reduction; efficiency rate 31.11%;              year 2023 and penetration rate
03.59%      Water      reduction; efficiency rate 15.56%;              year 2023 and penetration rate
09.34%   Electricity   reduction; efficiency rate 35.56%;              year 2024 and penetration rate
04.77%      Water      reduction; efficiency rate 17.78%;              year 2024 and penetration rate
11.81%   Electricity   reduction; efficiency rate 40.00%;              year 2025 and penetration rate
05.96%      Water      reduction; efficiency rate 20.00%;              year 2025 and penetration rate
```

```
In [8]: Elec  = transition_rate(0.0, 0.5, default_start=2016, default_end=2025)
        Water = transition_rate(0.0, 0.3, default_start=2016, default_end=2025)
        pr    = transition_rate(0.0, 0.6, default_start=2016, default_end=2025)
```

By modifying the growht rates we can create different scenarios.

```
In [9]: plot_transition_rate(
            {"Technology penetration rate": pr,
             "Electricity efficiency increase rate": Elec,
             "Water efficiency increase rate": Water},
             "scenario 2", start_year=2016, end_year=2025)
```



```
In [10]: for y, p, elec, water in zip(range(2016, 2026), pr, Elec, Water):
             _ = reduce_consumption(
                 file_name,
                 y, p, sampling_rules,
                 {'Electricity':elec, 'Water':water},
                 scenario_name = "scenario 2")
```

```
00.00%      both       reduction; efficiency rate 00.00%;             year 2016 and penetration rate
00.36%   Electricity   reduction; efficiency rate 05.56%;             year 2017 and penetration rate
00.22%      Water      reduction; efficiency rate 03.33%;             year 2017 and penetration rate
01.47%   Electricity   reduction; efficiency rate 11.11%;             year 2018 and penetration rate
00.89%      Water      reduction; efficiency rate 06.67%;             year 2018 and penetration rate
03.29%   Electricity   reduction; efficiency rate 16.67%;             year 2019 and penetration rate
02.01%      Water      reduction; efficiency rate 10.00%;             year 2019 and penetration rate
05.84%   Electricity   reduction; efficiency rate 22.22%;             year 2020 and penetration rate
03.56%      Water      reduction; efficiency rate 13.33%;             year 2020 and penetration rate
09.16%   Electricity   reduction; efficiency rate 27.78%;             year 2021 and penetration rate
05.51%      Water      reduction; efficiency rate 16.67%;             year 2021 and penetration rate
13.22%   Electricity   reduction; efficiency rate 33.33%;             year 2022 and penetration rate
07.97%      Water      reduction; efficiency rate 20.00%;             year 2022 and penetration rate
17.88%   Electricity   reduction; efficiency rate 38.89%;             year 2023 and penetration rate
10.79%      Water      reduction; efficiency rate 23.33%;             year 2023 and penetration rate
23.45%   Electricity   reduction; efficiency rate 44.44%;             year 2024 and penetration rate
14.29%      Water      reduction; efficiency rate 26.67%;             year 2024 and penetration rate
29.69%   Electricity   reduction; efficiency rate 50.00%;             year 2025 and penetration rate
17.92%      Water      reduction; efficiency rate 30.00%;             year 2025 and penetration rate
```

Brussels. Step 3.b. Visualize transition scenarios

```
In [1]: import datetime; print(datetime.datetime.now())
```

```
2018-04-09 11:47:45.683429
```

Notebook Abstract:

The following notebook **visualize** the the simple transition scenarios by plotting the total consumption over all simulation years and the per-capita consumption rate. Depending on the define scenarios the per-capita consumption rate can be maintained constant. The per-capita consumption value is computed as total consumption divided by population size.

Import libraries

```
In [2]: from smum.microsim.util_plot import plot_data_projection
```

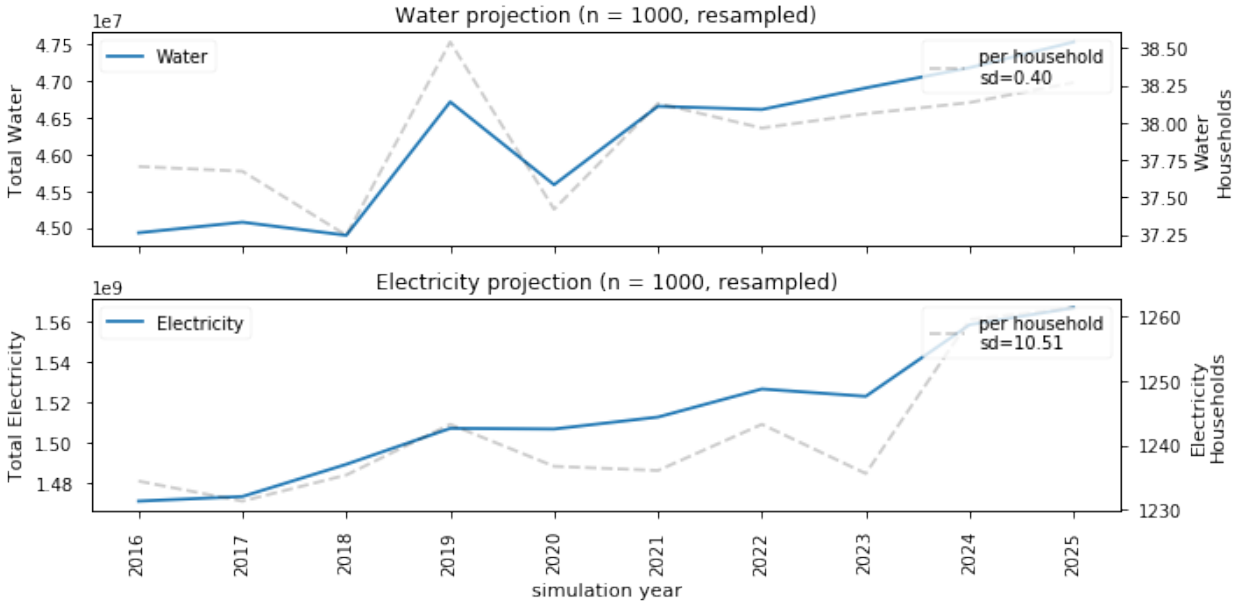
The visualization is performed with help of the module function `plot_data_projection()`.

Global variables

```
In [3]: iterations = 1000
        typ = 'resampled'
        model_name = "Brussels_Electricity_Water_projected_dynamic_{}_bias".format(typ)
        reweighted_survey = 'data/survey_{}_{}'.format(model_name, iterations)
```

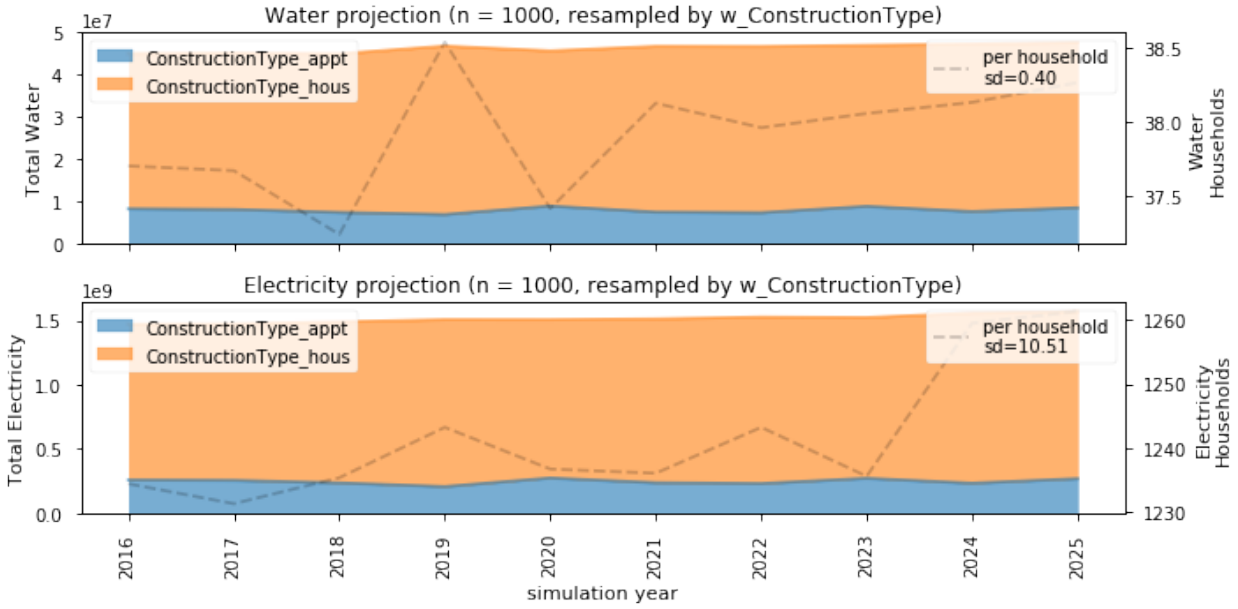
Base scenario

```
In [14]: var = ['Water', 'Electricity']
         data = plot_data_projection(
             reweighted_survey, var, "{}, {}".format(iterations, typ),
             benchmark_year=False, start_year=2016, end_year=2025
         )
```



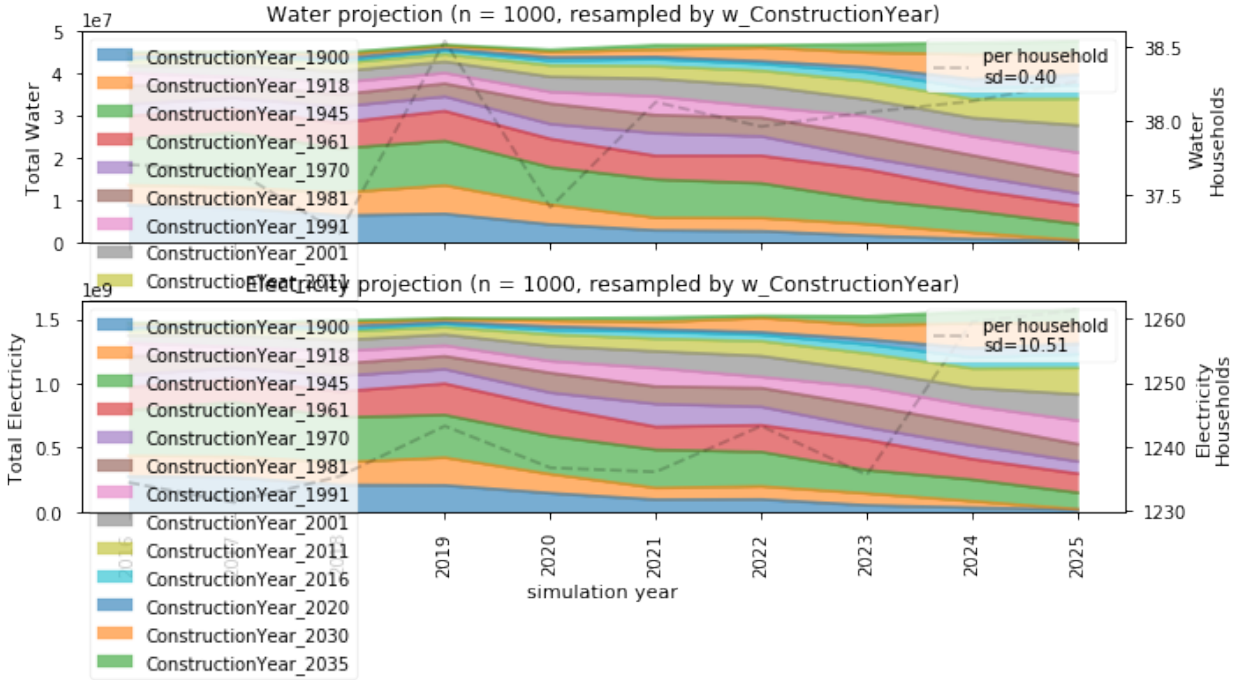
Base scenario grouped by construction type

```
In [5]: var = ['Water', 'Electricity']
        groupby = 'w_ConstructionType'
        data = plot_data_projection(
            reweighted_survey, var, "{}, {} by {}".format(iterations, typ, groupby),
            benchmark_year = False, start_year=2016, end_year=2025,
            groupby = groupby
        )
```



Base scenario grouped by construction year

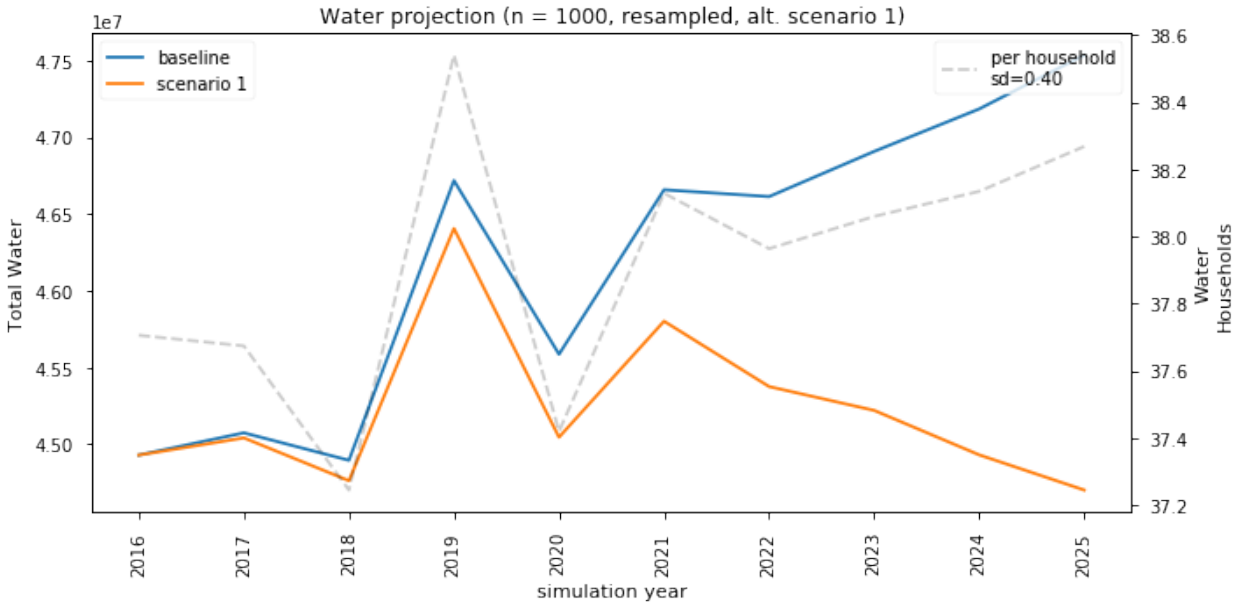
```
In [6]: var = ['Water', 'Electricity']
        groupby = 'w_ConstructionYear'
        data = plot_data_projection(
            reweighted_survey, var, "{}, {} by {}".format(iterations, typ, groupby),
            benchmark_year = False, start_year=2016, end_year=2025,
            groupby = groupby
        )
```

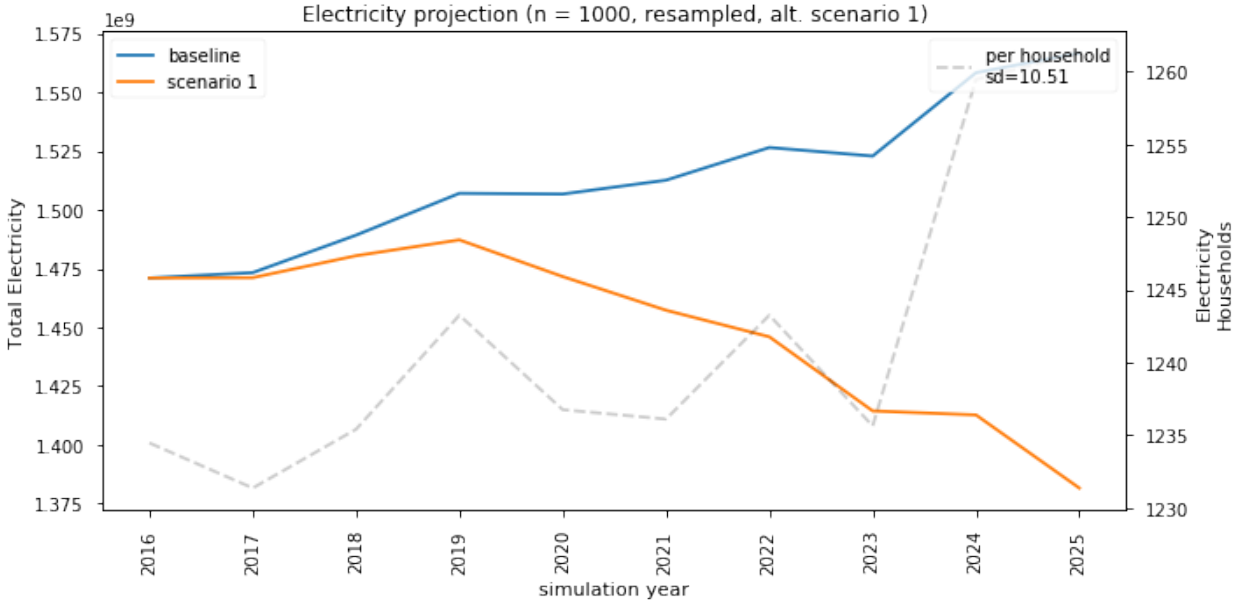


Scenario 1 compared to base scenario

```
In [7]: import numpy as np
        pr = [i for i in np.linspace(0, 0.3, num=10)]
        scenario_name = 'scenario 1'
```

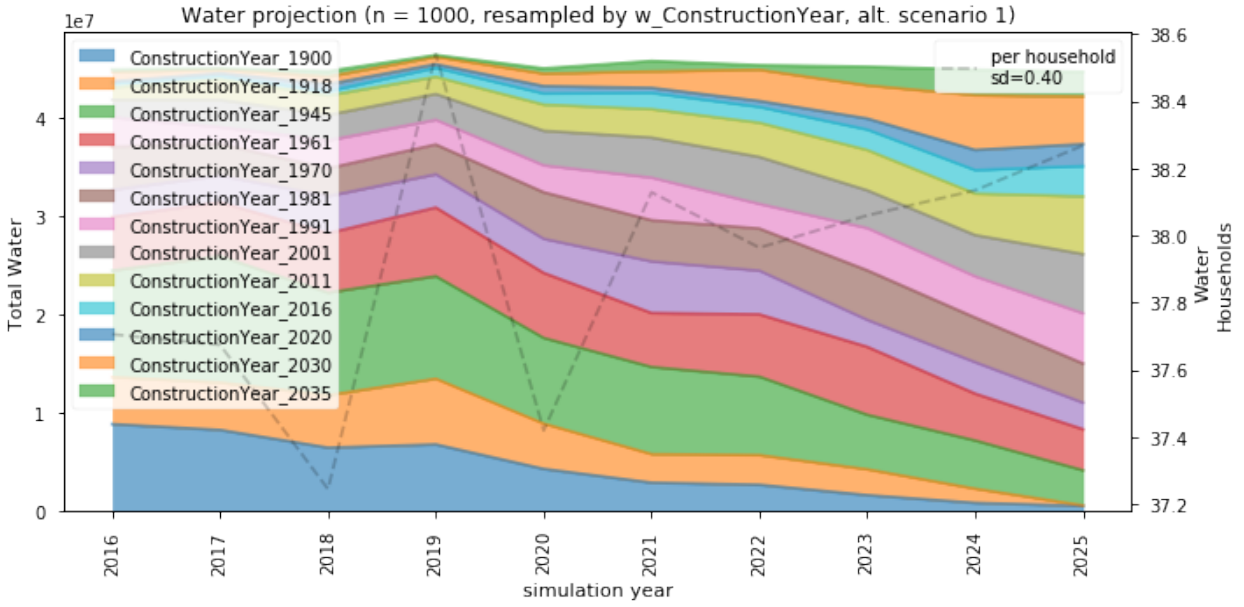
```
In [8]: variables = ['Water', 'Electricity']
        for var in variables:
            var = [var]
            data = plot_data_projection(
                reweighted_survey, var, "{}, {}, alt. scenario 1".format(iterations, typ),
                benchmark_year=False,  start_year=2016, end_year=2025,
                pr = pr, scenario_name = scenario_name,
                aspect_ratio = 2,
                )
```

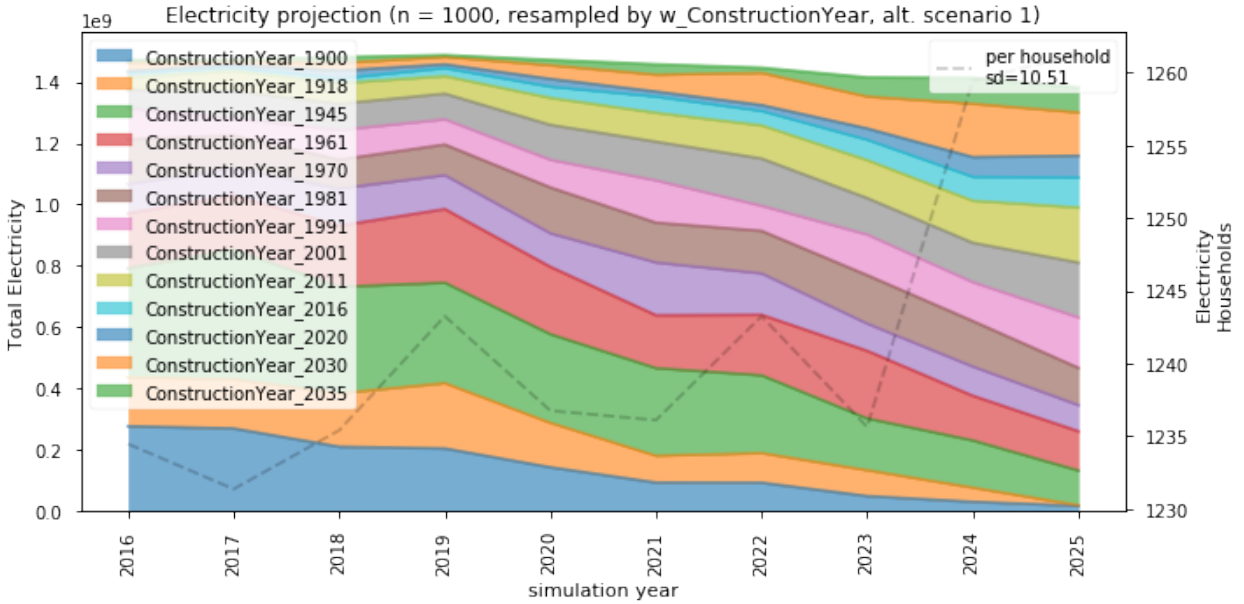




Scenario 1 grouped by construction year

```
In [9]: variables = ['Water', 'Electricity']
        groupby = 'w_ConstructionYear'
        for var in variables:
            var = [var]
            data = plot_data_projection(
                reweighted_survey, var, "{}, {} by {}, alt. scenario 1".format(iterations, typ, group
                benchmark_year=False,  start_year=2016, end_year=2025,
                pr = pr, scenario_name = scenario_name,
                groupby = groupby,
                aspect_ratio = 2,
            )
```



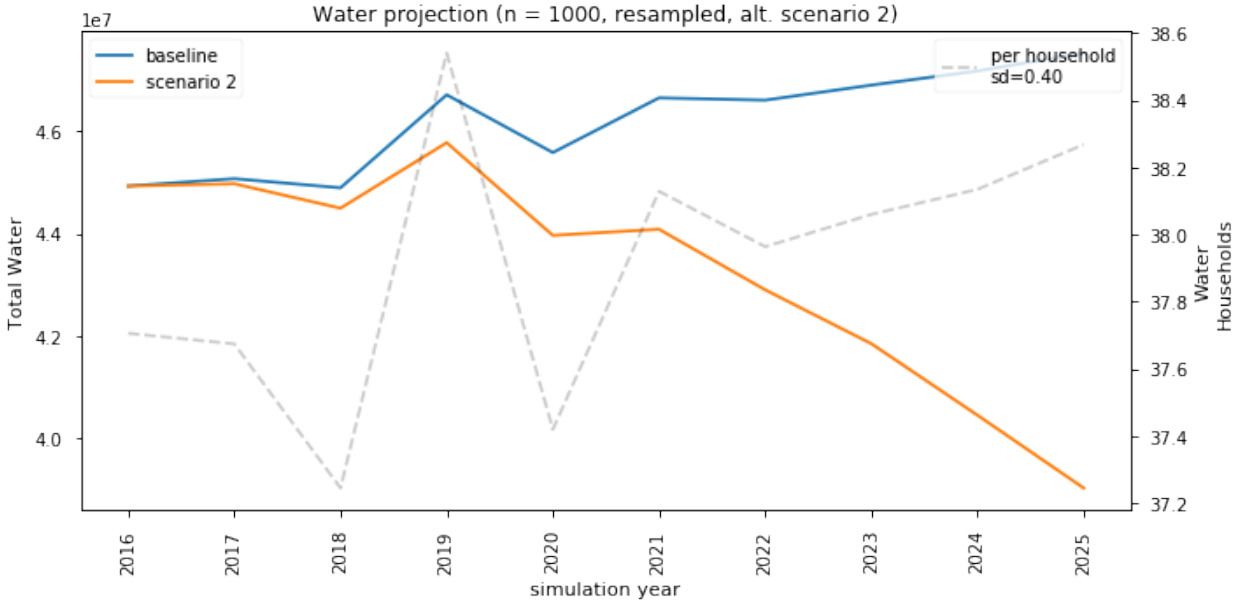


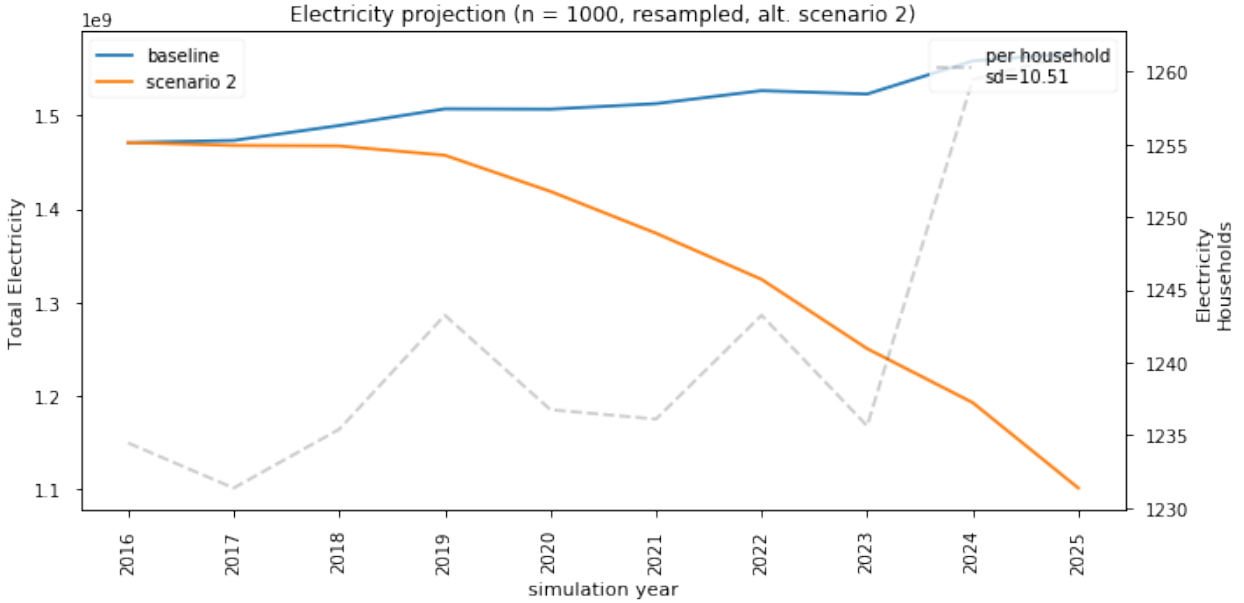

```
In [10]: import numpy as np
```

```
        pr = [i for i in np.linspace(0, 0.6, num=10)]
        scenario_name = 'scenario 2'
```

Scenario 2 compared to base scenario

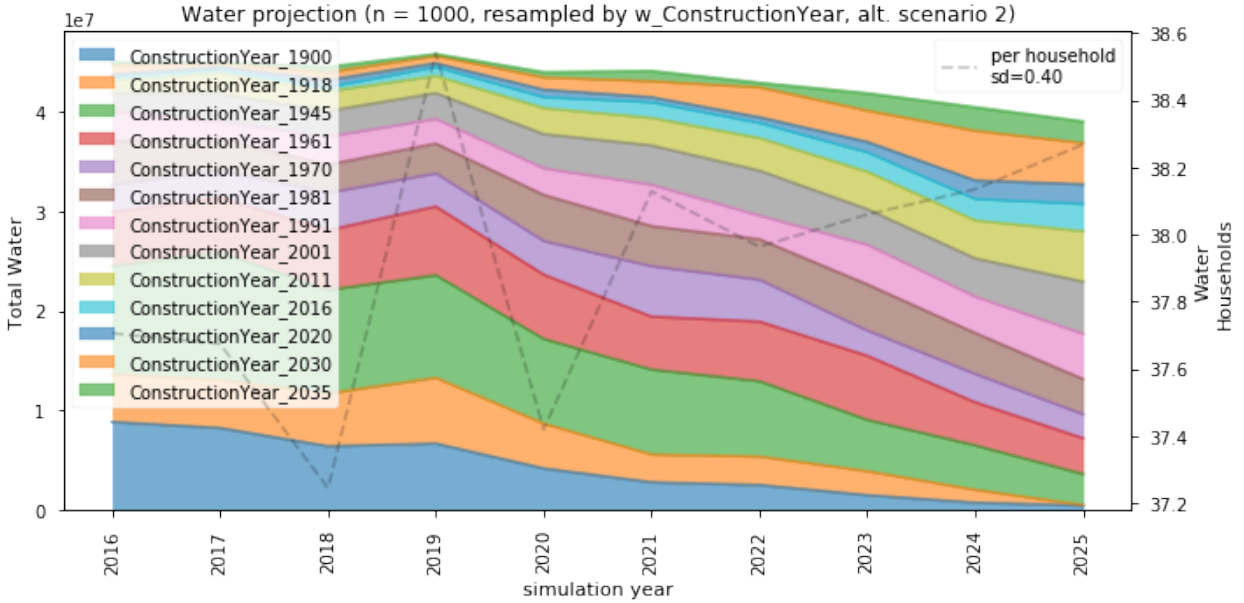
```
In [11]: variables = ['Water', 'Electricity']
         for var in variables:
             var = [var]
             data = plot_data_projection(
                 reweighted_survey, var, "{}, {}, alt. scenario 2".format(iterations, typ),
                 benchmark_year=False, start_year=2016, end_year=2025,
                 pr = pr, scenario_name = scenario_name,
                 aspect_ratio = 2,
             )
```

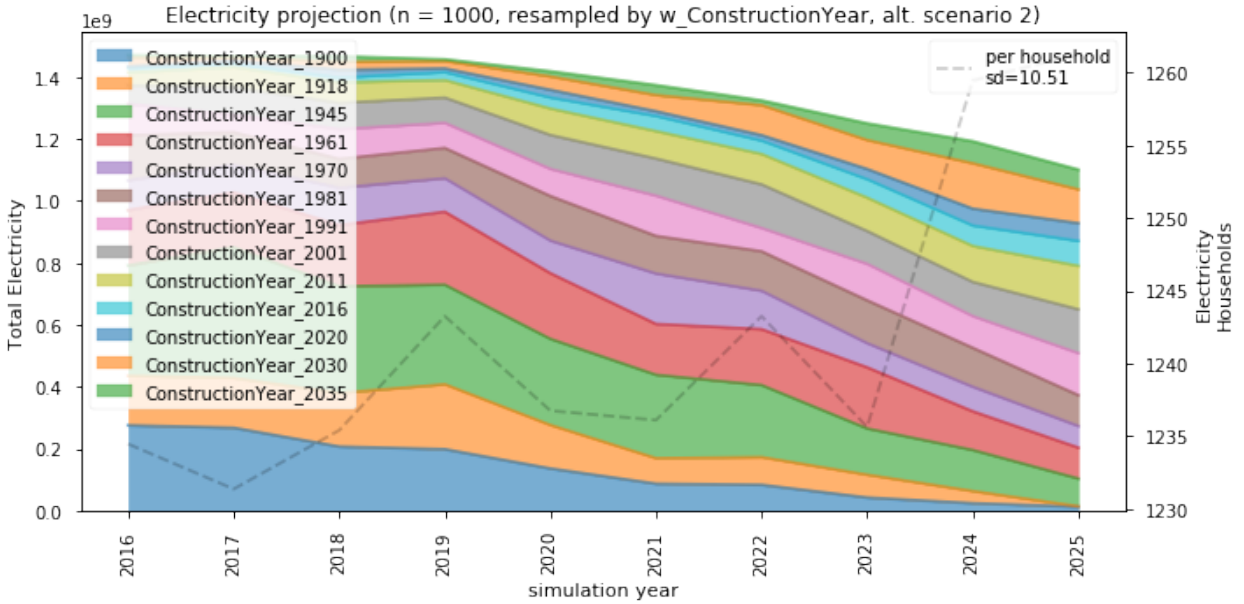




Scenario 2 grouped by construction year

```
In [12]: variables = ['Water', 'Electricity']
         groupby = 'w_ConstructionYear'
         for var in variables:
             var = [var]
             data = plot_data_projection(
                 reweighted_survey, var, "{}, {} by {}, alt. scenario 2".format(iterations, typ, grou
                 benchmark_year=False, start_year=2016, end_year=2025,
                 pr = pr, scenario_name = scenario_name,
                 groupby = groupby,
                 aspect_ratio = 2,
                 )
```



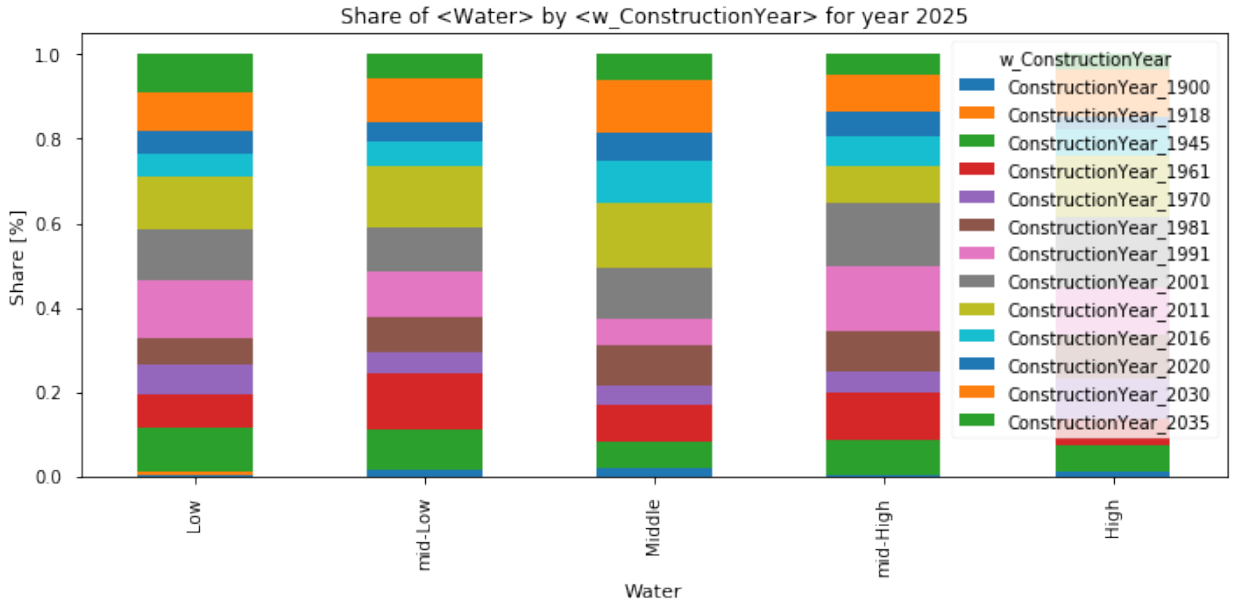


Scenario 2 grouped by construction year, cross tabulation

```
In [15]: from smum.microsim.util_plot import cross_tab
```

```
In [16]: a = 'Water'
         b = 'w_ConstructionYear'
         ct = cross_tab(a, b, 2025, reweighted_survey + "_{}_scenario 2_0.60.csv", split_a = True)
```

```
data saved as: data/Water_w_ConstructionYear_2025.xlsx
```



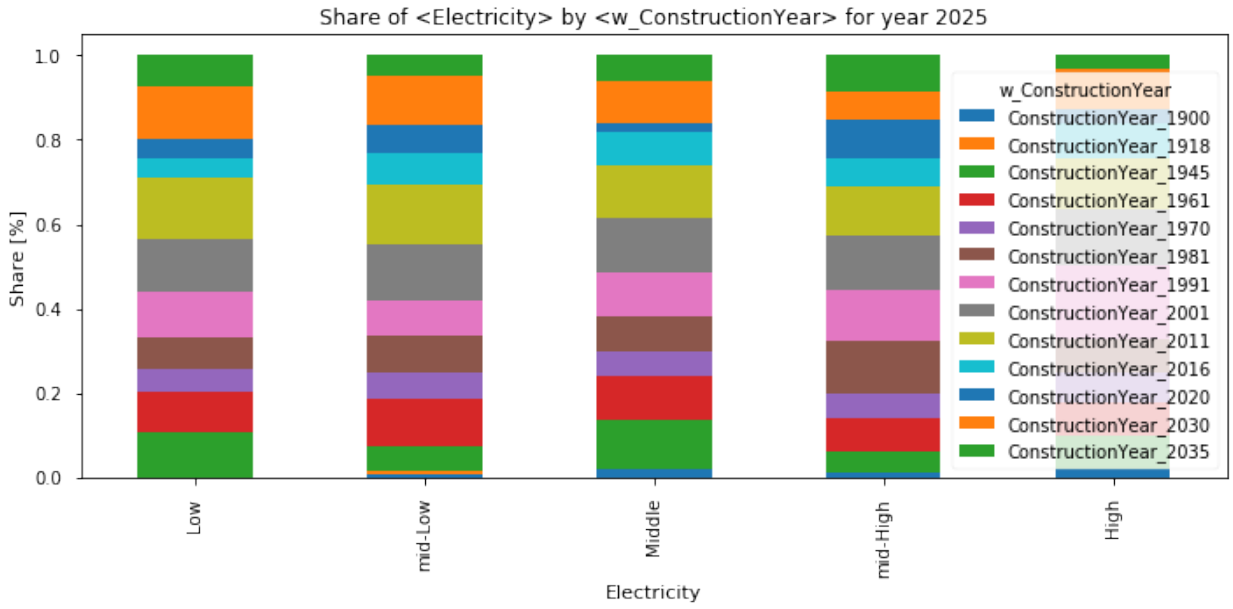

```
In [17]: ct
```

```
Out[17]: w_ConstructionYear  ConstructionYear_1900  ConstructionYear_1918  \
         Water
         Low                                756.00                2107.00
         mid-Low                           3783.00                    NaN
         Middle                            5432.62                  38.32
         mid-High                           910.64                    NaN
         High                              1989.63                    NaN

         w_ConstructionYear  ConstructionYear_1945  ConstructionYear_1961  \
         Water
         Low                                31728.00               24695.00
         mid-Low                            23941.59               33933.60
         Middle                             15585.17               21938.42
         mid-High                           17586.04               23999.04
         High                               14134.40               13426.42

         w_ConstructionYear  ConstructionYear_1970  ConstructionYear_1981  \
         Water
         Low                                20596.00               19762.00
         mid-Low                            12107.27               21273.51
         Middle                             11685.87               25276.22
         mid-High                           10350.21               20567.05
         High                               20346.68               22532.30

         w_ConstructionYear  ConstructionYear_1991  ConstructionYear_2001  \
         Water
```

```
Low                                41657.00                36786.00
mid-Low                            27826.15                25649.13
Middle                             16113.10                30999.13
mid-High                           33363.96                31628.68
High                               24775.92                35835.83

w_ConstructionYear  ConstructionYear_2011  ConstructionYear_2016  \
Water
Low                                37786.00                15789.00
mid-Low                            36460.21                15256.25
Middle                             38580.44                26178.14
mid-High                           19203.35                14848.76
High                               31014.19                13740.42

w_ConstructionYear  ConstructionYear_2020  ConstructionYear_2030  \
Water
Low                                17284.00                27518.00
mid-Low                            11792.58                25899.59
Middle                             16685.87                32066.15
mid-High                           12146.85                18881.18
High                                6450.16                24354.21

w_ConstructionYear  ConstructionYear_2035
Water
Low                                27313.31
mid-Low                            14275.55
Middle                             15668.16
mid-High                           10398.89
High                                7430.19
```

In [18]:
```
a = 'Electricity'
b = 'w_ConstructionYear'
ct = cross_tab(a, b, 2025, reweighted_survey + "_{}_scenario 2_0.60.csv", split_a = True)
```

```
data saved as: data/Electricity_w_ConstructionYear_2025.xlsx
```



In [19]:
```
ct
```

Out[19]:
```
w_ConstructionYear  ConstructionYear_1900  ConstructionYear_1918  \
```

```
Electricity
Low                                  NaN                   NaN
mid-Low                          2656.00               2107.00
Middle                           4184.00                   NaN
mid-High                         2015.63                   NaN
High                             4016.26                 38.32

w_ConstructionYear  ConstructionYear_1945  ConstructionYear_1961  \
Electricity
Low                                33418.00               29831.00
mid-Low                            16546.00               34296.00
Middle                             25195.41               21668.80
mid-High                           10714.45               15683.31
High                               17101.34               16513.37

w_ConstructionYear  ConstructionYear_1970  ConstructionYear_1981  \
Electricity
Low                                16378.00               24107.00
mid-Low                            18102.00               26041.00
Middle                             12685.21               17660.15
mid-High                           12347.43               25470.56
High                               15573.39               16132.37

w_ConstructionYear  ConstructionYear_1991  ConstructionYear_2001  \
Electricity
Low                                34345.00               38718.00
mid-Low                            25023.00               39102.00
Middle                             22405.50               27688.46
mid-High                           24511.52               27061.47
High                               37451.12               28328.85

w_ConstructionYear  ConstructionYear_2011  ConstructionYear_2016  \
Electricity
Low                                44712.00               15356.31
mid-Low                            42224.00               21567.00
Middle                             27039.38               17181.52
mid-High                           23394.93               13715.08
High                               25673.88               17992.67

w_ConstructionYear  ConstructionYear_2020  ConstructionYear_2030  \
Electricity
Low                                13317.00               39565.00
mid-Low                            20670.00               34543.00
Middle                              4476.64               20975.16
mid-High                           19022.35               13591.65
High                                6873.47               20044.31

w_ConstructionYear  ConstructionYear_2035
Electricity
Low                                22970.00
mid-Low                            14090.00
Middle                             13303.24
mid-High                           17715.70
High                                7007.16
```

Brussels. Step 3.a.2 Defining simple transition scenarios NR

```
In [1]: import datetime; print(datetime.datetime.now())
```

```
2018-04-09 10:51:22.613149
```

Notebook Abstract:

The following notebook describes the process to construct simple **transition scenarios**.

The transition scenarios are define as **efficiency** rates induced by **technology development** or **behavioral** changes. These rates can be used as proxies for all types of efficiency improvements.

In order to define transition scenarios the model need the following information:

1. A technology **penetration rate**. This defines the share of the population *adopting* the technology. The model uses `sampling rules` for the selection of the population adopting this technology.
2. Development of **efficiency rates**. This define the actual technology development rate.

Import libraries

```
In [2]: from smum.microsim.run import transition_rate
        from smum.microsim.util_plot import plot_transition_rate
        from smum.microsim.run import reduce_consumption
```

```
/usr/lib/python3.6/site-packages/h5py-2.7.1-py3.6-linux-x86_64.egg/h5py/__init__.py:36: FutureWarning
  from ._conv import register_converters as _register_converters
```

In order to compute the transition scenarios we make use of three modules of the `urbanmetabolism` library:

1. `growth_rate`. This module will return a vector with linear **growth rates** given a starting and end rate.
2. `plot_growth_rate`. This a simple function to **visualize** the defined growth rates.
3. `reduce_consumption`. This function creates **new samples** with reduced consumption levels for the selected selection of the population.

Define simple population selection rules

```
In [3]: sampling_rules = {
            "e_BuildingSqm >= 2000": 10,
        }
```

Part of the scenario development is to identified which section of the population will adopt the new technology. The model defined this by a **sampling probability**. This probability is initially define as a uniform distribution (i.e. each individual on the sample has equal probability of being selected). A scenario is defined by allocating new sampling probabilities to a section of the population, by defining sampling rules. The sampling rules are passes to the query function of a pandas `DataFrame`.

On the example above, all individuals with a `Income` larger of equal to 180 000 Philippine Pesos are 30 times more likely to adopt the technology that the rest of the population. The sampling probabilities will sum up. This means that an individual with `Income >= 180000` and living on an urban area `e_Urban == 'Urbanity_Urban'` is 60 times more likely to be selected (i.e. adopt a new technology) that other individuals.

Initial sample data

```
In [4]: import pandas as pd
        typ = 'resampled'
        file_name = "data/survey_Brussels_NonResidentialElectricity_wbias_projected_dynamic_resampled
        sample_survey = pd.read_csv(file_name.format(2016), index_col=0)
        sample_survey.head()
```

```
Out[4]: index  e_BuildingSqm          elec           heat           cool       w  \
        0      0          3600.0      0.000000       0.000000       0.000000  6.3048
        1      1           525.0  66742.068354   94458.852064       0.000000  6.3048
        2      2           525.0  66742.068354   31486.284021  140893.979372  6.3048
        3      3            39.0      0.000000  117168.862933   65124.328243  6.3048
        4      4           525.0      0.000000   15185.242330   46964.659791  6.3048

                wf
        0   9.524821
        1  13.568532
        2  13.568532
        3  14.207635
        4  13.568532
```

The `reduce_consumption` module will use as input the samples created by the MCMC algorithm and select specific sections of the population to reduce their consumption levels.

The input data is the constructed proxy sample data. Depending on the simulation type (reweight/resample) the input data is a single file containing the weights for each simulation year (reweight) or individual samples for each simulation year (resample).

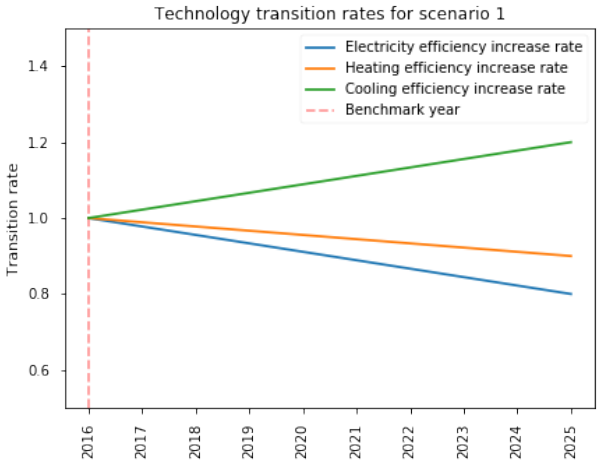
Most of the variables of the sample data is formatted as categorical data. The sample data shown above presents individual records with the predefine simulation variables as well as the computed **Electricity** and **Water** consumption levels. The sample also contains two sets of weights: **w** and **wf**. The `w` weights correspond to the weights assign by the MCMC algorithm (uniform distributed) while the `wf` (final weights) are the weights computed by the GREGWT algorithm. The `reduce_consumption` function will use the `wf` weights for the selection of individuals.

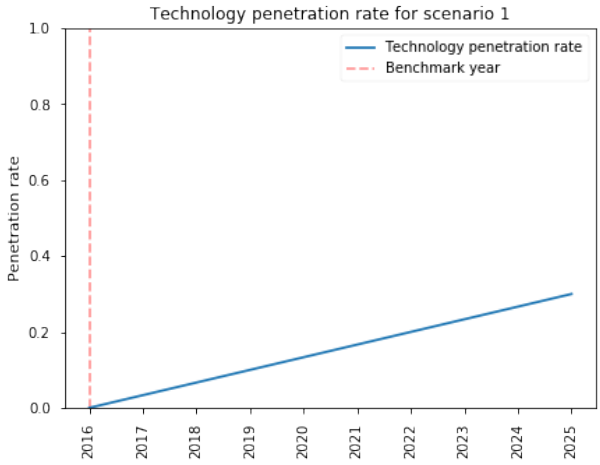
Define growth rates

```
In [5]: elec = transition_rate(0, 0.2, default_start=2016, default_end=2025)
        heat = transition_rate(0, 0.1, default_start=2016, default_end=2025)
        cool = transition_rate(0, -0.2, default_start=2016, default_end=2025)
        pr   = transition_rate(0, 0.3, default_start=2016, default_end=2025)
```

With help of the `growth_rate()` function we define the **efficiency growth rate** and the technology penetration rate. For the technology penetration rate we define the start year to be equal to the benchmark year (2016). The function will automatically include the necessary zeros at the beginning of the growth rate vector. The function `plot_growth_rate()` allow us to **visualize** the predefined efficiency growth rates and the technology growth rates.

```
In [6]: plot_transition_rate(
            {"Technology penetration rate": pr,
             "Electricity efficiency increase rate": elec,
             "Heating efficiency increase rate": heat,
             "Cooling efficiency increase rate": cool},
             "scenario 1", ylim_a=(0.5, 1.5), start_year=2016, end_year=2025)
```





Reduce consumption

The actual modifications on the sample is performed by the `reduce_consumption()` function. The function requires as input the following parameters:

1. A base file name for the samples (in case of implementing the resample method).
2. The sample year (in this case generated within the loop via `range(2010, 2031)`.
3. The penetration rate for the sample year (iterated from vector `pr`).
4. The predefined `sampling_rules`.
5. A dictionary containing the efficiency rates for specific variables. (in this case for Electricity and Water).
6. A name for the scenario.

```
In [7]: for y, p, e, h, c in zip(range(2016, 2026), pr, elec, heat, cool):
            _ = reduce_consumption(
                file_name,
                y, p, sampling_rules,
                {'elec':e, 'heat':h, 'cool':c},
                scenario_name = "scenario 1")
```

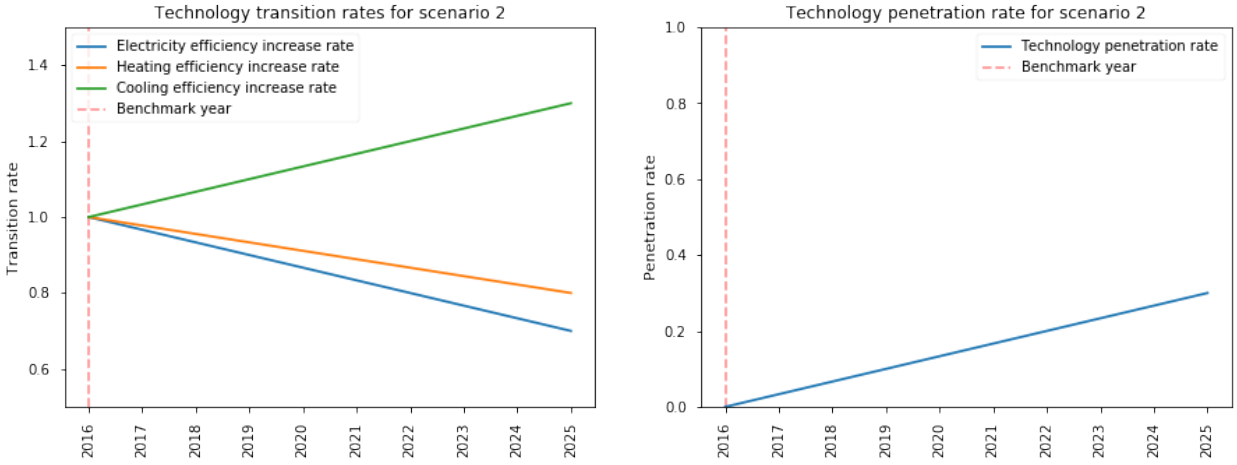
```
00.00%      both       reduction; efficiency rate 00.00%;            year 2016 and penetration rate
00.02%      elec       reduction; efficiency rate 02.22%;            year 2017 and penetration rate
00.04%      heat       reduction; efficiency rate 01.11%;            year 2017 and penetration rate
-0.03%      cool       reduction; efficiency rate -2.22%;            year 2017 and penetration rate
00.08%      elec       reduction; efficiency rate 04.44%;            year 2018 and penetration rate
00.11%      heat       reduction; efficiency rate 02.22%;            year 2018 and penetration rate
-0.25%      cool       reduction; efficiency rate -4.44%;            year 2018 and penetration rate
00.18%      elec       reduction; efficiency rate 06.67%;            year 2019 and penetration rate
00.26%      heat       reduction; efficiency rate 03.33%;            year 2019 and penetration rate
-0.52%      cool       reduction; efficiency rate -6.67%;            year 2019 and penetration rate
00.31%      elec       reduction; efficiency rate 08.89%;            year 2020 and penetration rate
00.43%      heat       reduction; efficiency rate 04.44%;            year 2020 and penetration rate
-0.85%      cool       reduction; efficiency rate -8.89%;            year 2020 and penetration rate
00.51%      elec       reduction; efficiency rate 11.11%;            year 2021 and penetration rate
00.66%      heat       reduction; efficiency rate 05.56%;            year 2021 and penetration rate
-1.15%      cool       reduction; efficiency rate -11.11%;            year 2021 and penetration rat
00.74%      elec       reduction; efficiency rate 13.33%;            year 2022 and penetration rate
01.16%      heat       reduction; efficiency rate 06.67%;            year 2022 and penetration rate
-1.95%      cool       reduction; efficiency rate -13.33%;            year 2022 and penetration rat
```

```
01.06%      elec       reduction; efficiency rate 15.56%;              year 2023 and penetration rate
01.51%      heat       reduction; efficiency rate 07.78%;              year 2023 and penetration rate
-2.38%      cool       reduction; efficiency rate -15.56%;              year 2023 and penetration rat
01.27%      elec       reduction; efficiency rate 17.78%;              year 2024 and penetration rate
01.63%      heat       reduction; efficiency rate 08.89%;              year 2024 and penetration rate
-2.70%      cool       reduction; efficiency rate -17.78%;              year 2024 and penetration rat
01.73%      elec       reduction; efficiency rate 20.00%;              year 2025 and penetration rate
02.32%      heat       reduction; efficiency rate 10.00%;              year 2025 and penetration rate
-3.81%      cool       reduction; efficiency rate -20.00%;              year 2025 and penetration rat
```

```
In [8]: elec = transition_rate(0, 0.3, default_start=2016, default_end=2025)
        heat = transition_rate(0, 0.2, default_start=2016, default_end=2025)
        cool = transition_rate(0, -0.3, default_start=2016, default_end=2025)
        pr   = transition_rate(0, 0.3, default_start=2016, default_end=2025)
```

By modifying the growht rates we can create different scenarios.

```
In [9]: plot_transition_rate(
            {"Technology penetration rate": pr,
             "Electricity efficiency increase rate": elec,
             "Heating efficiency increase rate": heat,
             "Cooling efficiency increase rate": cool},
             "scenario 2", ylim_a=(0.5, 1.5), start_year=2016, end_year=2025)
```




```
In [10]: for y, p, e, h, c in zip(range(2016, 2026), pr, elec, heat, cool):
             _ = reduce_consumption(
                 file_name,
                 y, p, sampling_rules,
                 {'elec':e, 'heat':h, 'cool':c},
                 scenario_name = "scenario 2")
```

```
00.00%      both       reduction; efficiency rate 00.00%;              year 2016 and penetration rate
00.03%      elec       reduction; efficiency rate 03.33%;              year 2017 and penetration rate
00.06%      heat       reduction; efficiency rate 02.22%;              year 2017 and penetration rate
-0.06%      cool       reduction; efficiency rate -3.33%;              year 2017 and penetration rate
00.11%      elec       reduction; efficiency rate 06.67%;              year 2018 and penetration rate
00.32%      heat       reduction; efficiency rate 04.44%;              year 2018 and penetration rate
-0.25%      cool       reduction; efficiency rate -6.67%;              year 2018 and penetration rate
00.29%      elec       reduction; efficiency rate 10.00%;              year 2019 and penetration rate
00.48%      heat       reduction; efficiency rate 06.67%;              year 2019 and penetration rate
-0.71%      cool       reduction; efficiency rate -10.00%;              year 2019 and penetration rat
00.43%      elec       reduction; efficiency rate 13.33%;              year 2020 and penetration rate
00.61%      heat       reduction; efficiency rate 08.89%;              year 2020 and penetration rate
-0.96%      cool       reduction; efficiency rate -13.33%;              year 2020 and penetration rat
```

```
00.76%      elec       reduction; efficiency rate 16.67%;              year 2021 and penetration rate
01.11%      heat       reduction; efficiency rate 11.11%;              year 2021 and penetration rate
-2.30%      cool       reduction; efficiency rate -16.67%;              year 2021 and penetration rat
01.15%      elec       reduction; efficiency rate 20.00%;              year 2022 and penetration rate
01.62%      heat       reduction; efficiency rate 13.33%;              year 2022 and penetration rate
-3.18%      cool       reduction; efficiency rate -20.00%;              year 2022 and penetration rat
01.51%      elec       reduction; efficiency rate 23.33%;              year 2023 and penetration rate
02.45%      heat       reduction; efficiency rate 15.56%;              year 2023 and penetration rate
-3.70%      cool       reduction; efficiency rate -23.33%;              year 2023 and penetration rat
02.00%      elec       reduction; efficiency rate 26.67%;              year 2024 and penetration rate
02.78%      heat       reduction; efficiency rate 17.78%;              year 2024 and penetration rate
-4.34%      cool       reduction; efficiency rate -26.67%;              year 2024 and penetration rat
02.53%      elec       reduction; efficiency rate 30.00%;              year 2025 and penetration rate
03.50%      heat       reduction; efficiency rate 20.00%;              year 2025 and penetration rate
-5.70%      cool       reduction; efficiency rate -30.00%;              year 2025 and penetration rat
```

Brussels. Step 3.b.2 Visualize transition scenarios NR

```
In [1]: import datetime; print(datetime.datetime.now())
```

```
2018-04-09 10:53:30.322370
```

Notebook Abstract:

The following notebook **visualize** the the simple transition scenarios by plotting the total consumption over all simulation years and the per-capita consumption rate. Depending on the define scenarios the per-capita consumption rate can be maintained constant. The per-capita consumption value is computed as total consumption divided by population size.

Import libraries

```
In [2]: from smum.microsim.util_plot import plot_data_projection
```

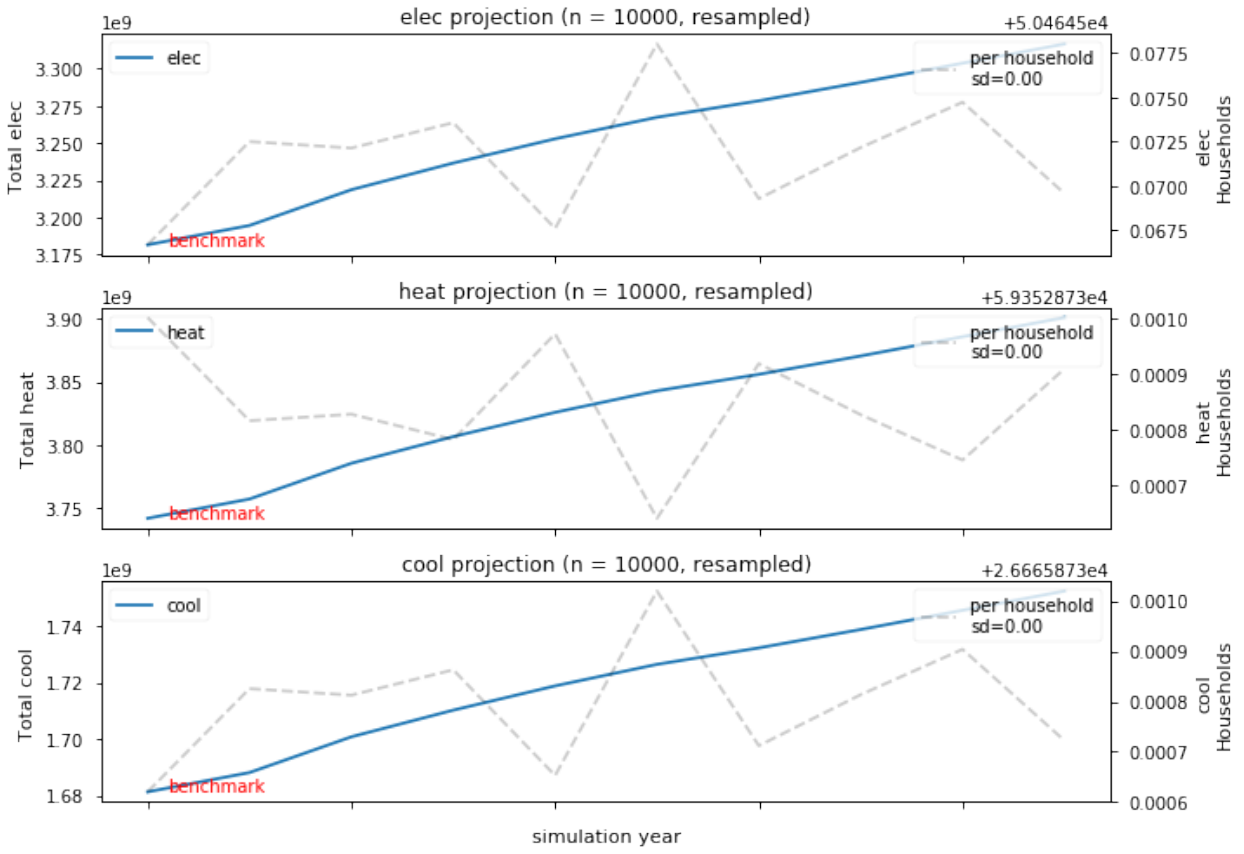
The visualization is performed with help of the module function `plot_data_projection()`.

Global variables

```
In [3]: iterations = 10000
        typ = 'resampled'
        model_name = "Brussels_NonResidentialElectricity_wbias_projected_dynamic_{}".format(typ)
        reweighted_survey = 'data/survey_{}_{}'.format(model_name, iterations)
```

Base scenario

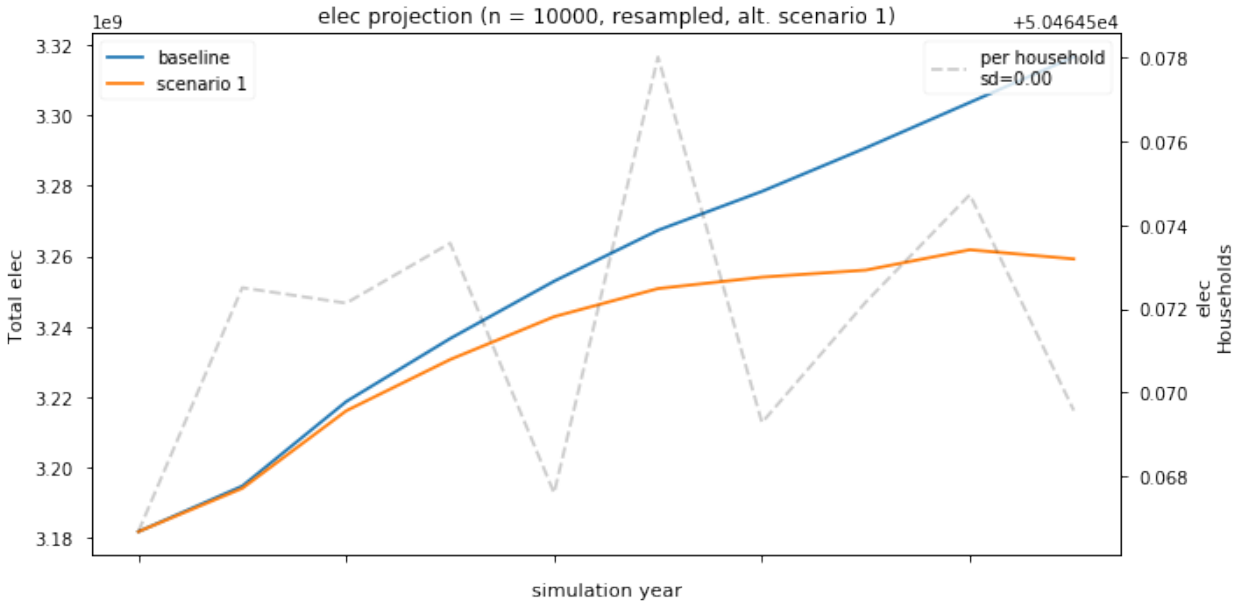
```
In [4]: var = ['elec', 'heat', 'cool']
        data = plot_data_projection(
            reweighted_survey, var, "{}, {}".format(iterations, typ),
            benchmark_year=2016, start_year=2016, end_year=2025
        )
```

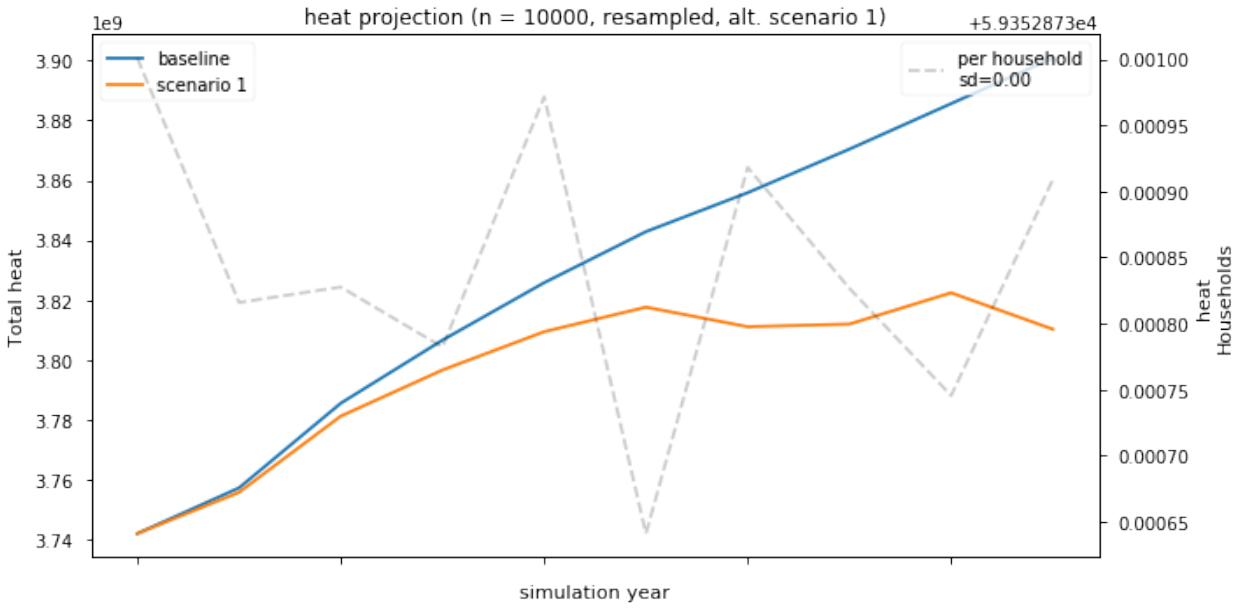


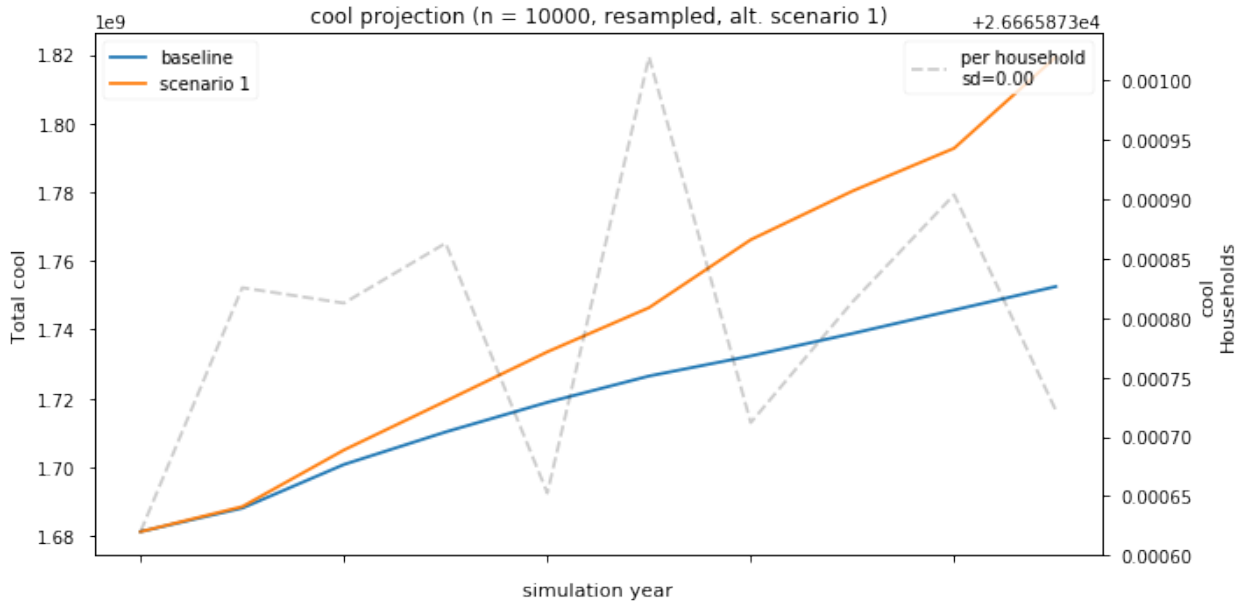
Scenario 1 compared to base scenario

```
In [5]: import numpy as np
        pr = [i for i in np.linspace(0, 0.3, num=10)]
        scenario_name = 'scenario 1'
```

```
In [6]: variables = ['elec', 'heat', 'cool']
        for var in variables:
            var = [var]
            data = plot_data_projection(
                reweighted_survey, var, "{}, {}, alt. scenario 1".format(iterations, typ),
                benchmark_year=False,  start_year=2016, end_year=2025,
                pr = pr, scenario_name = scenario_name,
                aspect_ratio = 2,
            )
```

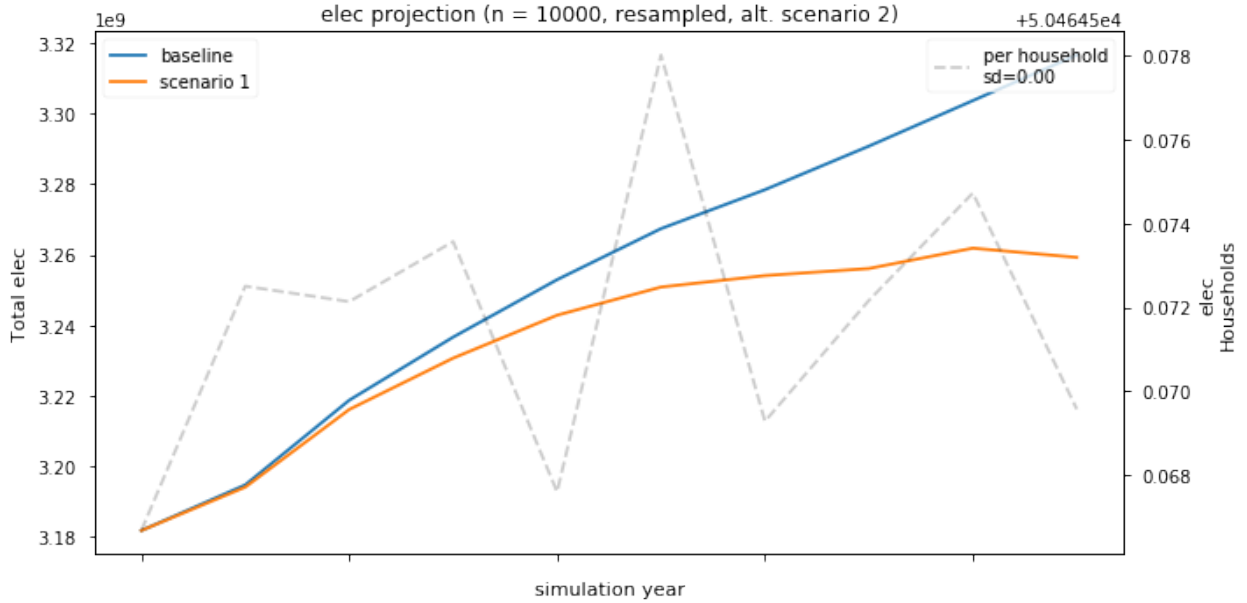

elec projection (n = 10000, resampled, alt. scenario 1)
1e9
+5.04645e4
baseline
scenario 1
per household
sd=0.00
Total elec
elec
Households
simulation year

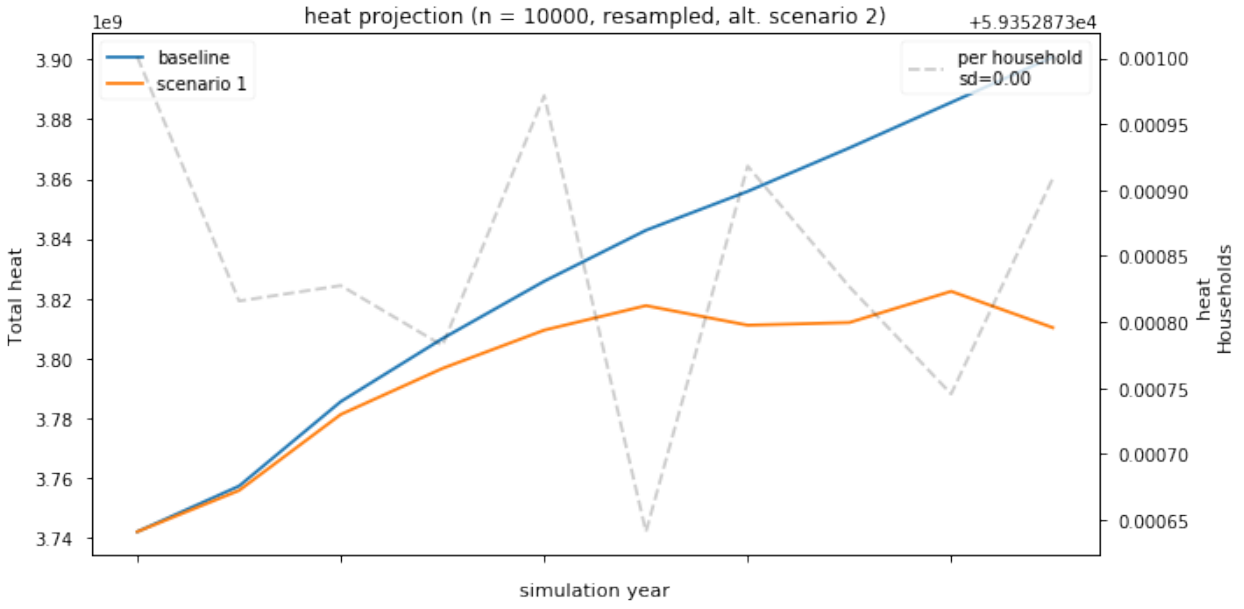

heat projection (n = 10000, resampled, alt. scenario 1)
1e9
+5.9352873e4
baseline
scenario 1
per household
sd=0.00
Total heat
heat
Households
simulation year

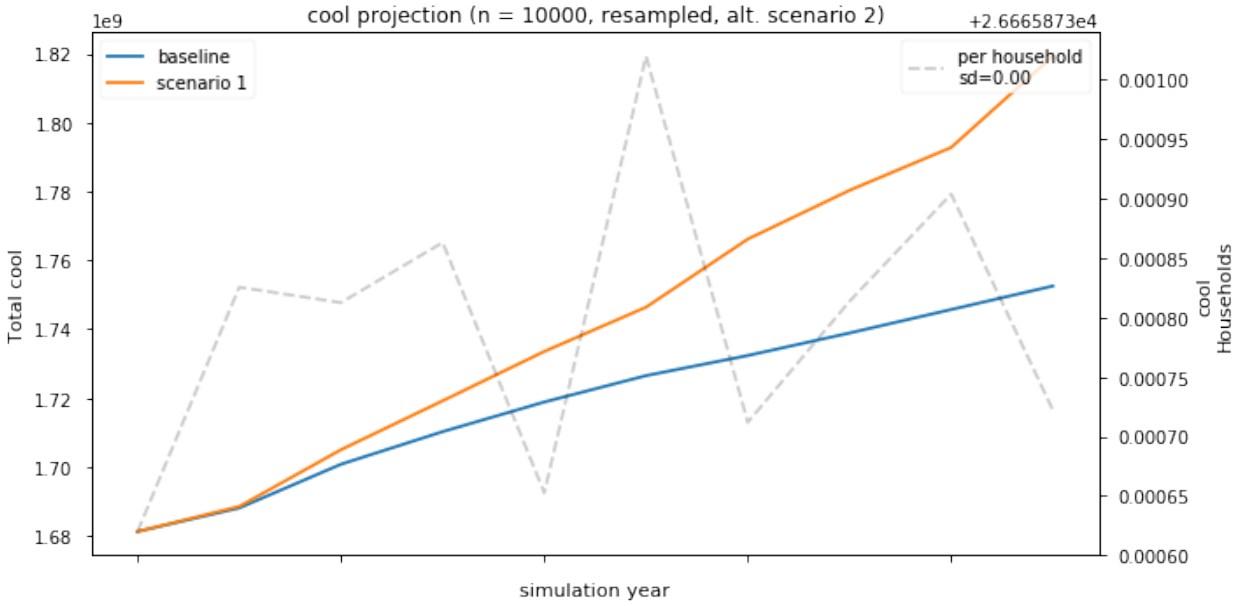


Scenario 2 compared to base scenario

```
In [7]: variables = ['elec', 'heat', 'cool']
        for var in variables:
            var = [var]
            data = plot_data_projection(
                reweighted_survey, var, "{}, {}, alt. scenario 2".format(iterations, typ),
                benchmark_year=False, start_year=2016, end_year=2025,
                pr = pr, scenario_name = scenario_name,
                aspect_ratio = 2,
            )
```







4.2.4 Advanced Example 2: Brussels Materials

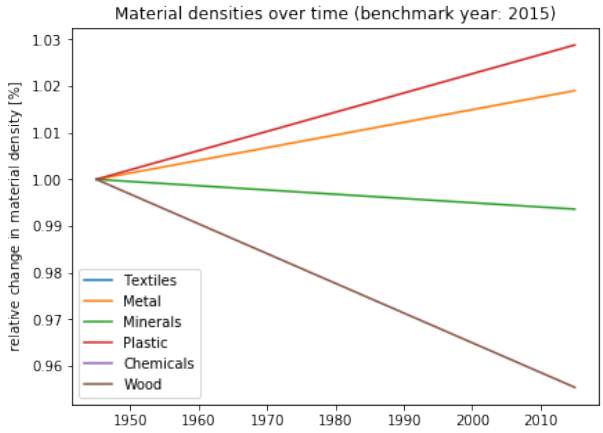
Brussels Mat. Step 1.a Material Densities

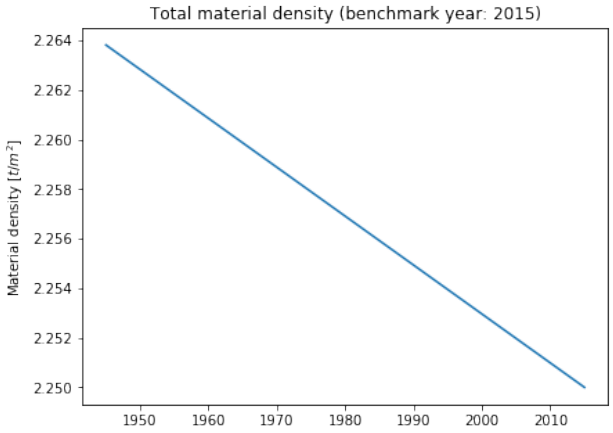
The following code describes the process for the computation of material densities by m2 of floor space. These material densities are used for the estimation of total material demands for the building stock.

The following file describes the definition of building types for the residential rector for Brussels.

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline
```

```
In [2]: data = pd.DataFrame(
            [np.nan, 0.03, 2.18, 0.02, np.nan, 0.02],
            index=['Textiles', 'Metal', 'Minerals', 'Plastic', 'Chemicals', 'Wood'])
        data.columns = [2013]
        data.index.name = 'Material type'

In [3]: data_years = pd.DataFrame(index=['Textiles', 'Metal', 'Minerals', 'Plastic', 'Chemicals', 'Wo

In [4]: end_year = 2016
        str_year = 1945
        step = 5
        diff = int((end_year - str_year -1) / step + 1)
        rate = np.asarray([0, -0.0006, 0.015, -0.0006, 0, 0.001])

In [5]: for e, year in enumerate(range(str_year, end_year, step)):
            e += 1
            a = (diff - e) / diff
            data_years.loc[:, year] = data.loc[:, 2013] + a * rate

In [6]: data_years

Out[6]: 1945      1950      1955      1960      1965      1970      1975  \
        Textiles        NaN       NaN       NaN       NaN       NaN       NaN       NaN
        Metal      0.029440  0.029480  0.02952  0.029560  0.029600  0.02964  0.029680
        Minerals   2.194000  2.193000  2.19200  2.191000  2.190000  2.18900  2.188000
        Plastic    0.019440  0.019480  0.01952  0.019560  0.019600  0.01964  0.019680
        Chemicals       NaN       NaN       NaN       NaN       NaN       NaN       NaN
        Wood       0.020933  0.020867  0.02080  0.020733  0.020667  0.02060  0.020533

                       1980      1985      1990      1995      2000      2005      2010  \
        Textiles        NaN       NaN       NaN       NaN       NaN       NaN       NaN
        Metal      0.029720  0.02976  0.029800  0.029840  0.02988  0.029920  0.029960
        Minerals   2.187000  2.18600  2.185000  2.184000  2.18300  2.182000  2.181000
        Plastic    0.019720  0.01976  0.019800  0.019840  0.01988  0.019920  0.019960
        Chemicals       NaN       NaN       NaN       NaN       NaN       NaN       NaN
        Wood       0.020467  0.02040  0.020333  0.020267  0.02020  0.020133  0.020067

                    2015
        Textiles     NaN
        Metal       0.03
        Minerals    2.18
        Plastic     0.02
        Chemicals    NaN
        Wood        0.02

In [7]: fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15,5))
        data_years.div(data_years.loc[:, 1945], axis=0).T.plot(ax=ax1)
        ax1.set_ylabel('relative change in material density $[\%]$')
        ax1.set_title('Material densities over time (benchmark year: 2015)')
        data_years.sum().plot(ax=ax2)
        ax2.set_ylabel('Material density $[t/m^2]$')
        ax2.set_title('Total material density (benchmark year: 2015)');
```






```
In [8]: typ = [
            "Detached",
            "Semi-detached",
            "Terraced",
            "Apartment - enclosed",
            "Apartment - exposed"]
```

```
In [9]: def formatnumbers(x):
            x = str(x).replace(',', '.')
            x = float(x)
            return x
```

```
In [10]: tabula = pd.read_excel('TABULA.xlsx', sheet_name='TABULA-2',
                                usecols=[0,2,3,4,5,6,7,8,9], index_col=[2,0,1],
                                skip_footer = 148,
                                converters = {
                                    1: formatnumbers,
                                    2: formatnumbers,
                                    3: formatnumbers,
                                    4: formatnumbers,
                                    5: formatnumbers,
                                }
                               )
```

```
In [11]: tabula.index.names = ['Construction Year', 'Construction Type', 'Size']
```

```
In [12]: tabula.head()
```

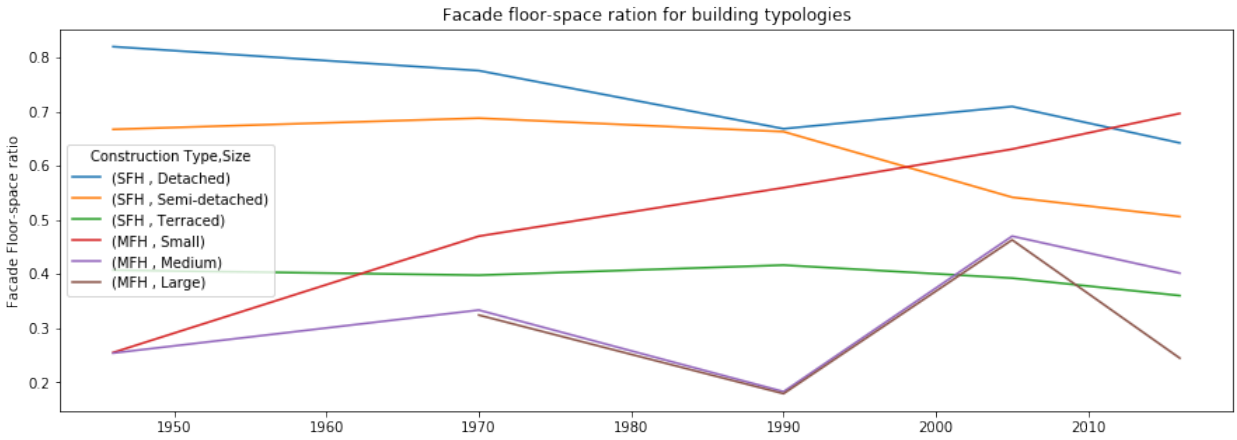
```
Out[12]: Number of housing units  \
         Construction Year Construction Type Size
         pre '46           SFH               Detached                              1
                                             Semi-detached                         1
                                             Terraced                              1
                           MFH               Small                                 6
                                             Medium                               12

                                                                 Floor surface area per housing unit (m2)
         Construction Year Construction Type Size
         pre '46           SFH               Detached                                                279.0
                                             Semi-detached                                           237.0
                                             Terraced                                                225.9
                           MFH               Small                                                    75.0
                                             Medium                                                   96.0

                                                                 Protected volume (m³)  \
```

```
                                                                  Total building envelope area (m²)  \
Construction Year Construction Type Size
pre '46           SFH               Detached                                                599.2
                                    Semi-detached                                           447.1
                                    Terraced                                                323.0
                  MFH               Small                                                   558.8
                                    Medium                                                 1112.7

                                                                  Roof (m²)  \
Construction Year Construction Type Size
pre '46           SFH               Detached                        158.4
                                    Semi-detached                   119.9
                                    Terraced                         90.0
                  MFH               Small                           165.0
                                    Medium                          215.0

                                                                  Exterior wall (m2)
Construction Year Construction Type Size
pre '46           SFH               Detached                                 228.9
                                    Semi-detached                            158.2
                                    Terraced                                  92.0
                  MFH               Small                                    114.7
                                    Medium                                   292.5
```

In [13]:
```
tabula.to_csv('data/tabula.csv')
```

In [14]:
```
facade = tabula.iloc[:, 5] / tabula.iloc[:, 0].mul(tabula.iloc[:, 1])
facade = facade.unstack(level=[1,2])
facade.index = [1970, 1990, 2005, 2016, 1946]
facade = facade.sort_index()
```

In [15]:
```
a = facade.plot(figsize=(15,5))
a.set_ylabel("Facade Floor-space ratio")
a.set_title("Facade floor-space ration for building typologies");
```



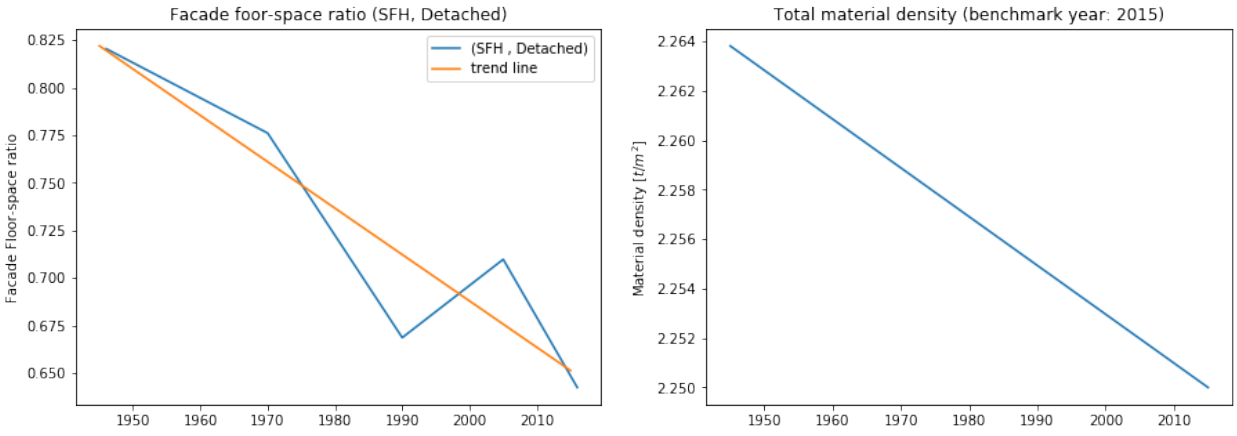
In [16]:
```
facade_plot = facade.loc[:, (['SFH '], ['Detached'])]
```

In [17]:
```
def make_trend(facade_plot, years_to_fit = [i for i in data_years.columns]):
```

```
        if isinstance(facade_plot, pd.Series):
            FFR = facade_plot
        else:
            FFR = np.asanyarray(facade_plot.iloc[:, 0])
        z = np.polyfit(y=FFR, x=facade_plot.index, deg=1)
        p = np.poly1d(z)
        return p(years_to_fit)
```

In [18]:
```
trend_p = make_trend(facade_plot)
```

In [19]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15,5))
facade_plot.plot(ax=ax1)
ax1.plot([i for i in data_years.columns], trend_p, label="trend line")
ax1.legend()
ax1.set_ylabel("Facade Floor-space ratio")
ax1.set_title('Facade foor-space ratio (SFH, Detached)')
#ax1.legend('off')
data_years.sum().plot(ax=ax2)
ax2.set_ylabel('Material density $[t/m^2]$')
ax2.set_title('Total material density (benchmark year: 2015)');
```



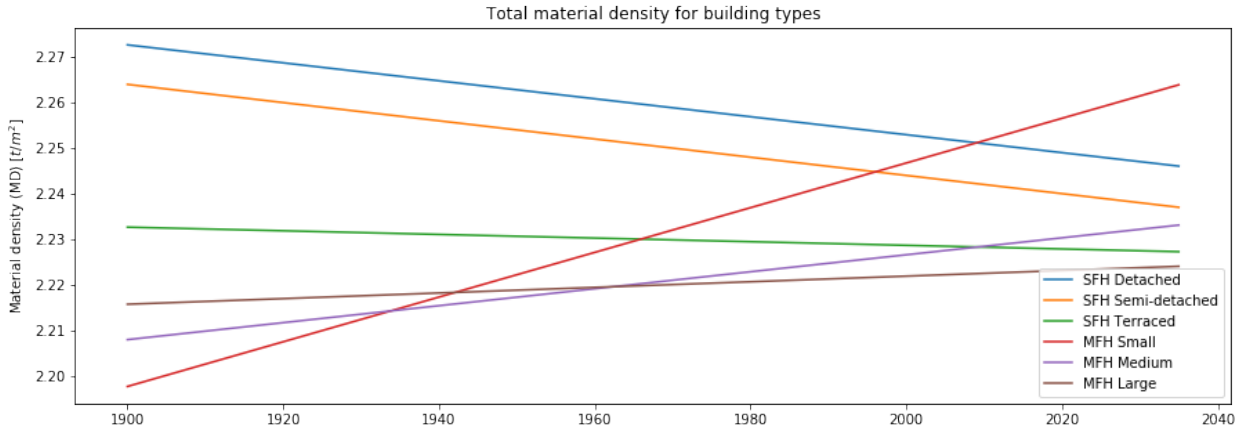
In [20]:
```
def _plot_fig(trend_md, ratio, years_to_fit = []):
    fig, ax1 = plt.subplots(1, 1, figsize=(7,5))
    if len(years_to_fit) >= 1:
        ax1.plot(years_to_fit, trend_md)
        ax1.set_xlabel('Construction year')
        ax1.set_ylabel('Material density (MD) $[t/m^2]$')
        ax1.set_title('Relationship between construction year and MD')
    else:
        ax1.plot(trend_md, ratio)
        ax1.set_xlabel('Material density (MD) $[t/m^2]$')
        ax1.set_title('Relationship between FFR and MD')
        ax1.set_ylabel('Facade Floor-space ratio (FFR)')
```

In [21]:
```
z = np.polyfit(x=make_trend(facade.iloc[:, 0]), y=data_years.sum(), deg=1)
p = np.poly1d(z)
def get_MD(facade_plot, ratio = False, plot_fig = False, years_to_fit = []):
    if len(years_to_fit) >= 1:
        ratio = make_trend(facade_plot, years_to_fit = typ_years)
    trend_md = p(ratio)
    if plot_fig:
        _plot_fig(trend_md, ratio, years_to_fit = years_to_fit)
    return trend_md
```

In [22]:
```
typ_years = [1900, 1918, 1945, 1961, 1970, 1981, 1991, 2011, 2016, 2020, 2035]
```

```
        FFR_typ = pd.DataFrame(index=typ_years)
        for e, i in enumerate(facade):
            if any(facade.iloc[:, e].isna()):
                inx_nan = [e for e,i in enumerate(facade.iloc[:, e].isna()) if i]
                if e >= 1:
                    facade.iloc[inx_nan, e] = facade.iloc[inx_nan, e-1]
                else:
                    facade.iloc[inx_nan, e] = facade.iloc[inx_nan, e+1]
            FFR = facade.iloc[:, e]
            trend_md = get_MD(FFR, years_to_fit = typ_years)
            inx = "".join(i)
            FFR_typ.loc[:, inx] = trend_md
```

In [23]:
```
p = FFR_typ.plot(figsize=(15,5))
p.set_title("Total material density for building types")
p.set_ylabel('Material density (MD) $[t/m^2]$');
#p.set_ylim((2.2726, 2.273));
```



In [24]:
```
FFR_typ = FFR_typ.T
```

In [25]:
```
FFR_typ
```

Out[25]:

	1900	1918	1945	1961	1970	1981	\
SFH Detached	2.272693	2.269141	2.263813	2.260656	2.258880	2.256709	
SFH Semi-detached	2.264013	2.260413	2.255012	2.251812	2.250012	2.247812	
SFH Terraced	2.232643	2.231925	2.230848	2.230209	2.229850	2.229411	
MFH Small	2.197642	2.206479	2.219734	2.227589	2.232007	2.237408	
MFH Medium	2.207932	2.211288	2.216320	2.219302	2.220980	2.223030	
MFH Large	2.215709	2.216820	2.218487	2.219475	2.220030	2.220710	

	1991	2011	2016	2020	2035
SFH Detached	2.254736	2.250789	2.249803	2.249013	2.246053
SFH Semi-detached	2.245812	2.241811	2.240811	2.240011	2.237011
SFH Terraced	2.229012	2.228214	2.228014	2.227854	2.227256
MFH Small	2.242317	2.252136	2.254590	2.256554	2.263918
MFH Medium	2.224894	2.228622	2.229554	2.230300	2.233095
MFH Large	2.221327	2.222562	2.222870	2.223117	2.224043

In [26]:
```
data_years_nn = data_years.dropna()
```

In [27]:
```
def _get_p(data, i):
    z = np.polyfit(x=data.columns, y=data.iloc[i], deg=1)
    p = np.poly1d(z)
    return p
```

In [28]:
```
all_shares = pd.DataFrame(index=typ_years)
```

```
        for i in range(data_years_nn.shape[0]):
            p =  _get_p(data_years_nn, i)
            shares = [p(y) for y in typ_years]
            all_shares.loc[:, data_years_nn.index[i]] = shares
        all_shares = all_shares.div(all_shares.sum(axis=1), axis=0)
```

```
In [29]: densities = pd.DataFrame()
         for i in range(FFR_typ.shape[0]):
             temp = all_shares.mul(FFR_typ.iloc[i], axis=0)
             #temp.index = ["a", temp.index]
             tuples = [(FFR_typ.index[i], j) for j in temp.index]
             index = pd.MultiIndex.from_tuples(tuples, names=['Building type', 'Construction year'])
             temp.index = index
             densities = densities.append(temp)
```

```
In [30]: densities.head(15)
```

Out[30]:

Metal	Minerals	Plastic	Wood
Building type	Construction year		
SFH Detached	1900	0.029080	2.203000	0.019080	0.021533
	1918	0.029224	2.199400	0.019224	0.021293
	1945	0.029440	2.194000	0.019440	0.020933
	1961	0.029568	2.190800	0.019568	0.020720
	1970	0.029640	2.189000	0.019640	0.020600
	1981	0.029728	2.186800	0.019728	0.020453
	1991	0.029808	2.184800	0.019808	0.020320
	2011	0.029968	2.180800	0.019968	0.020053
	2016	0.030008	2.179800	0.020008	0.019987
	2020	0.030040	2.179000	0.020040	0.019933
	2035	0.030160	2.176000	0.020160	0.019733
SFH Semi-detached	1900	0.028969	2.194586	0.019007	0.021451
	1918	0.029112	2.190940	0.019150	0.021211
	1945	0.029326	2.185470	0.019364	0.020852
	1961	0.029452	2.182229	0.019491	0.020639

```
In [31]: densities.to_csv('data/densities.csv')
```

```
In [32]: con = ["{}".format(i).strip() for i in tabula.index.droplevel([0,1])]
         typ = ["{}".format(j).strip() for j in tabula.index.droplevel([0,2])]
```

```
In [33]: tabula.loc[:, 'con'] = con
         tabula.loc[:, 'typ'] = typ
```

```
/usr/lib/python3.6/site-packages/ipykernel/ipkernel.py:208: PerformanceWarning: indexing past lexsort
  res = shell.run_cell(code, store_history=store_history, silent=silent)
```

```
In [34]: tabula_reg = tabula.loc[:, ['Floor surface area per housing unit (m2)', 'con', 'typ']]
         tabula_reg.columns = ['area', 'con', 'typ']
```

```
In [35]: tabula.typ == "SFH"
```

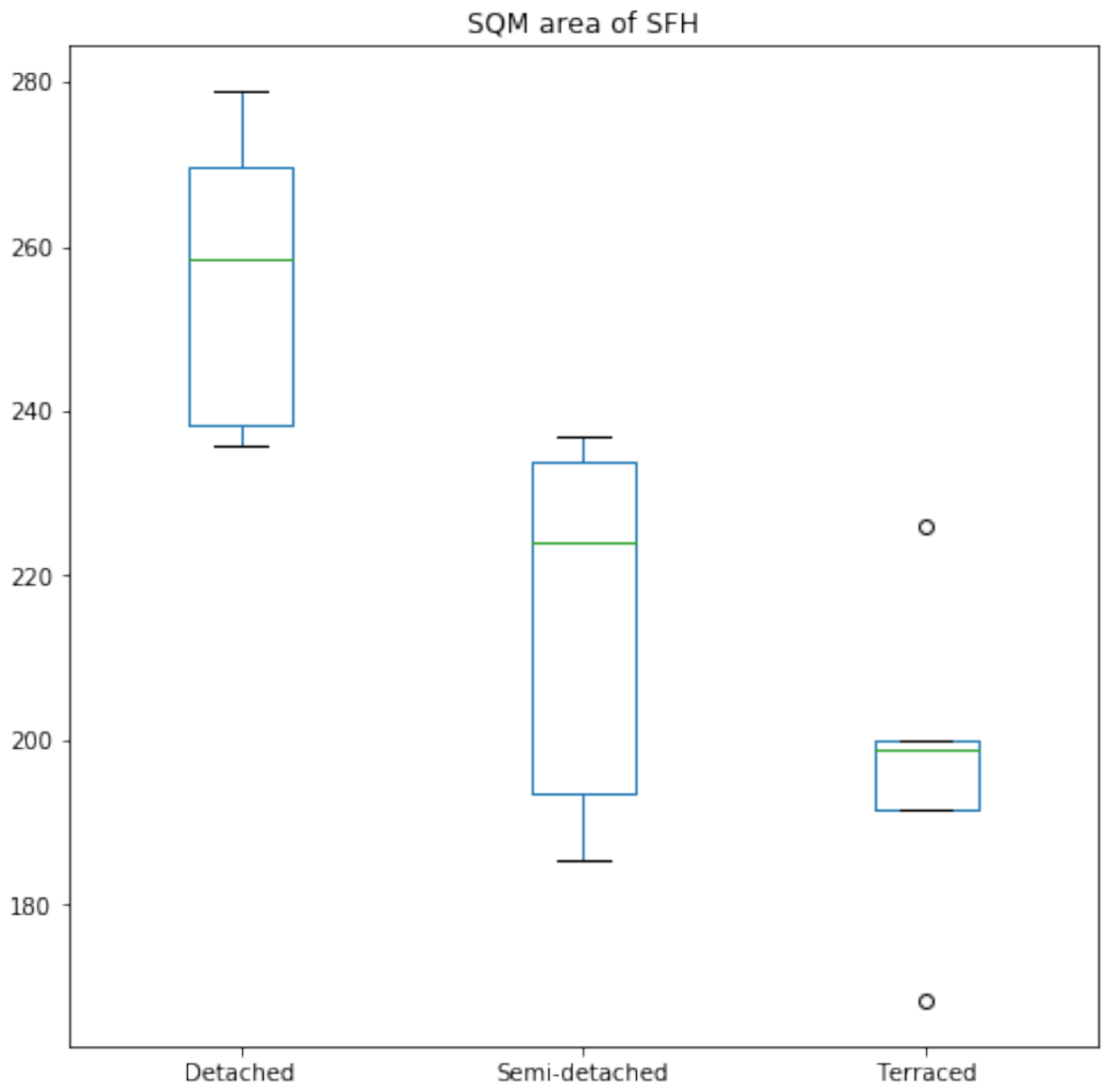
Out[35]:

Construction Year	Construction Type	Size	
pre '46	SFH	Detached	True
		Semi-detached	True
		Terraced	True
	MFH	Small	False
		Medium	False
'46-'70	SFH	Detached	True
		Semi-detached	True
		Terraced	True
	MFH	Small	False
		Medium	False
		Large	False

```
        '71-'90            SFH               Detached          True
                                             Semi-detached     True
                                             Terraced          True
                           MFH               Small            False
                                             Medium           False
                                             Large            False
        '91-'05            SFH               Detached          True
                                             Semi-detached     True
                                             Terraced          True
                           MFH               Small            False
                                             Medium           False
                                             Large            False
        post '05           SFH               Detached          True
                                             Semi-detached     True
                                             Terraced          True
                           MFH               Small            False
                                             Medium           False
                                             Large            False
        Name: typ, dtype: bool
```

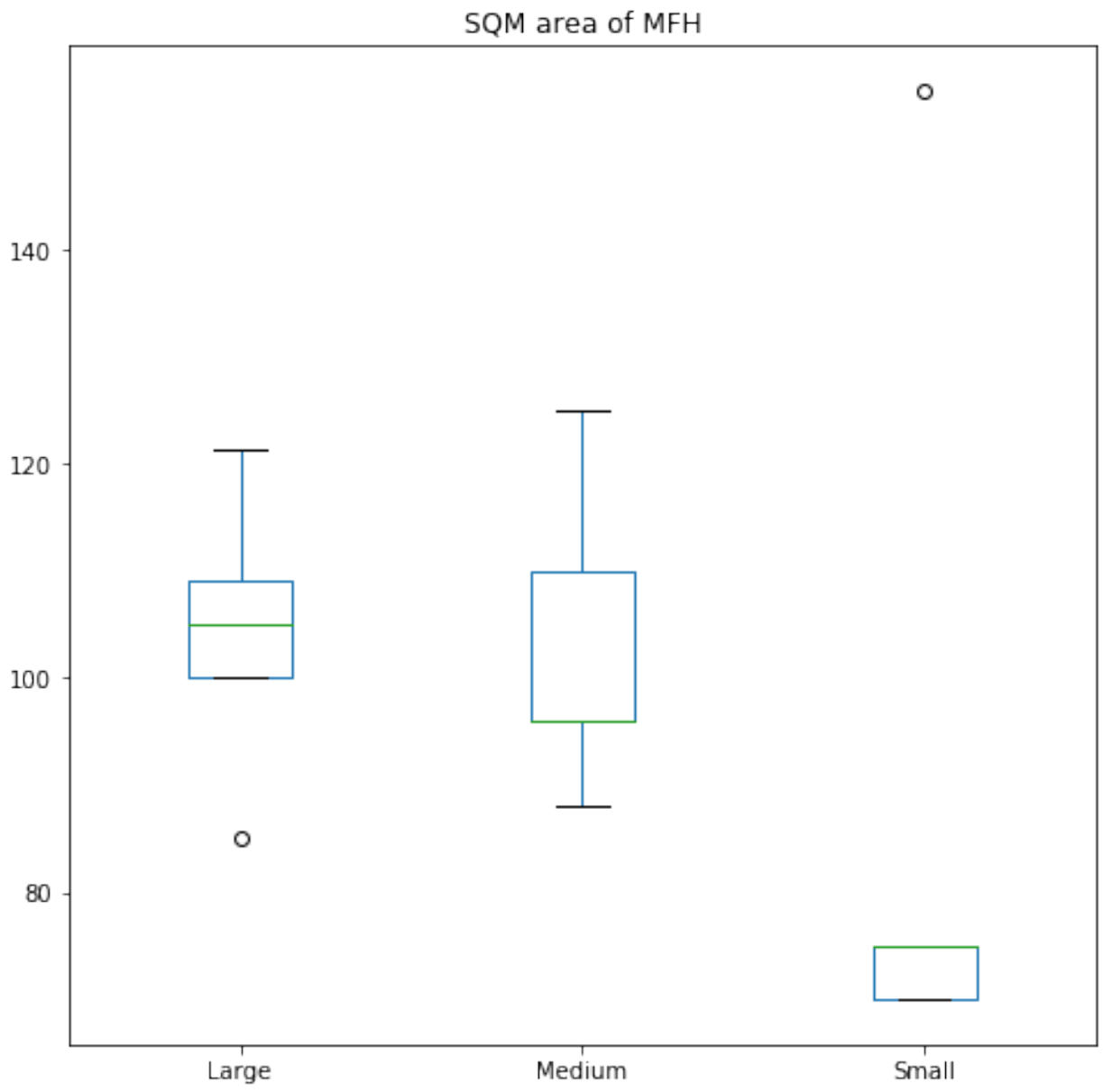
```
In [36]: p = tabula_reg.loc[tabula.typ == 'SFH', 'area'].unstack(2).plot.box(figsize=(8,8))
         p.set_title("SQM area of SFH");
```

```
/usr/lib/python3.6/site-packages/ipykernel_launcher.py:1: PerformanceWarning: indexing past lexsort d
  """Entry point for launching an IPython kernel.
```




```
In [37]: p = tabula_reg.loc[tabula.typ == 'MFH', 'area'].unstack(2).plot.box(figsize=(8,8))
         p.set_title("SQM area of MFH");
```

```
/usr/lib/python3.6/site-packages/ipykernel_launcher.py:1: PerformanceWarning: indexing past lexsort d
  """Entry point for launching an IPython kernel.
```




```
In [38]: tabula_reg.loc[tabula.typ == 'MFH', 'area'].unstack(2)
```

```
/usr/lib/python3.6/site-packages/ipykernel_launcher.py:1: PerformanceWarning: indexing past lexsort d
  """Entry point for launching an IPython kernel.
```

Out[38]:

Size		Large	Medium	Small
Construction Year	Construction Type			
'46-'70	MFH	105.0	88.0	70.0
'71-'90	MFH	85.0	125.0	70.0
'91-'05	MFH	105.0	110.0	155.0
post '05	MFH	121.3	96.0	75.0
pre '46	MFH	NaN	96.0	75.0

```
In [46]: def get_year_lable(year, year_ranges, tabula):
             inx_year = [e for e, i in enumerate(year_ranges) if year >= i[0] and year <= i[1]][0]
             year_lable = tabula.index.get_level_values(0).unique()[inx_year]
             return year_lable
```

```
        def clean_year(y):
            years = []
            flip = False
            highest = 0
            for i in y.split("'"):
                if 'post' in i: flip = True
                if len(i) >= 1:
                    i = i.replace('-', '')
                    i = i.replace('pre', '-inf')
                    i = i.replace('post', 'inf')
                    i = float(i)
                    if i >= 40:
                        i += 1900
                    else:
                        i += 2000
                    if i <= highest:
                        i += 1
                    highest = i
                    years.append(i)
            if flip:
                years = years[::-1]
            return years

        def get_typ(year, construction, sqm, tabula = tabula):
            construction += ' '
            con_year = get_year_lable(2006, year_ranges, tabula)
            temp = tabula.loc[(con_year, construction, slice(None)), "Floor surface area per housing
            p = random.random()
            temp_p = 1 - (abs(temp - sqm) / sum(abs(temp - sqm)))
            temp_sel = temp_p.loc[temp_p >= p]
            if temp_sel.shape[0] > 1:
                temp_sel = temp_sel.loc[temp_sel == temp_sel.min()]
            elif temp_sel.shape[0] == 0:
                temp_sel = temp_p.loc[temp_p == temp_p.max()]
            if temp_sel.shape[0] == 0:
                print('error')
            btyp = "{} {}".format(
                temp_sel.index.get_level_values(1)[0].strip(),
                temp_sel.index.get_level_values(2)[0].strip())
            return(btyp)

        def get_den(year, p, densities):
            cyear_inx = densities.index.get_level_values(1).unique()
            year_diff = [abs(int(i) - int(year)) for i in cyear_inx]
            sel_inx = [e for e, i in enumerate(year_diff) if i == min(year_diff)][0]
            den = densities.loc[(p, cyear_inx[sel_inx])]
            return den
```

In [47]:
```
        import random
        year_ranges = [clean_year(i) for i in tabula.index.get_level_values(0).unique()]
```

In [48]:
```
        sqm = 210
        year = 1946
        construction = "SFH"
        random.seed(1234)
        print("Tab. Material intensities of selected building typologies{sep}\
        for construction year:\t{}{sep}\
        and construction type:\t{}{sep}\
        with a flor area of:\t{} m^2".format(year, construction, sqm, sep='\n'))
```

```
        print("-" * 60)
        print("| {:<20} | {:^6} | {:^6} | {:^6} | {:^6} |".format('Building Type', "Metals", "Mine."
        print("=" * 60)
        for i in range(20):
            p = get_typ(year, construction, sqm, tabula = tabula)
            d = get_den(year, p, densities)
            print("| {:<20} | {} |".format(p, " | ".join(["{:0.4f}".format(i) for i in d])))
        print("-" * 60)
```

```
Tab. Material intensities of selected building typologies
for construction year:  1946
and construction type:  SFH
with a flor area of:    210 m^2
------------------------------------------------------------
| Building Type        | Metals | Mine.  | Plas.  |  Wood  |
============================================================
| SFH Terraced         | 0.0290 | 2.1621 | 0.0192 | 0.0206 |
| SFH Semi-detached    | 0.0293 | 2.1855 | 0.0194 | 0.0209 |
| SFH Detached         | 0.0294 | 2.1940 | 0.0194 | 0.0209 |
| SFH Terraced         | 0.0290 | 2.1621 | 0.0192 | 0.0206 |
| SFH Terraced         | 0.0290 | 2.1621 | 0.0192 | 0.0206 |
| SFH Semi-detached    | 0.0293 | 2.1855 | 0.0194 | 0.0209 |
| SFH Semi-detached    | 0.0293 | 2.1855 | 0.0194 | 0.0209 |
| SFH Detached         | 0.0294 | 2.1940 | 0.0194 | 0.0209 |
| SFH Terraced         | 0.0290 | 2.1621 | 0.0192 | 0.0206 |
| SFH Detached         | 0.0294 | 2.1940 | 0.0194 | 0.0209 |
| SFH Detached         | 0.0294 | 2.1940 | 0.0194 | 0.0209 |
| SFH Terraced         | 0.0290 | 2.1621 | 0.0192 | 0.0206 |
| SFH Detached         | 0.0294 | 2.1940 | 0.0194 | 0.0209 |
| SFH Semi-detached    | 0.0293 | 2.1855 | 0.0194 | 0.0209 |
| SFH Semi-detached    | 0.0293 | 2.1855 | 0.0194 | 0.0209 |
| SFH Detached         | 0.0294 | 2.1940 | 0.0194 | 0.0209 |
| SFH Detached         | 0.0294 | 2.1940 | 0.0194 | 0.0209 |
| SFH Detached         | 0.0294 | 2.1940 | 0.0194 | 0.0209 |
| SFH Detached         | 0.0294 | 2.1940 | 0.0194 | 0.0209 |
| SFH Semi-detached    | 0.0293 | 2.1855 | 0.0194 | 0.0209 |
------------------------------------------------------------
```

Brussels Mat. Step 2.a Construction Aggregates

```
In [1]: import pandas as pd
        import numpy as np
        from matplotlib import pyplot as plt
```

```
In [62]: constructions = pd.read_excel('IBSA/11.2_amenagement_territoire_parc_batiments_20170626.xls
                               sheetname='11.2.5.1', skiprows = 1, #header = 0,
                               index_col = 0, skip_footer = 10)
         constructions.loc[:, 'region'] = constructions.index
         constructions = constructions.set_index('Année')
```

```
In [63]: constructions.head()
```

```
Out[63]: Bâtiments résidentiels           Unnamed: 3             Unnamed: 4  \
         Année
         NaN      Nouvelles constructions                    NaN                      NaN
         NaN          Nombre de bâtiments  Nombre de logements  Nombre d'appartements
          1996.0                      234                 1375                   1231
          1996.0                     3875                 4693                   1003
          1996.0                     1324                 2546                   1319
```

```
                                                Unnamed: 5                  Unnamed: 6  \
Année
NaN                                                    NaN                         NaN
NaN      Nombre de bâtiments avec un seul logement  Superficie habitable (m²)
 1996.0                                                144                      167147
 1996.0                                               3690                      723710
 1996.0                                               1227                      308394

                    Unnamed: 7           Unnamed: 8 Bâtiments non résidentiels  \
Année
NaN                Rénovation                    NaN    Nouvelles constructions
NaN      Nombre de bâtiments  Nombre de logements        Nombre de bâtiments
 1996.0                  960                 2264                         77
 1996.0                 2102                 2389                        489
 1996.0                  949                 1460                        177

                   Unnamed: 10  Unnamed: 11          Unnamed: 12  \
Année
NaN                        NaN          NaN           Rénovation
NaN      Nombre de logements  Volume (m³)  Nombre de bâtiments
 1996.0                   36      1670453                  167
 1996.0                   64      2378497                  367
 1996.0                   24       724580                  143

                   Unnamed: 13                         region
Année
NaN                        NaN                            NaN
NaN      Nombre de logements                       Colonne3
 1996.0              117.938  Région de Bruxelles-Capitale
 1996.0               43.015               Brabant flamand
 1996.0                 27.3                Brabant wallon
```

```
In [56]: def get_new_cols(iterable, keyword):
             if not isinstance(keyword, str):
                 keyword = 'NaN'
             new_cols_0 = list()
             old_cols = str()
             for i in iterable:
                 if not isinstance(i, str):
                     i = 'NaN'
                 if keyword in i or keyword == i:
                     new_cols_0.append(old_cols)
                 else:
                     new_cols_0.append(i)
                     old_cols = i
             return(new_cols_0)
```

```
In [57]: new_cols_0 = get_new_cols(constructions.columns, 'Unnamed')
         new_cols_1 = get_new_cols(constructions.iloc[0,:], np.nan)
         new_cols_2 = get_new_cols(constructions.iloc[1,:], np.nan)
```

```
In [59]: constructions.columns = [new_cols_0, new_cols_1, new_cols_2]
         constructions = constructions.iloc[2:, :]
         inx = constructions.region == 'Région de Bruxelles-Capitale'
         constructions = constructions.loc[inx.iloc[:,0]]
         constructions = constructions.iloc[:,:-1]
```

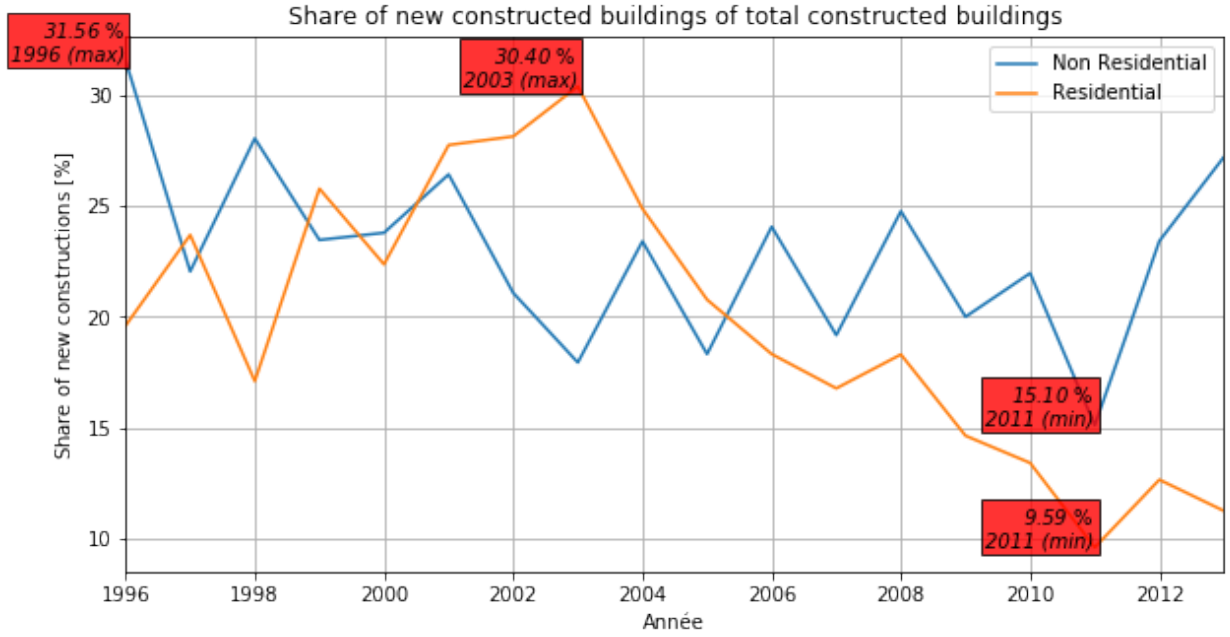
```
In [6]: constructions = constructions.T.sort_index()
```

```
In [7]: sqm = constructions.loc[(
            slice(None),
```

```
            #'Bâtiments résidentiels',
            slice(None),
            #slice('Nouvelles constructions'),
            [
                #'Superficie habitable (m²)',
                #'Nombre de logements',
                'Nombre de bâtiments'
            ],
        ), :]
```

In [8]:
```
sqm.index = sqm.index.droplevel([-1])
sqm_per = pd.concat([
    pd.DataFrame(sqm.iloc[0].div(sqm.iloc[1] + sqm.iloc[0]).mul(100), columns=['Non Residenti
    pd.DataFrame(sqm.iloc[2].div(sqm.iloc[3] + sqm.iloc[2]).mul(100), columns=['Residential']
    axis=1)
```

In [9]:
```
g = sqm_per.plot(figsize=(10, 5))
g.set_ylabel(r'Share of new constructions $[\%]$');
g.set_title("Share of new constructed buildings of total constructed buildings");
for i in [0, 1]:
    year_max = sqm_per.index[sqm_per.iloc[:, i] == sqm_per.iloc[:, i].max()][0]
    year_min = sqm_per.index[sqm_per.iloc[:, i] == sqm_per.iloc[:, i].min()][0]
    g.text(year_max, sqm_per.iloc[:, i].max(),
           "{1:0.2f} $\%$\n{0:0.0f} (max)".format(year_max, sqm_per.iloc[:, i].max()),
           style='italic', horizontalalignment='right',
           bbox={'facecolor':'red', 'alpha':0.8, 'pad':2})
    g.text(year_min, sqm_per.iloc[:, i].min(),
           "{1:0.2f} $\%$\n{0:0.0f} (min)".format(year_min, sqm_per.iloc[:, i].min()),
           style='italic', horizontalalignment='right',
           bbox={'facecolor':'red', 'alpha':0.8, 'pad':2})
g.grid()
plt.savefig('FIGURES/btyp_share_new.png', dpi=300)
```

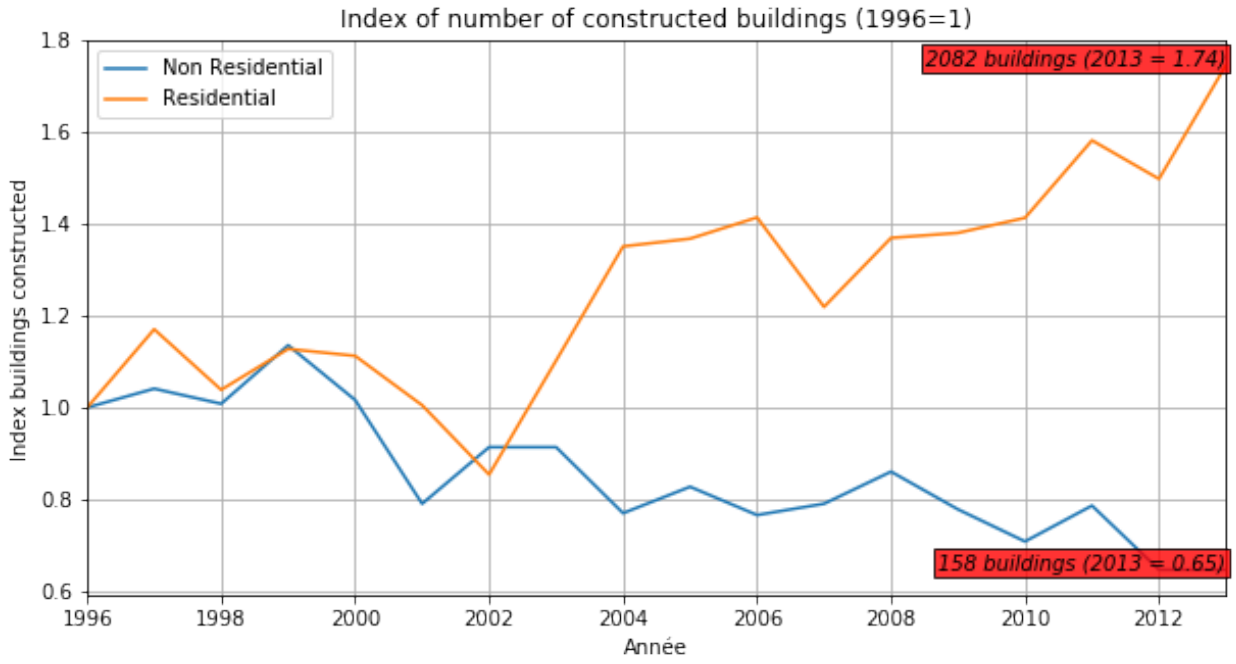


In [10]:
```
sqm_tot = pd.concat([
    pd.DataFrame(sqm.iloc[1] + sqm.iloc[0], columns=['Non Residential']),
    pd.DataFrame(sqm.iloc[3] + sqm.iloc[2], columns=['Residential'])],
    axis=1)
```

```
In [11]: sqm_tot.div(sqm_tot.iloc[0]).iloc[-1]
```

```
Out[11]: Non Residential    0.647541
         Residential         1.74372
         Name: 2013.0, dtype: object
```

```
In [12]: g = sqm_tot.div(sqm_tot.iloc[0]).plot(figsize=(10, 5))
         g.set_ylabel(r'Index buildings constructed');
         g.set_title("Index of number of constructed buildings (1996=1)");
         for i in [0, 1]:
             g.text(2013, sqm_tot.div(sqm_tot.iloc[0]).iloc[-1, i],
                    "{0:0.0f} buildings (2013 = {1:0.2f})".format(sqm_tot.iloc[-1, i],
                                                                  sqm_tot.div(sqm_tot.iloc[0]).iloc[-1, i]),
                    style='italic', horizontalalignment='right',
                    bbox={'facecolor':'red', 'alpha':0.8, 'pad':2})
         g.grid()
         plt.savefig('FIGURES/btyp_total_buildings.png', dpi=300)
```

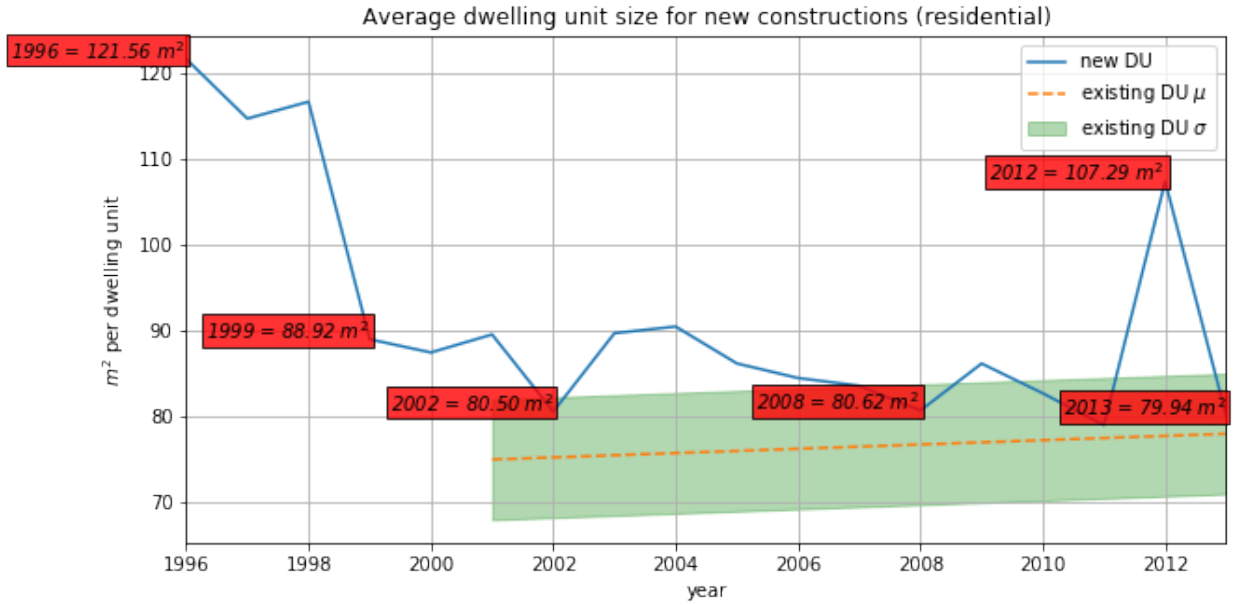



```
In [13]: sqm_du = constructions.loc[(
             'Bâtiments résidentiels',
             slice('Nouvelles constructions'),
             [
                 'Superficie habitable (m$^2$)',
                 'Nombre de logements',
             ],
         ), :]
         sqm_du.index = sqm_du.index.droplevel([0,1])
```

```
In [14]: sqm_du = sqm_du.iloc[1].div(sqm_du.iloc[0])
         #sqm_du.name = 'sqm per DU'
```

```
In [49]: MU_du = 74.92
         MU_sd = 7.01
         x = [i for i in range(2001, 2014)]
         y = [MU_du + i/4 for i in range(13)]
         y1 = [i+MU_sd for i in y]
         y2 = [i-MU_sd for i in y]
```

```
In [53]: g = sqm_du.plot(figsize=(10, 5), label='new DU')
         g.set_ylabel('$m^2$ per dwelling unit')
         g.set_xlabel('year')
         g.set_title("Average dwelling unit size for new constructions (residential)")
         for e, i in enumerate(sqm_du.index):
             if e in [0, 3, 6, 12, 16, 17]:
                 g.text(i, sqm_du.loc[i],
                        "{0:0.0f} = {1:0.2f} $m^2$".format(i, sqm_du.loc[i]),
                        style='italic', horizontalalignment='right',
                        bbox={'facecolor':'red', 'alpha':0.8, 'pad':2})
         g.grid()
         g.plot(x, y, "--", label='existing DU $\mu$')
         g.fill_between(x, y1=y1, y2=y2,
                        label='existing DU $\sigma$', color='green', alpha=0.3)
         g.legend()
         plt.savefig('FIGURES/btyp_du_size.png', dpi=300)
```



Brussels Mat. Step 3.a Residential Floor Space

```
In [1]: import pandas as pd
        import numpy as np
        from matplotlib import pyplot as plt
```

```
In [2]: bua = pd.read_excel("./IBSA/11.1_amenagement_territoire_occupation_sol_20151125.xls",
                           sheetname='11.1.1.4', skiprows = 1, #header = [0,1],
                           index_col = 0, skip_footer = 10)
```

```
In [3]: def get_new_cols(iterable, keyword):
            if not isinstance(keyword, str):
                keyword = 'NaN'
            new_cols_0 = list()
            old_cols = str()
            for i in iterable:
                if isinstance(i, str) or isinstance(i, int):
                    pass
                else:
                    i = 'NaN'
```

```
            if keyword == i or (isinstance(i, str) and keyword in i):
                new_cols_0.append(str(old_cols))
            else:
                new_cols_0.append(str(i))
                old_cols = str(i)
        return(new_cols_0)
```

```
In [4]: new_cols_0 = get_new_cols(bua.columns, 'Unnamed')
        new_cols_1 = get_new_cols(bua.iloc[0,:], np.nan)
        new_cols_2 = get_new_cols(bua.iloc[1,:], np.nan)
```

```
In [5]: bua.columns = [new_cols_0, new_cols_1, new_cols_2]
        bua = bua.iloc[2:, 1:]
        bua = bua.T.sort_index()
```

```
In [6]: #bua#.columns#.sort_values()
```

```
In [7]: bau_build = bua.loc[(
            slice(None),
            slice(None),
            ['Bâti']), :]
```

```
In [8]: bau_build.index = bau_build.index.droplevel([1,2])
        bau_build = bau_build.T
        bau_build.index = [i.replace('St', 'Saint') for i in bau_build.index]
        bau_build = bau_build * 10000
```

```
In [9]: hh = pd.read_excel("./IBSA/1.4_population_menages_20170829.xlsx",
                           sheetname='1.4.2.1', skiprows = 1, #header = [0,1],
                           index_col = 0, skip_footer = 10)
        hh.loc[:, '1 personne'] = hh.iloc[:, 2]
        hh = hh.iloc[1:, 2:-1]
        hh.columns = ['hh_{}'.format(i) for i in range(1,9)]
        hh = hh.mul([i for i in range(1,9)])
```

```
In [10]: sqm = pd.read_excel('IBSA/log08_0_2001.xls',
                            header = 2,
                            index_col = 1,
                            skip_footer = 6
                           )
```

```
In [11]: sqm = sqm.iloc[2:, [1]]
         sqm.columns = ['sqm']
         sqm = sqm.mul(hh.sum(axis=1), axis=0)
```

```
In [12]: sqm.loc[:, 'bau'] = bau_build.loc[:, '2005']
```

```
In [13]: sqm
```

```
Out[13]: sqm          bau
         Territoire
         Anderlecht             7.97353e+06  7.56164e+06
         Auderghem              2.69623e+06  2.73296e+06
         Berchem-Sainte-Agathe    1.879e+06  1.53595e+06
         Bruxelles              1.22655e+07  1.38573e+07
         Etterbeek              3.40235e+06   2.1059e+06
         Evere                  2.88231e+06  2.22417e+06
         Forest                 4.11621e+06  3.26044e+06
         Ganshoren              1.79878e+06  1.10303e+06
         Ixelles                6.27498e+06  3.91626e+06
         Jette                  3.73996e+06  2.60211e+06
         Koekelberg             1.54673e+06       655600
         Molenbeek-Saint-Jean   6.65206e+06  3.07914e+06
         Saint-Gilles           3.29663e+06   1.6263e+06
```

```
Saint-Josse-ten-Noode  1.71305e+06       703388
Schaerbeek             9.55993e+06  4.41833e+06
Uccle                  7.04683e+06  1.01344e+07
Watermael-Boitsfort    1.99856e+06  2.60981e+06
Woluwe-Saint-Lambert       4.41e+06  3.86748e+06
Woluwe-Saint-Pierre    3.73977e+06  4.76298e+06
```

```
In [17]: sqm.sqm.sum()
```

```
Out[17]: 86992460.27692918
```

```
In [14]: sqm.sqm.div(sqm.bau)
```

```
Out[14]: Territoire
         Anderlecht               1.05447
         Auderghem               0.986561
         Berchem-Sainte-Agathe    1.22335
         Bruxelles               0.885134
         Etterbeek                1.61563
         Evere                    1.29591
         Forest                   1.26247
         Ganshoren                1.63076
         Ixelles                  1.60229
         Jette                    1.43728
         Koekelberg               2.35926
         Molenbeek-Saint-Jean     2.16036
         Saint-Gilles             2.02707
         Saint-Josse-ten-Noode    2.43542
         Schaerbeek                2.1637
         Uccle                   0.695339
         Watermael-Boitsfort     0.765788
         Woluwe-Saint-Lambert     1.14028
         Woluwe-Saint-Pierre     0.785174
         dtype: object
```

CHAPTER 5

API: Top-Down (Urban Metabolism)

5.1 City

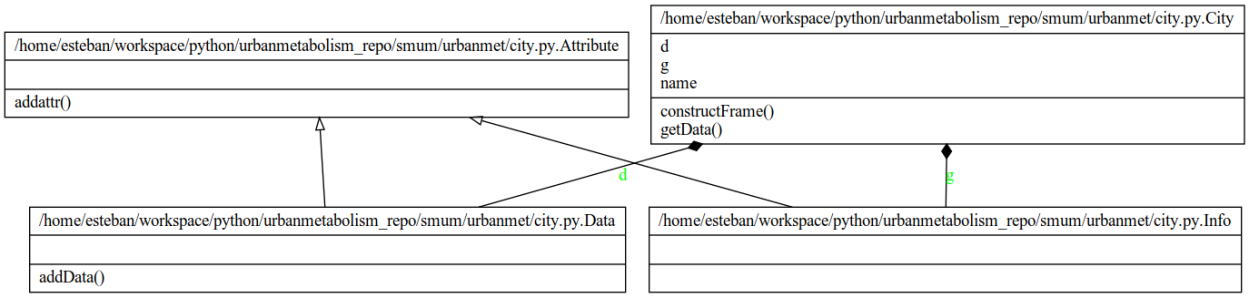


Fig. 5.1: City class diagram.

class urbanmet.city.**City**(*name*)

City class definition.

Reads and process the input data required for a city level analysis.

name

str – city name.

dat

City.Data – Data of city-object.

g_info

City.Info – Info of city-object.

class Data

Construct city data frames and methods of computation.

add_data(*sheet*, *sheet_data*)

Add a data frame to *Data* class.

Adds data:

- Affluence
- Technology

Parameters
- **sheet_data** (*pandas.DataFrame*) – Excel file data as pandas DataFrame.
- **sheet** (*str*) – Excel file sheet name.

class Info(*data*)
Construct city initial constants.

Parameters **data** (*pandas.DataFrame*) – City constant info data.

__init__(*data*)
Initialize self. See help(type(self)) for accurate signature.

__init__(*name*)
Initialize self. See help(type(self)) for accurate signature.

__weakref__
list of weak references to the object (if defined)

construct_frame(*xls*)
Construct pandas DataFrames from an excel file.

Reads all sheets from the *xls* excel file. Sheet names *City* calls function *Info*, all the other sheet call function *Data.add_data*

Parameters **xls** (*pandas.io.excel.ExcelFile*) – Excel file.

get_data(*excel_file='./data/InputTables.xlsx'*)
Get excel data.

Parameters **excel_file** (*str*) – Excel file containing input-tables.

5.2 Materials

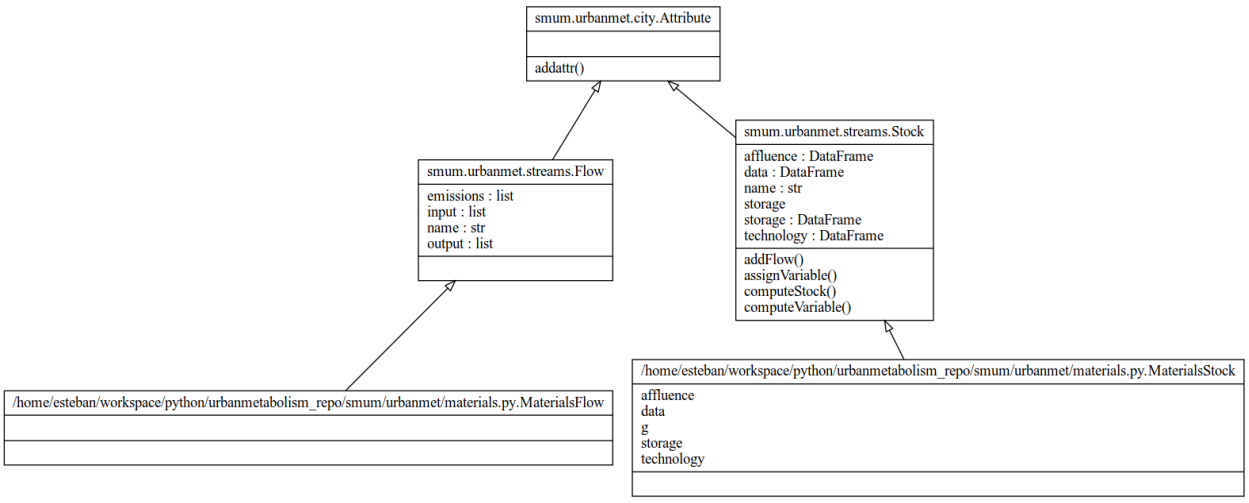


Fig. 5.2: Materials class diagram.

class urbanmet.materials.**MaterialsFlow**(*city*)
Sample Flow class.

__init__(*city*)
MaterialsFlow class initiator.

Parameters **city** (*urbanmetabolism.city.City*) – city object.

class urbanmet.materials.**MaterialsStock**(*city*)
Material Stock.

This class defines the existing material stock by sector.

affluence
pandas.DataFrame – city affluence data

technology
pandas.DataFrame – city technology data

data
pandas.DataFrame – city materiasl data

g
pandas.DataFrame – city constant values

storage

__init__(*city*)
MaterialsStock class initiator.

Parameters **city** (*urbanmetabolism.city.City*) – city object.

5.3 Water

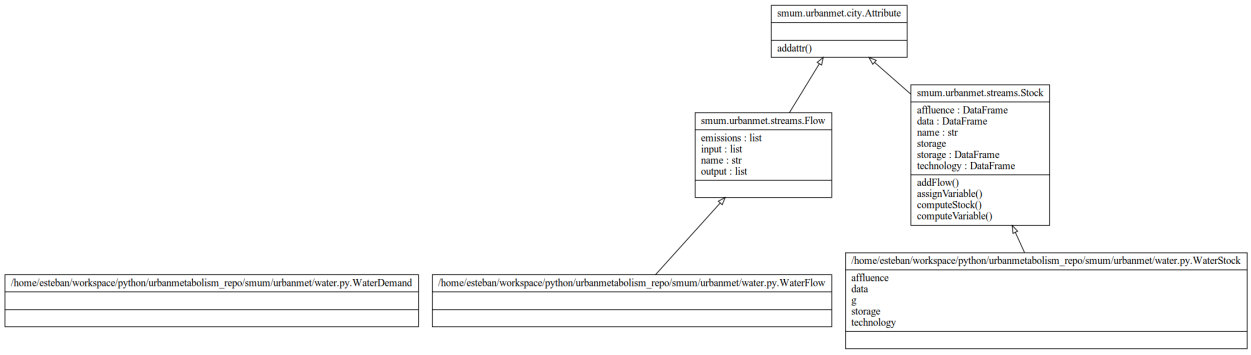


Fig. 5.3: Water class diagram.

class urbanmet.water.**WaterDemand**
Water Demand Model

This class defines the household water demand model.

__weakref__
list of weak references to the object (if defined)

class urbanmet.water.**WaterFlow**(*flow_name*, *internal=False*)
Urban Water Flow

This class defines the *WaterFlow* of a city.

This water flow is balanced as follows:

class urbanmet.water.**WaterStock**(*city*)
Urban Water Stock

This class defines the *WaterFlow* of a city.

__init__(*city*)
Class initiator.

5.4 Energy

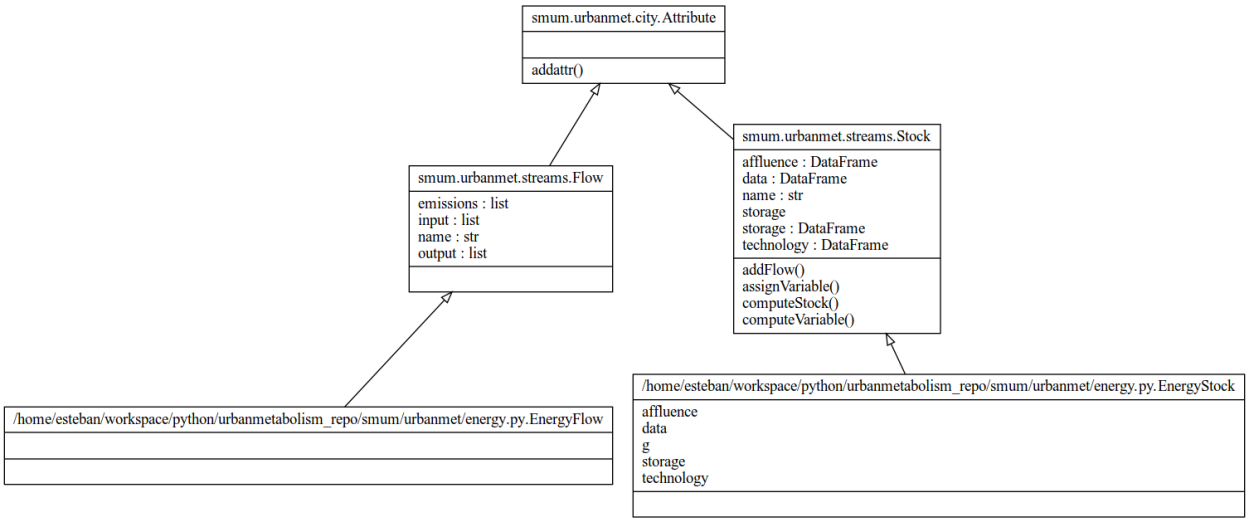


Fig. 5.4: Energy class diagram.

class urbanmet.energy.**EnergyFlow**(*city*)
Energy Flow class.

__init__(*city*)
Class initiator for urban flows.

transport_I(*mode*, *Pp*, *P*, *rho*, *h*, *epsilon*)

class urbanmet.energy.**EnergyStock**(*city*)
Energy Stock.

__init__(*city*)
Energy Stock class initiator.

- Require input: *city*

5.5 Food

class urbanmet.food.**FoodFlow**(*city*)
Food Flow class.

__init__(*city*)
Class initiator for urban flows.

class urbanmet.food.**FoodStock**(*city*)
Food Stock.

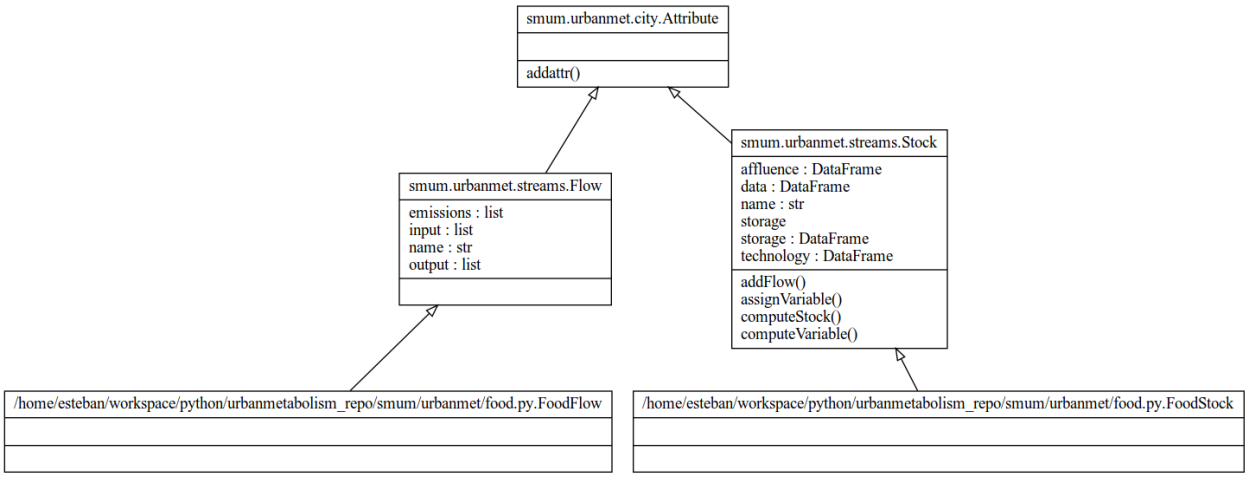


Fig. 5.5: Food class diagram.

__init__(*city*)
Food Stock class initiator.

- Require input: *city*

5.6 Waste

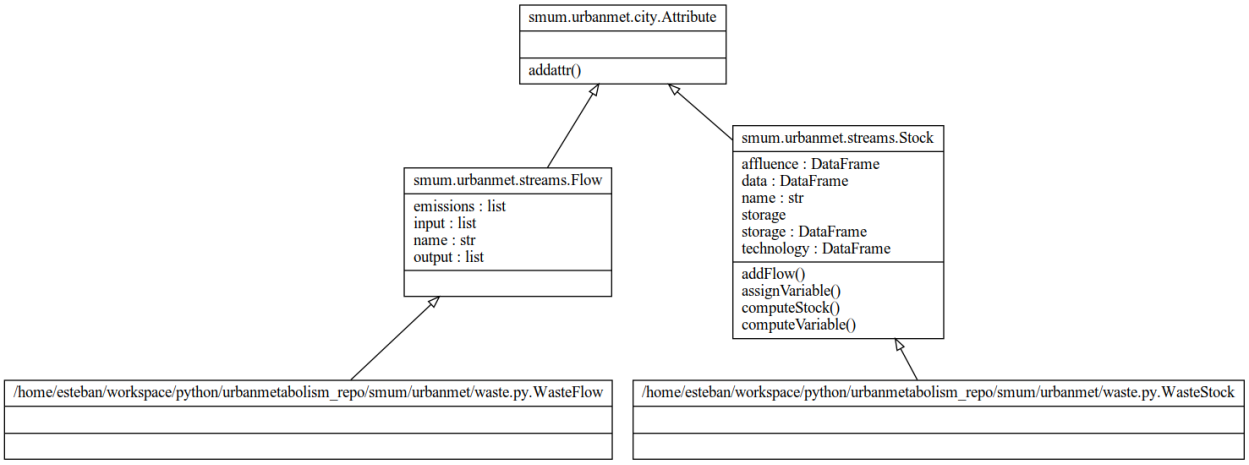


Fig. 5.6: Waste class diagram.

class urbanmet.waste.**WasteFlow**(*city*)
Waste Flow class.

__init__(*city*)
Class initiator for urban flows.

class urbanmet.waste.**WasteStock**(*city*)
Waste Stock.

__init__(*city*)
Waste Stock class initiator.

- Require input: *city*

CHAPTER 6

API: Bottom-Up (Spatial Microsimulation)

microsim.run.**run_calibrated_model**(*model_in*, *log_level=50*, *err='wf'*, *project='reweight'*, *resample_years=[]*, *rep={}*, ***kwargs*)

Run and calibrate model with all required iterations.

Parameters

- **model_in** (*dict*) – Model defines as a dictionary, with specified variables as keys, containing the *table_model* for each variable and an optional formula.
- **err** (str) – Weight to use for error optimization. Defaults to *'wf'*.
- **log_level** (int, optional) – Logging level passed to pymc3 log-level.
- **project** (str) – Method used for the projection of the sample survey. Defaults to *'reweight'*, this method will reweight the synthetic sample survey to match aggregates from the census file. This method is fast but might contain large errors on the resulting marginal sums (i.e. TAE). An alternative method is define as *'resample'*. This method will construct a new sample for each iteration and reweight it to the know aggregates on the census file, this method is more time consuming as the samples are created on each iteration via MCMC. If the variable is set to *False* the method will create a sample for a single year.
- **rep** (dict) – Dictionary containing rules for replacing names on sample survey. Defaults to *dict()* i.e no modifications, empty dictionary.
- ****kwargs** – Keyword arguments passed to 'run_composite_model'.

Returns calibrated reweighted survey.

Return type reweighted_survey (pandas.DataFrame)

Example

```
>>> elec = pd.read_csv('data/test_elec.csv', index_col=0)
>>> inc =  pd.read_csv('data/test_inc.csv',  index_col=0)
>>> model = {"Income":      {'table_model': inc },
```

(continues on next page)

(continued from previous page)

```
            "Electricity": {'table_model': elec}
>>> reweighted_survey = run_calibrated_model(
        model,
        name = 'Sorsogon_Electricity',
        population_size = 32694,
        iterations = 100000)
```

microsim.run.**run_composite_model**(*model*, *sufix*, *population_size=False*, *err='wf'*, *iterations=1000*, *align_census=True*, *name='noname'*, *census_file='data/benchmarks.csv'*, *drop_col_survey=False*, *verbose=False*, *from_script=False*, *k={}*, *year=2010*, *sigma_mu=False*, *sigma_sd=0.1*, *njobs=2*, *to_cat=False*, *to_cat_census=False*, *reweight=True*, ***kwargs*)

Run and calibrate a single composite model.

Parameters

- **model** (*dict*) – Dictionary containing model parameters.
- **sufix** (*str*) – model *name* sufix.
- **name** (str, optional) –
- **population_size** (int, optional) – Defaults to 1000.
- **err** (str) – Weight to use for error optimization. Defaults to *'wf'*.
- **iterations** (int, optional) – Number of sample iterations on MCMC model. Defaults to *100*.
- **census_file** (str, optional) – Define census file with aggregated benchmarks. Defaults to *'data/benchmarks.csv'*.
- **drop_col_survey** (list, optional) – Columns to drop from survey. Defaults to *False*.
- **verbose** (bool, optional) – Be verbose. Defaults to *False*.
- **from_script** (bool, optional) – Run reweighting algorithm from file. Defaults to *Fasle*.
- **k** (dict, optional) – Correction k-factor. Default 1.
- **year** (int, optional) – year in *census_file* (i.e. index) to use for the model calibration *k* factor.
- **to_cat** (bool, optional) – Convert survey variables to categorical variables. Default to *False*.
- **to_cat_census** (bool, optional) – Convert census variables to categorical variables. Default to *False*.
- **reweight** (bool, optional) – Reweight sample. Default to *True*.

Returns

Returns a list containing the estimated k-factors as *model.PopModel.aggregates.k* and the reweighted survey as *model.PopModel.aggregates.survey*.

Return type result (list of objects)

Examples

```
>>> k_out, reweighted_survey = run_composite_model(model, sufix)
```

microsim.run.**transition_rate**(*start_rate*, *final_rate*, *as_array=True*, *start=False*, *end=False*, *default_start=2010*, *default_end=2030*)

Construct growth rates.

Parameters

- **model_in** (*dict*) – Model defines as a dictionary, with specified variables as keys, containing the *table_model* for each variable and an optional formula.
- **err** (str) – Weight to use for error optimization. Defaults to *'wf'*.
- **log_level** (int, optional) – Logging level passed to pymc3 log-level.

Returns Numpy vector with transitions rates computed as a linear function.

Example

```
>>> Elec = transition_rate(
>>>     0, 0.4, default_start=2016, default_end=2025)
```

microsim.run.**reduce_consumption**(*file_name, penetration_rate, sampling_rules, reduction, sim_years=[2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030], method='reweight', atol=10, verbose=False, scenario_name='scenario 1'*)

Reduce consumption levels given a penetration rate and sampling rules.

Parameters

- **file_name** (*str*) – Sample file name or file name pattern.
- **penetration_rate** (*float*) – Penetration rate.
- **sampling_rules** (*dict*) – Sampling rules.
- **reduction** (*dict*) – Consumption reduction values for specific consumption variable.
- **atol** (int, optional) – Toleration level. Defaults to: 10.
- **verbose** (bool, optional) – Be verbose.
- **scenario_name** (str, optional) – Name of the scenario. Default to: 'scenario 1'.
- **method** (str, optional) – Defines the implemented simulation method. If *resample*, the function will read individual files based on *file_name* pattern. Requires a list of years to loop through, passes through variable *sim_years*. If *reweight*, the function will read the single file *file_name* and retrieve specific columns of this file for each simulation year. Default to: 'reweight'.
- **sim_years** (list) – List of simulation years to compute consumption reduction. Defaults to: *[y for y in range(2010, 2031)]*

Returns Pandas Data Frame with the reduced consumption levels based on sampling rules and penetration rates.

Example

```
>>> sampling_rules = {
>>>     "w_ConstructionType == 'ConstructionType_appt'": 10,
>>>     "e_sqm < 70": 10,
>>>     "w_Income > 13650": 10,
>>> }
>>> Elec = transition_rate(
>>>     0, 0.4, default_start=2016, default_end=2025)
>>> Water = transition_rate(
>>>     0, 0.2, default_start=2016, default_end=2025)
>>> pr = transition_rate(
>>>     0, 0.3, default_start=2016, default_end=2025)
>>> reduce_consumption(
>>>     file_name,
>>>     pr, sampling_rules,
>>>     {'Electricity':Elec, 'Water':Water},
>>>     method = 'resample',
>>>     scenario_name = "scenario 1")
```

microsim.util_plot.**cross_tab**(*a*, *b*, *year*, *file_name*, *split_a=False*, *split_b=False*, *print_tab=False*)

Print cross tabulation data.

Parameters

- **a** (*str*) –
- **b** (*str*) –
- **year** (*int*) –
- **file_name** (*str*) –
- **split_a** (bool, optional) –
- **split_b** (bool, optional) –
- **print_tab** (bool, optional) –

Returns cross tabulation table.

microsim.util_plot.**plot_data_projection**(*reweighted_survey*, *var*, *iterations='n.a.'*, *groupby=False*, *pr=False*, *scenario_name='scenario 1'*, *verbose=False*, *cut_data=False*, *col_num=1*, *col_width=10*, *aspect_ratio=4*, *unit='household'*, *start_year=2010*, *end_year=2030*, *benchmark_year=False*)

Plot projected data as total sum and as per capita.

Parameters

- **reweighted_survey** (*str*) – Base survey name. This is the reweighted survey resulting from the simulation.
- **var** (*list*) – Python list containing the name of the variables to be included in the plot.
- **iterations** (str, optional) – Number of simulation iterations, this variable is only used in the title of the plot. Default to *'n.a.'*.
- **groupby** (str, list, optional) –
- **pr** (str, optional) – Penetration rates.

- **scenario_name** (str, optional) – Name of scenario.
- **verbose** (bool, optional) – Be verbose, Default to *False*.
- **cut_data** (bool, optional) – Make quintiles, Default to *False*.
- **col_num** (int, optional) – Number of columns to plot. Default to *1*.
- **col_width** (int, optional) – Width of plot column. Default to *10*.
- **aspect_ratio** (int, optional) – Plot aspect ratio. Default to *4*.
- **unit** (str, optional) – y-label plot. Default to *household*.
- **start_year** (int) – Plot start year, defines start of x-axis. Default to *2010*.
- **end_year** (int) – Plot end year, defines end of x-axis. Default to *2030*.
- **bechmark_year** (int) – Defined benchmark year, plots a red dotted line on benchmark year. Default to *False*.

microsim.util_plot.**plot_error**(*trace_in, census_in, iterations, pop=False, skip=[], fit_col=[], weight='wf', fit_cols=['Income', 'Electricity', 'Water'], add_cols=False, verbose=False, plot_name=False, force_fit=True, is_categorical=True, wbins=50, wspace=0.2, hspace=0.9, top=0.91, year=2010, save_all=False*)

Plot modeling errro distribution

microsim.util_plot.**plot_projected_weights**(*trace_out*, *iterations*)

Plot projected weights.

microsim.util_plot.**plot_transition_rate**(*variables*, *name*, *benchmark_year=2016*, *start_year=2010*, *end_year=2030*, *title='Technology transition rates for {}'*, *title_p='Technology penetration rate for {}'*, *ylab='Transition rate'*, *ylab_p='Penetration rate'*, *ylim_a=(0, 1)*, *ylim_b=(0, 1)*)

Plot growth rates.

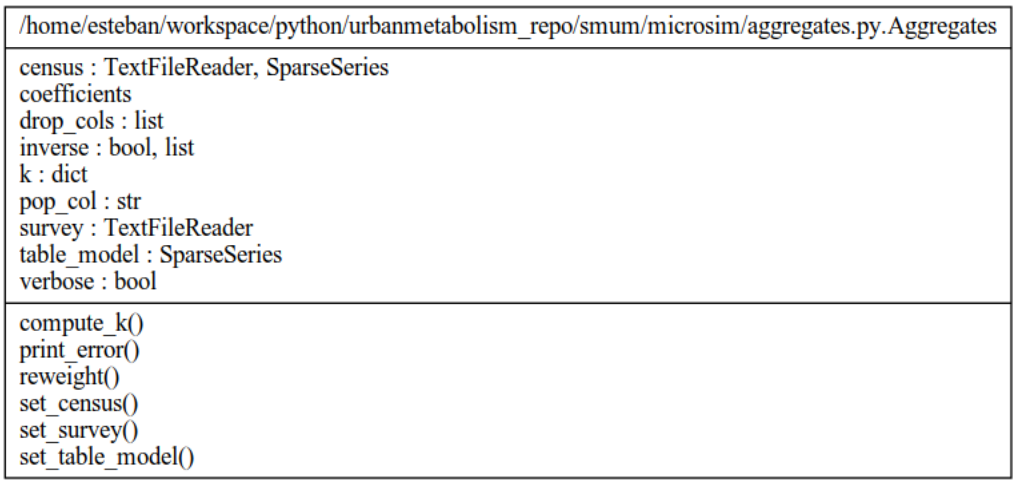


Fig. 6.1: Aggregates class diagram.

class microsim.aggregates.**Aggregates**(*verbose=False*, *pop_col='pop'*)

Class containing the aggregated data.

Parameters

- **pop_col** (str, optional) – Defaults to *'pop'*.
- **verbose** (bool, optional) – Defaults to *False*.

pop_col
str – Population column on census data.

verbose
bool – Be verbose.

k
dict – Dictionary containing the k-factors.

inverse
list – List of categories to invert.

drop_cols
list – List of columns to drop.

compute_k(*init_val=False*, *inter='Intercept'*, *prefix='e_'*, *year=2010*, *weight='wf'*, *var='Electricity'*)
Compute the k factor to minimize the error using the Newton-Raphson algorithm.

Parameters

- **init_val** (float, optional) – Estimated initial value passed to the Newton-Raphson algorithm for optimizing the error value. Defaults to *False*. If no value is given the function will try get the intercept value from the *table_model*. If the function does not find the intercept value on the *table_model* it will set it to 1.
- **inter** (str, optional) – Intercept name on *table_model*. Defaults to *'Intercept'*.
- **prefix** (str, optional) – Model prefix. Defaults to *'e_'*.
- **year** (int, optional) – year to be used from aggregates. Defaults to *2010*.
- **weight** (str, optional) – Weight value to use for optimizations. Defaults to *wf*.
- **var** (str, optional) – Variable to minimize the error for. Defaults to *'Electricity'*.

print_error(*var*, *weight*, *year=2010*, *lim=1e-06*)
Print computed error to command line.

Parameters

- **var** (*str*) – Variable name.
- **weight** (*str*) – Weight variable to use for the estimation of error.
- **year** (int, optional) – Year to compute the error for. Defaults to *2010*.
- **lim** (float, optional) – Limit for model to converge. Defaults to *1e-6*.

Returns

Computed error as: $\sum_i X_{i,var} * w_i * k_{var} - Tx_{var,year}$

Where:

- Tx are the known marginal sums for variable var.
- X is the generated sample survey. For each i record on the sample of variable var.
- w are the estimated new weights.
- k estimated correction factor for variable var.

Return type error (float)

reweight(*drop_cols*, *from_script=False*, *year=2010*, *max_iter=100*, *weights_file='temp/new_weights.csv'*, *script='reweight.R'*, *align_census=True*, ***kwargs*)
Reweight survey using GREGWT.

Parameters

- **drop_cols** (*list*) – list of columns to drop previous to the reweight.
- **from_script** (bool, optional) – runs the reweight from a script.
- **script** (str, optional) – script to run for the reweighting of the sample survey. *from_script* needs to be set to *True*. Defaults to *'reweight.R'*
- **weights_file** (str, optional) – Only required if reweight is run from script. Defaults to *'temp/new_weights.csv'*

set_census(*census*, *total_pop=False*, *to_cat=False*, ***kwargs*)
define census.

Parameters

- **census** (*str, pandas.DataFrame*) – Either census data as *pandas.DataFrame* or name of a file as *str*.
- **total_pop** (int, optional) – Total population. Defaults to *False*.
- ****kwargs** – Optional kword arguments for reading data from file, only used if *census* is a file.

Raises

- TypeError – If *census* is neither not a string or a DataFrame.
- ValueError – If *census* is not a valid file.

set_survey(*survey*, *inverse=False*, *drop=False*, *to_cat=False*, ***kwargs*)
define survey.

Parameters

- **survey** (*str, pandas.DataFrame*) – Either survey data as *pandas.DataFrame* or name of a file as *str*.
- **inverse** (bool, optional) – Defaults to *False*.
- **drop** – (bool, optional): Defaults to *False*.
- **to_cat** – (bool, optional): Defaults to *False*.
- ****kwargs** – Optional kword arguments for reading data from file, only used if *survey* is a file.

Raises

- TypeError – If *survey* is neither not a string or a DataFrame.
- ValueError – If *survey* is not a valid file.

set_table_model(*input_table_model*)
define table_model.

Parameters **input_table_model** (*list, pandas.DataFrame*) – input *table_model* either as a list of *pandas.DataFrame* or as a single *pandas.DataFrame*.

Raises ValueError – if *input_table_model* is not a list of *pandas.DataFrame* or a single *pandas.DataFrame*.

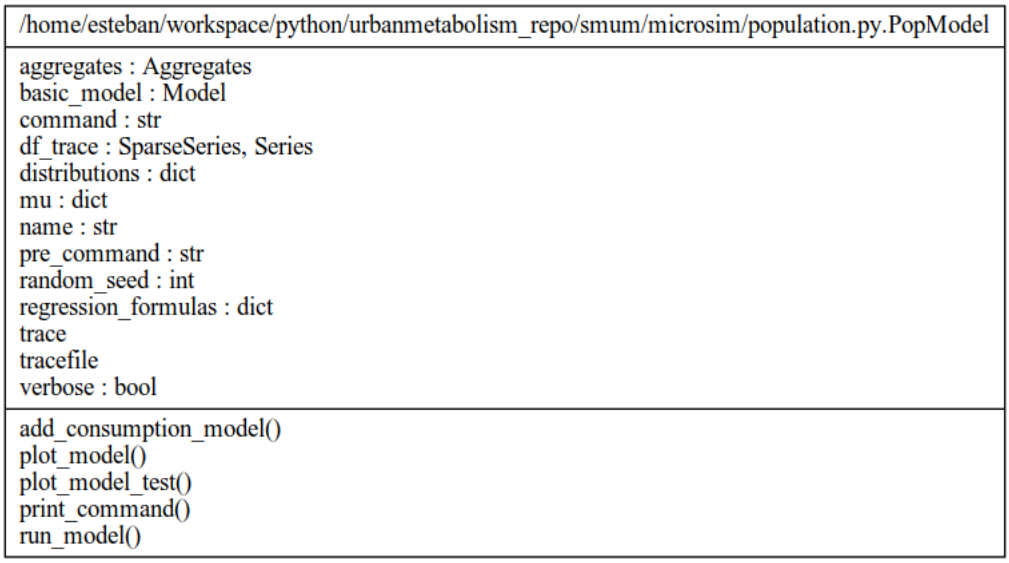


Fig. 6.2: Population class diagram.

class microsim.population.**PopModel**(*name='noname'*, *verbose=False*, *random_seed=12345*)

Main population model class.

add_consumption_model(*yhat_name, table_model, k_factor=1, sigma_mu=False, sigma_sd=0.1, prefix=False, bounds=[-inf, inf], constant_name='Intercept', formula=False*)

Define new base consumption model.

plot_model()

Model traceplot.

plot_model_test(*yhat_name*, *mu_var*, *sd_var*)

Model test plot

print_command()

Print computed command.

run_model(*iterations=100000*, *population=False*, *burn=False*, *thin=2*, *njobs=2*, ***kwargs*)

Run the model.

class microsim.table.**TableModel**(*census_file=False*, *verbose=False*, *normalize=True*)

Static and dynamic table model.

add_formula(*formula*, *name*)

add formula to table model.

add_model(*table*, *name*, *index_col=0*, *reference_cat=[]*, *static=False*, *skip_cols=[]*, ***kwargs*)

Add table model.

from_excel(*file_name*, *name*)

read table from excel file.

make_model()

prepare model for simulation.

print_formula(*name*)

pretty print table_model formula.

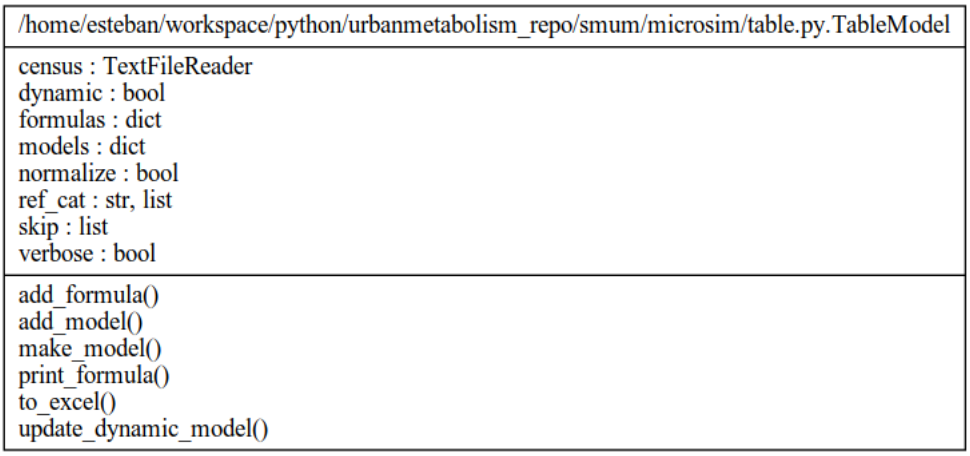


Fig. 6.3: Table class diagram.

to_excel(*sufix=''*, *var=False*, *year=False*, ***kwargs*)
Save table model as excel file.

update_dynamic_model(*name*, *val='p'*, *static=False*, *sd_variation=0.01*, *select=False*, *specific_col_as=False*, *specific_col=False*, *compute_average=True*)
Update dynamic model.

CHAPTER 7

Authors

- Dr. M. Esteban Muñoz H.

CHAPTER 8

Contributing

CHAPTER 9

History

- Fri 27 Oct 2017 03:34:12 PM CEST Documentation on readthedocs.
- Tue 20 Feb 2018 03:49:00 PM CET version 0.2.1
- Wed 28 Feb 2018 01:28:39 PM CET: 1. Restructured project. 2. Change library name to SMUM 3. Add jupyter notebook examples to documentation

Index

Symbols

A

C

D

E

F

G

I

K

M

N

P

R

S

T

U

V

W