Beatrix Haddock

8/26/2022

# Viviarum\_census\_synth\_pop

#### SE-side project overview.

The research-side docs for this model are located [here](file:///C:\Users\beatrixh\Documents\SE-side%20project%20overview).

# Background data

The primary datasets that this sim is built off of are a household dataset and a person dataset from PUMS. PUMS is a census product/survey of the US with much more detailed info than the decennial census. Further info [here](https://www.census.gov/programs-surveys/acs/microdata.html). A data dictionary for the two datasets can be found [here](https://www2.census.gov/programs-surveys/acs/tech_docs/pums/data_dict/PUMS_Data_Dictionary_2019.pdf). A notebook checking out the data can be found [here](https://github.com/ihmeuw/vivarium_census_prl_synth_pop/blob/main/notebooks/2022_04_05_PRL_explore_acs_data.ipynb).

The variables of interest to us (thus far) are:

HOUSEHOLDS\_COLUMN\_MAP = {

"ST": "state",

"SERIALNO": "census\_household\_id",

"PUMA": "puma",

"WGTP": "household\_weight",

}

PERSONS\_COLUMNS\_MAP = {

"ST": "state",

"SERIALNO": "census\_household\_id",

"AGEP": "age",

"RELSHIPP": "relation\_to\_household\_head",

"SEX": "sex",

"HISP": "latino",

"RAC1P": "race",

}

(note these are of the form “Census PUMS name”: “our name for the variable”)

The person file and the households file have different columns (e.g., the households file has info on income, and whether or not there is a washer-dryer, etc.). There is also a household id (SERIALNO) in the person file matching it to the household file. For some reason not all household\_ids are represented in the person file, so we subset the household data down to households represented in the person file.

# Vivarium Synthpop Component Guide

#### Overview:

Vivarium\_census\_synth\_pop is a model that initializes simulants with the following characteristics:

* Age
* Sex
* Race/ethnicity
* First name
* Middle name
* Last name
* Household id
* Relation to household head (e.g., spouse, brother, ...)
* Household address
* Zipcode
* PUMA (this is a geographic unit of a similar mesh to zipcodes)
* State (e.g., Florida)
* SSN
* Parent\_id (currently applies only to simulants born during the sim)
* Last\_birth\_time (currently applies only to simulants who have given birth during the sim)
* Employer\_id
* Employer address
* Employer zipcode

On each timestep:

* Simulants can:
  + Be born
  + Die
  + Move to a new address
  + Move into a new household (it has to be a household that already exists; no new households are created at this time)
  + Change employers if >= 18 years old (this includes becoming unemployed)
* Households can:
  + Move to a new address
* Businesses can:
  + Move to a new address

has the following components:

1. Population
2. Mortality
3. Fertility
4. PersonMigration
5. HouseholdMigration
6. Businesses

Along with a some fake data generators.

## Population

This is the component that initializes our population, and manages the following columns:

* "household\_id",
* "state",
* "puma",
* "relation\_to\_household\_head",
* "sex",
* "age",
* "race\_ethnicity",
* "first\_name",
* "middle\_name",
* "last\_name",
* "ssn",
* "alive",
* "entrance\_time",
* "exit\_time",

Note that typical vivarium models use vivarium\_public\_health.BasePopulation instead of a custom population component. We have a custom population.py because rather than initializing a population from scratch, we are sampling a population from census data.

This component has two event-driven methods, on initialize\_simulants, and time\_step\_\_cleanup.

#### Initialize\_simulants

At the start of the simulation: given a desired pop size N, this component samples N households from the census household file. Because many households have more than one person, this results in sampling far more than N people. We select the household with the (N+1)th person in it, and throw away that household and all households below that one. This results in some selection of M households with N\_hat <= N simulants.

On a time step: we check which simulants give birth. We then initialize simulants for all parents who gave birth. The following traits of the mother are inherited by the newborn:

inherited\_traits = ["household\_id", "state", "puma", "race\_ethnicity", “relation\_to\_household\_head", "last\_name", "alive", "tracked"]

The remaining traits are randomly generated (i.e. sex) or determined (age = 0.0).

#### On\_time\_step\_\_prepare

Simulants are aged the length of the time step

## Mortality

This is mostly vph’s mortality component. We had to overwrite the \_get\_population\_view method so that it would return the columns we needed.

## Fertility

This is mostly vph’s FertilityAgeSpecificRates. Given the list of all simulants eligible to give birth on a given time step, (for us, that is female and within some age range and didn’t give birth in the last 9 months) this uses GBD’s ASFR to select which sims will give birth.

## PersonMigration

This is responsible for individuals moving into new households. This is instantiated via updating the “household\_id”, “address”, “zipcode”, and “relation\_to\_household\_head” columns in the pop table.

Note that people are only eligible to move if their relationship to household head is not “Reference person”. Note additionally that people never move into brand new households; they always move into existing households, and their relationship to household head becomes “Other nonrelative”.

This component takes a data input in the form of a table with age, sex, race/eth, and then a move\_rate column. At the moment, this is fake data. In the future Abie will give us real data, and it may have more dependent variables.

This component has one event-driven method:

**On\_time\_step**, everyone who is available to move (non-“Reference person”s) are selected at the rate specified by our input data. Then, for each of those people who move, we choose an existing household\_id uniformly at random from all household\_ids other than the one they currently live in. Sims are then moved to those new household\_ids, and their addresses and zipcodes are updated.

Note that we ought to also update their PUMA, but we don’t currently have a map between PUMAs and zipcodes, so we don’t know yet how to update the PUMAs.

## HouseholdMigration

This is responsible for household addresses/zipcodes. It perhaps ought to be renamed to Households or HouseholdAddresses.

This component creates the columns [“address”, “zipcode”]. It takes a data input in the form of a table with age, sex, race/eth, and then a household move rate. As above, this is currently fake data. In the future Abie will give us real data, and it may have more dependent variables.

This component has two event-driven methods:

#### On\_initialize\_simulants

At the start of the simulation: an address and zipcode is assigned to every household\_id in the population table.

When new simulants are born mid-simulation: those newborns are assigned the address and zipcode of their mothers

#### On\_time\_step

From all households, those that will move to a new address on this time step are selected at the rate specified by our input data. Then, each household that moves is assigned a brand-new address and zipcode. Unless via low probability of RNG generating the same address/zipcode twice, no one will ever move into an address/zipcode that has previously been used in this sim.

## Businesses

This component maintains the employers of all sims in the pop table. It creates the following columns: [“employer\_id”, “employer\_name”, “employer\_address”, “employer\_zipcode”].

It takes three data inputs:

* job\_change\_rate\_data. This is the rate at which a sim changes jobs. Currently, it’s just a scalar. In the future we expect to get data from abie such that this rate is dependent on age, sex, race/eth, maybe other variables.
* move\_rate\_data. This is the rate at which a business changes addresses. Currently, it’s just a scalar. In the future we expect to get more detailed data from abie.
* The proportion of working-age people who are employed

At the start of the sim, we generate a table of businesses. For now, this is a complete list of businesses that never go out of business / no new ones are ever created. This table includes the variables [“employer\_id”, “employer\_name”, “employer\_address”, “employer\_zipcode”,”prevalence”]. Note that by “prevalence” we mean the proportion of the workforce that is working for a given employer. Thinking about in terms of prevalence vs. incidence, but might rename later.

We have both “known employers” and “random employers”. Known employers are those we want to be able to keep track of: for now, we’re thinking specifically about “unemployed” and “military”. We add to the table all known employers, and their “prevalence” among the working population. We then generate random employers to cover the remaining proportion of the workforce, such that the number of employees at each business fits a lognormal distribution.

It has two event-driven methods:

#### On\_initialize\_simulants

At the start of the sim, a table of businesses is generated, as described above. Then, everyone who is working age and older is assigned an employer at random (note this includes “unemployed”). Their employer\_address and employer\_zipcode are updated accordingly. Everyone under working age is assigned “unemployed”.

When new sims are born into the sim, they are assigned “unemployed”.

#### On\_time\_step

Using the move\_rate\_data, businesses who will move to a new address are selected. They are then assigned a new address and zipcode. The pop table and self.businesses table are updated accordingly.

Using the job\_change\_rate, we decide which sims who are already working-age that get a new job. We then select a new job based on the businesses “prevalences” from all employers that are not their current one. (note in the future might include an employer “incidence” rate, because this is currently incorrect. Prevalence should not equal incidence.)

For any sim who is turning working age during this time step, we choose a job for them from the table of businesses using prevalence as probability.

## Synthetic\_pii

This is a bit different from a normal module component. It’s a module we are using to generate fake names, addresses, and SSNs for sims. We had to make it a vivarium component in order to give it builder access.

It has three classes that are fairly self-explanatory:

* SSNGenerator
  + This generates SSNs without any input data; just RNG.
* NameGenerator
  + Generates first names by sampling from input data of first names. It takes sex and age as inputs/subsets accordingly.
  + Generates middle names by sampling from input data of first names. It takes sex and age as inputs and subsets accordingly.
  + Generates last names by sampling from input data of last names. Takes race\_ethnicity as an input and subsets input data accordingly. Hyphens and spaces are added at a race-ethnicity specified rate. Hyphenated or spaced names are just two last names randomly sampled and concatenated.
* AddressGenerator
  + Takes in the state (e.g., Florida) for which an addresses will be generated. Generates an address and zipcode by sampling the street number, street name, and unit independently and uniformly at random from a dataset of addresses across the US. Then samples the city and postal code independently and uniformly at random from cities and postal codes in the state for which we are generating an address.

The Population component uses the SSN and name generators. The HouseholdMigration and Businesses components use the AddressGenerator.