SynthPops

Release 1.10.4

Institute for Disease Modeling

Jul 05, 2021

Contents

SynthPops is used construct synthetic networks of people that satisfy statistical properties of real-world populations (such as the age distribution, household size, etc.). SynthPops can create generic populations with different network characteristics, as well as synthetic populations that interact in different layers of a multilayer contact network. These synthetic populations can then be used with agent-based models like COVID-19 Agent-based Simulator (Covasim) to simulate epidemics. SynthPops is available on GitHub. For more information on Covasim see Covasim on GitHub.

CHAPTER 1

Installation

Follow the instructions below to install SynthPops.

1.1 Requirements

Python 3.6 64-bit. (Note: Python 2 is not supported.)

We also recommend, but do not require, using Python virtual environments. For more information, see documentation for venv or Anaconda.

1.2 Installation

Complete the following steps to install SynthPops:

1. Fork and clone the SynthPops GitHub repository.
2. Open a command prompt and navigate to the SynthPops directory.
3. Run the following script:

```
python setup.py develop
```

Note: while *synthpops* can also be installed via pypi, this method does not currently include the data files which are required to function, and thus is not recommended.

1.3 Quick start guide

The following code creates a synthetic population for Seattle, Washington:

```
import synthpops as sp

sp.validate()

datadir = sp.datadir # this should be where your demographics data folder resides

location = 'seattle_metro'
state_location = 'Washington'
country_location = 'usa'
sheet_name = 'United States of America'
level = 'county'

npop = 10000 # how many people in your population
sp.generate_synthetic_population(npop,datadir,location=location,
                                 state_location=state_location,country_
↪location=country_location,
                                 sheet_name=sheet_name,level=level)
```

CHAPTER 2

SynthPops overview

Fundamentally, the population network can be considered a multilayer network with the following qualities:

- Nodes are people, with attributes like age.
- Edges represent interactions between people, with attributes like the setting in which the interactions take place (for example, household, school, or work). The relationship between the interaction setting and properties governing disease transmission, such as frequency of contact and risk associated with each contact, is mapped separately by Covasim or other *agent-based model*. SynthPops reports whether the edge exists or not.

If you are using SynthPops with Covasim, note that the relevant value in Covasim is the parameter **beta**, which captures the probability of transmission via a given edge per *time step*. The value of this parameter captures both number of effective contacts for disease transmission and transmission probability per contact.

The generated network is a multilayer network in the sense that it is possible for people to be connected by multiple edges each in different layers of the network. The layers are referred to as *contact layers*. For example, the *workplace contact layer* is a representation of all of the pairwise connections between people at work, and the *household contact layer* represents the pairwise connections between household members. Typically these networks are clustered; in other words, everyone within a household interacts with each other, but not with other households. However, they may interact with members of other households via their school or workplace. Some level of community contacts outside of these networks can be configured using Covasim or other model being used with SynthPops.

SynthPops functions in two stages:

1. Generate people living in households, and then assign individuals to workplaces and schools. Save the output to a cache file on disk. Implemented in `generate_synthetic_population()`.
2. Load the cached file and produce a dictionary that can be used by Covasim. Implemented in `make_population()`. Covasim assigns community contacts at random on a daily basis to reflect the random and stochastic aspect of contacts in many public spaces, such as shopping centers, parks, and community centers.

CHAPTER 3

What's new

Starting with SynthPops version 1.5.2, this file will document all changes to the codebase. By nature, SynthPops is a library to generate stochastic networked populations, so over time there will be model and code updates that change regression results. When these kinds of changes are made, we'll flag that here with the term "Regression Information". In addition, here are some other terms useful for understanding updates documented here.

3.1 Legend for changelog

- "Feature": a new feature previously unavailable.
- "Efficiency": a refactor of a previous method to make the calculation faster or require less memory.
- "Fix": a fix to a bug in the code base where a method either did not work under certain conditions or results were not as expected.
- "Deprecated": a method or feature that has been removed or support will be removed for in the future.
- "Regression Information": a change to the model or update to data resulted in a change to regression results.
- "Github Info": the associated PRs to any changes.

3.2 Latest versions (1.9.x – 1.10.x)

3.2.1 Version 1.10.4 (2021-07-05)

- Update to README to include the working title of the manuscript. We're doing this so that if people start using the model prior to the manuscript being available, they will still cite the work appropriately instead of using just the website.

3.2.2 Version 1.10.3 (2021-05-25)

- *Fix*: Addressing a bug when schools smaller than expected by the school size distributions are created in the case where there are not enough students left to place in the selected type of school.
- *Github Info*: PR 505

3.2.3 Version 1.10.2 (2021-05-22)

- Adding new json data files plus some example scripts to show users how to make their own json data files from individual raw data files for a given location.
- *Github Info*: PRs 494, 506

3.2.4 Version 1.10.1 (2021-05-20)

- *Fix*: Reinstating previous tests on methods within `sp.data.py` and updating `sp.load_location_from_json_str()` to take an optional parameter to control when checks on data loading are performed.
- *Github Info*: PR 502

3.2.5 Version 1.10.0 (2021-05-20)

- *Feature*: Ports Covasim's `People` class to SynthPops, and adds a new method to the `Pop` object, `to_people()`.
- While the differences are numerous, the major difference is that the `People` class stores data as NumPy arrays rather than as dicts or objects. This leads to performance improvements, at a cost of reduced flexibility. Most notable, `people.contacts` is a single edge list per layer, which is very fast to iterate over. Other quantities, such as `people['age']`, are also flat vectors.
- This functionality is not imported into the global SynthPops namespace; you can use it via `sp.people.People()` or `import synthpops.people as spp; spp.People()`.
- In future, these new classes and functions will be incorporated more tightly into the main SynthPops `Pop` class.
- *Github*: PR 497

3.2.6 Version 1.9.3 (2021-05-20)

- *Feature*: Minor feature update - plotting methods will now automatically search for location information to include in the figure titles. In order of *location*, *state_location*, *country_location*, `sp.plotting.plkwargs.make_title()` will look for the first available string to prefix the figure title.
- This update also changes behavior of some logging statements when using plotting methods. Instead of always sending users information about kwargs missing and the value of defaults being used in place, this behavior is now available under the debug mode for SynthPops.
- *Github Info*: PR 498

3.2.7 Version 1.9.2 (2021-05-20)

- *Fix*: Fix to how different layer classes get ages of members in the group or subgroups within. Specifically, this fixes how ages for members of schools and long term care facilities are calculated so that these layer classes can also call on the ages of members with specific roles in the class (i.e., students vs. teachers vs. non-teaching staff, or residents vs. staff). Tests have been added to verify these methods now work as expected.
- Slight reorganizing of module imports in `pop.py`
- *Github Info*: PR 495

3.2.8 Version 1.9.1 (2021-05-20)

- *Fix*: Fixing the logic in `sp.contact_networks.get_contact_counts_by_layer` so that it no longer returns an empty list in the dictionary counting contacts by layer group id, but rather returns lists populated with actual counts. Test assertions have also been added to catch this in case of future refactor work; see `test_plotting.py:test_plot_contact_counts_on_pop`.
- *Github Info*: PR 483

3.2.9 Version 1.9.0 (2021-05-16)

- Data folder cleaned up and removed individual csv data files now that synthpops has json data files instead for the collection of data used for each location.
- Json data objects also updated with documentation on the sources for the original and estimated data. When data have been estimated or inferred, to the best of our ability, we've added a note about this in the notes field.
- *Github Info*: PR 427

3.3 Versions 1.8.x (1.8.0 – 1.8.4)

3.3.1 Version 1.8.4 (2021-05-14)

- *Fix*: Catching rare events when schools are created with fewer than the smallest expected school size because there are no more students left to place in a school.
- *Feature*: Additional functionality to allow for the average classroom size to be different based on school mixing type (random, age_clustered, or age_and_class_clustered).
- Warning users when average class size and the average student teacher ratio parameters are incompatible as well as how synthpops handles these situations.
- *Fix*: Logic on how average class size and the average student teacher ratio parameters interact to create cohorts of students when the mixing type is age_and_class_clustered. The cohort size is drawn from a poisson on the larger of the two values. Why? Because for schools where students are cohorted into classrooms, there should be at least one teacher per classroom (average student teacher ratio), but there may be more than one (if average class size > average student teacher ratio).
- *Regression Information*: Refactoring related to schools as described above.
- *Github*: PR 459

3.3.2 Version 1.8.3 (2021-05-14)

- *Fix*: Refactored population generation methods to first determine the ages to be generated or expected to be generated, then have this be an input for methods to generate long term care facility residents' ages, and then methods to generate households and household member ages for the rest of the population residing in that layer. Addresses small n population bug identified with the household_method of 'fixed_ages' (issues 311 / 333) and allows for arbitrarily small populations (n > 0) to be created, although with smaller n matching the age distribution expected gets harder.
- *Fix*: Also fixes zero division errors when calculating pop properties like the enrollment and employment rates by age when there is at least one age with a count of zero people in the population (issue 383).
- Moved all household generation methods to sp.households
- Method to generate the count of household sizes for a fixed population renamed: `sp.households.generate_household_sizes_from_fixed_pop_size` -> `sp.households.generate_household_size_count_from_fixed_pop_size`
- `sp.households.generate_larger_household_sizes` generalized to all household sizes (now including size 1) in sp.households.generate_household_sizes
- `sp.households.generate_larger_household_head_ages` generalized to all household sizes (now including size 1) in `sp.households.generate_household_head_ages`
- New method: `sp.households.generate_age_count_multinomial`
- *Deprecated*: `sp.households.generate_household_head_age_by_size`, `sp.households.generate_living_alone`, `sp.households.generate_living_alone_method_2`
- *Regression Information*: Refactoring population generation methods to first determine the ages to be generated and then place people in residences produces a stochastic change in the regression population. Take a look at how the generated age distributions compare to the expected via pop.plot_ages().
- *Github Info*: PRs: 384

3.3.3 Version 1.8.2 (2021-05-12)

- *Fix*: Fix changes when constraints and other checks are performed in the data loading step. Now all checks should be performed only once after synthpops has checked the location and all of its parent locations for the necessary data to create the networked populations.
- *Github*: PR 485

3.3.4 Version 1.8.1 (2021-05-09)

- *Fix*: Minor fix to how the expected data are called when plotting the head of household age distributions by household size in `sp.plotting.plot_household_head_ages_by_size()`. Temporarily this method set the location parameter to None when the ability to traverse up parent locations was not yet functional. With that implemented now, we can keep information about all levels of the location and synthpops will look for the first data set available starting from the child location and moving upwards through all parent locations.
- *Github*: PR 478

3.3.5 Version 1.8.0 (2021-05-07)

- This is a big one!

- *Feature*: Class structures implemented for each layer and added to pop objects generated via *pop = sp.Pop()*. For example, now you can do `pop.get_household(i)` to get the household with integer `hhid` with value `i` which will be a `sp.Household` object with at minimum the attributes `hhid`, `member_uids`, `reference_uid`, and `reference_age`.
- Base class for layer groups available in `sp.base.py`; see class `sp.base.LayerGroup()` for more info. Important to note that this class has a method `member_ages()` which takes in a mapping of person ids to age to return the ages of individuals in a layer group. Optional parameter *subgroup_member_uids* allows you to return the ages for a subgroup of individuals.
- The specific layer classes implemented are `sp.Household`, `sp.School`, `sp.Classroom`, `sp.Workplace`, `sp.LongTermCareFacility`. Each is based off of `sp.LayerGroup`.
- Class also added for classroom structures in schools when schools are strictly cohorted into classrooms (school_mixing_type equals 'age_and_class_clustered').
- Method name changes: `sp.get_age_by_brackets_dic()` -> `sp.get_age_by_brackets()`, `sp.get_index_by_brackets_dic()` -> `sp.get_index_by_brackets()`, `sp.get_ids_by_age_dic()` -> `sp.get_ids_by_age()`, `sp.make_contacts_from_microstructure_objects()` -> `sp.make_contacts()`, `sp.get_contact_matrix_dic()` -> `sp.get_contact_matrices()`,
- `sp.make_contacts()` now returns a tuple; a dictionary version of the population and a dictionary version of schools to identify classrooms and other other groupings in schools. These are then used to populate the school and classroom structures in `sp.Pop.generate()`.
- *Regression Information*: Attribute names related to Long Term Care Facilities have changed to be more consistent with class name; `snfid` -> `ltcfid`, `snf_res` -> `ltcf_res`, `snf_staff` -> `ltcf_staff`.
- *Github*: PR 347

3.4 Versions 1.7.x (1.7.0 – 1.7.7)

3.4.1 Version 1.7.7 (2021-05-07)

- Made changes to allow SynthPops to be installed via `pip`.
- Updated examples in the folder `synthpops/examples`.
- Most significantly, changed the default data folder from `synthpops/data` to `synthpops/synthpops/data`.
- *Github*: PRs: 465

3.4.2 Version 1.7.6 (2021-05-05)

- Updated random graph model to use networkx's fast Erdos-Renyi graph generator implementation, which speeds up generation time for the model.
- *Regression Information*: The fast Erdos Renyi graph implementation changes the edges chosen, though not the statistical properties of the degree distribution.
- *Github*: PRs: 449

3.4.3 Version 1.7.5 (2021-05-03)

- `sp.contact_networks.get_contact_counts_by_layer()` now returns two dictionaries, one that gives the number of contacts between different roles in settings, like the number of contacts for students to teachers in schools, as well as the number of contacts per group in a setting, for example the number of contacts people have in the workplace with *wpid == 0*.
- `sp.sampling.statistic_test()` with *verbose = True* prints to screen details about the expected and actual distributions when the test fails.
- *Fix*: Default *n* value now assigned in `sp.defaults.py` when `sp.Pop` supplied *n = None* and when *n* is lower than `sp.defaults.default_pop_size`
- *Github*: PRs 435, 448

3.4.4 Version 1.7.4 (2021-04-21)

- *Feature*: new summary information added to pop objects: `pop.summary.average_age`, `pop.summary.layer_degrees`, `pop.summary.layer_stats`, and `pop.summary.layer_degree_description`, using the pandas DataFrame describe method. These give information on the overall degree distribution as well as the degree distribution by age for different layers generated using synthpops. Methods added to calculate these are generalized so in principle if other layers are added to the population post hoc or if connections change, these information can be re-calculated.
- Also added is `pop.summarize()` which will print to screen and return a string of a brief description of the population generated using SynthPops.
- *Github* : PR 442

3.4.5 Version 1.7.3 (2021-04-16)

- *Fix*: Restructured how default location parameters are stored; now moved from `sp.config.py` into a dictionary available from `sp.defaults.py`. Methods added in `sp.defaults.py` to reset these values to user specified information.
- *Deprecated*: `sp.get_config_data()` is no longer available. The data returned from that method are now simply stored as a dictionary available as `sp.defaults.default_data`. Previous globally available parameters, most of which were not in use: `sp.datadir`, `sp.localdatadir`, `sp.rel_path`, `sp.alt_rel_path`, `sp.default_country`, `sp.default_state`, `sp.default_location`, `sp.default_sheet_name`, `sp.alt_location`, `sp.default_household_size_1_included`, are either now stored in and accesible via `sp.defaults.py` or removed from use.
- *Github*: PRs 436, 438

3.4.6 Version 1.7.2 (2021-04-13)

- *Feature*: Re-enabled support of age distributions for any number of age brackets. Json data files have been updated to accomodate this flexibility.
- *Fix*: Catching division by zero when calculating enrollment, employment, etc. rates by age and the number of people in a given age is zero (can occur when population size is very small, e.g. n~200).
- *Github Info*: PRs 401, 422

3.4.7 Version 1.7.1 (2021-04-09)

- *Feature*: Added checks for probability distributions with methods `sp.check_all_probability_distribution_sums()`, `sp.check_all_probability_distrubution_nonnega` `sp.check_probability_distribution_sum()`, `sp.check_probability_distribution_nonnegative()` These check that probabilities sum to 1 within a tolerance level (0.05), and have all non negative values. Added method to convert data from pandas dataframe to json array style, `sp.convert_df_to_json_array()`. Added statistical test method `sp.statistic_test()`. Added method to count contacts, `sp.get_contact_counts_by_layer()`, and method to plot the results, `sp.plot_contact_counts()`. See `sp.contact_networks.get_contact_counts_by_layer()` for more details on the method.
- Added example of how to load data into the location json objects and save to file. See `examples/create_location_data.py` and `examples/modify_location_data.py`.
- *Github Info*: PRs 410, 413, 423

3.4.8 Version 1.7.0 (2021-04-05)

- *Efficiency*: Major refactor of data methods to read from consolidated json data files for each location and look for missing data from parent locations or alternatively json data files for default locations. Migration of multiple data files for locations into a single json object per location under the `data` directory. This will should make it easier to identify all of the available data per location and where missing data are read in from. Examples of how to create, change, and save new json data files will come in the next minor version update.
- *Feature*: Location data jsons now have fields for the data source, reference links, and citations! These fields will be fully populated shortly. Please reference the links provided for any data obtained from SynthPops as most population data are sourced from other databases and should be referenced as such.
- *Deprecated*: Refactored data methods no longer support the reading in of data from user specified file paths. Use of methods to read in age distributions aggregated to a number of age brackets not equal to 16, 18, or 20 (officially supported values) is currently turned off. Next minor update will re-enable these features. Old methods are available in *synthpops.data_distributions_legacy.py*, however this file will be removed in upcoming versions once we have migrated all examples to use the new data methods and have fully enabled all the functionality of the original data methods. Please update your usage of SynthPops accordingly.
- Updated documentation about the input data layers.
- *Github Info*: PRs 407, 303

3.5 Versions 1.6.x (1.6.0 – 1.6.2)

3.5.1 Version 1.6.2 (2021-04-01)

- *Feature*: Added new methods, `sp.get_household_head_ages_by_size()`, `sp.plot_household_head_ages_by_size()`. Also accessible pop methods as `pop.get_household_head_ages_by_size()`, `pop.plot_household_head_ages_by_size()`. These calculate the generated count the household head age by the household size, and the plotting methods compare this to the expected age distributions by size as matrices.
- *Github Info*: PR 385

3.5.2 Version 1.6.1 (2021-03-25)

- *Feature*: Added new methods, `sp.check_dist()` and aliases `sp.check_normal()` and `sp.check_poisson()`, to check whether the observed distribution matches the expected distribution.
- *Github Info*: PR 373

3.5.3 Version 1.6.0 (2021-03-20)

- *Feature*: Adding summary methods for SynthPops pop objects accesible as pop.summary and computed using pop.compute_summary(). Also adding several plotting methods for these summary data.
- Updating `sp.workplaces.assign_rest_of_workers()` to work off a copy of the workplace age mixing matrix so that the copy stored in SynthPops pop objects is not modified during generation.
- More tests for summary methods in pop.py, methods in config.py, plotting methods in plotting.py
- *Regression Information*: Adding new workplace size data specific for the Seattle metro area which changes the regression results. The previous data from the Washington state level and the new data for the metropolitan statistical area (MSA) of Seattle for the 2019 year are very similar, however the use of this data with random number generators does result in slight stochastic differences in the populations generated.
- *Github Info*: PRs 356, 357, 358, 360

3.6 Versions 1.5.x (1.5.2 – 1.5.3)

3.6.1 Version 1.5.3 (2021-03-16)

- *Deprecated*: Removing use of verbose parameter to print statements to use logger.debug() instead and removing the verbose parameter where deprecated.
- *Github Info*: PRs 363, 379, 380

3.6.2 Version 1.5.2 (2021-03-09)

- *Feature*: Added metadata to pop objects.
- Updated installation instructions and reference citation.
- *Github Info*: PRs 365, 351

CHAPTER 4

SynthPops algorithm

This topic describes the algorithm used by SynthPops to generate the connections between people in each of the *contact layers* for a given location in the real world. The fundamental algorithm is the same for homes, schools, and workplaces, but with some variations for each.

The method draws upon the following previously published models to infer high-resolution age-specific contact patterns in different physical settings and locations:

- Mossong et al. 2008
- Fumanelli et al. 2012
- Prem et al. 2017
- Mistry et al. 2020

The general idea is to use age-specific contact matrices that describe age mixing patterns for a specific population. By default, SynthPops uses Prem et al.'s (2017) matrices, which project inferred age mixing patterns from the POLYMOD study (Mossong et al. 2008) in Europe to other countries. However, user-specified contact matrices can also be implemented for customizing age mixing patterns for the household, school, and workplace settings (see the social contact data on Zenodo for other empirical contact matrices from survey studies).

The matrices represent the average number of contacts between people for different age bins (the default matrices use 5-year age bins). For example, a household of two individuals is relatively unlikely to consist of a 25-year-old and a 15-year-old, so for the 25-29 year age bin in the household layer, there are a low number of expected contacts with the 15-19 year age bin (c.f., Fig. 2c in Prem et al.).

CHAPTER 5

Using SynthPops

The overall SynthPops workflow is contained in `generate_synthetic_population()` and is described below. The population is generated through households, not a pool of people.

You can provide required data to SynthPops in a variety of formats including .csv, .txt, or Microsoft Excel (.xlsx).

1. Instantiate a collection of households with sizes drawn from census data. Populations cannot be created outside of the *household contact layer*.
2. For each household, sample the age of a "reference" person from data that maps household size to a reference person in those households. The reference person may be referred to as the head of the household, a parent in the household, or some other definition specific to the data being used. If no data mapping household size to ages of reference individuals are available, then the age of the reference person is sampled from the age distribution of adults for the location.
3. The age bin of the reference people identifies the row of the contact matrix for that location. The remaining household members are then selected by sampling an age for the distribution of contacts for the reference person's age (in other words, normalizing the values of the row and sampling for a column) and assigning someone with that age to the household.
4. As households are generated, individuals are given IDs.
5. After households are constructed, students are chosen according to enrollment data by age to generate the *school contact layer*.
6. Students are assigned to schools using a similar method as above, where we select the age of a reference person and then select their contacts in school from an age-specific contact matrix for the school setting and data on school sizes.
7. With all students assigned to schools, teachers are selected from the labor force according to employment data.
8. The rest of the labor force are assigned to workplaces in the *workplace contact layer* by selecting a reference person and their contacts using an age-specific contact matrix and data on workplace sizes.

5.1 Examples

Examples live in the *examples* folder. These can be run as follows:

- `python examples/make_generic_contacts.py`

  Creates a dictionary of individuals, each of whom are represented by another dictionary with their contacts contained in the `contacts` key. Contacts are selected at random with degree distribution following the Erdos-Renyi graph model.

- `python examples/generate_contact_network_with_microstructure.py`

  Creates and saves to file households, schools, and workplaces of individuals with unique IDs, and a table mapping IDs to ages. Two versions of each contact layer (households, schools, or workplaces) are saved; one with the unique IDs of each individual in each group (a single household, school or workplace), and one with their ages (for easy viewing of the age mixing patterns created).

- `python examples/load_contacts_and_show_some_layers.py`

  Loads a multilayer contact network made of three layers and shows the age and ages of contacts for the first 20 people.

In the *tests* folder, you can view the following to see examples of additional functionality.

- `test_synthpop.py`

  Reads in demographic data and generates populations matching those demographics.

- `test_contacts.py`

  Generates random contact networks with individuals matching demographic data or reads in synthetic contact networks with three layers (households, schools, and workplaces).

- `test_contact_network_generation.py`

  Generates synthetic contact networks in households, schools, and workplaces with Seattle Metro data (and writes to file).

The other topics in this section walk through the specific data sources and details about the settings for each of the *contact layers*.

5.1.1 Household contact layer

The *household contact layer* represents the pairwise connections between household members. The population is generated within this contact layer, not as a separate pool of people.

As locations, households are special in the following ways:

- Unlike schools and workplaces, everyone must be assigned to a household.
- The size of the household is important (for example, a 2-person household looks very different in comparison to a 5- or 6-person household) and some households only have 1 person.
- The reference person/head of the household can be well-defined by data.

Data needed

The following data sets are required for households:

1. **Age bracket distribution** specifying the distribution of people in age bins for the location. For example:

```
age_bracket , percent
0_4         , 0.0594714358950416
5_9         , 0.06031137308234759
10_14       , 0.05338015778985113
15_19       , 0.054500690394160285
20_24       , 0.06161403846144956
25_29       , 0.08899312471888453
30_34       , 0.0883533486774803
35_39       , 0.07780767611060545
40_44       , 0.07099017823587304
45_49       , 0.06996903280562596
50_54       , 0.06655242534751997
55_59       , 0.06350008343899961
60_64       , 0.05761405140489549
65_69       , 0.04487122889235999
70_74       , 0.030964420778483555
75_100       , 0.05110673396642193
```

2. **Age distribution of the reference person for each household size**

   The distribution is what matters, so it doesn't matter if absolute counts are available or not, each *row* is normalized. If this is not available, default to sampling the age of the reference individual from the age distribution for adults:

```
family_size , 18-20 , 20-24 , 25-29 , 30-34 , 35-39 , 40-44 , 45-49 , 50-54 , 55-
↪64 , 65-74 , 75-99
2           , 163   , 999   , 2316  , 2230  , 1880  , 1856  , 2390  , 3118  ,␣
↪9528  , 9345  , 5584
3           , 115   , 757   , 1545  , 1907  , 2066  , 1811  , 2028  , 2175  ,␣
↪3311  , 1587  , 588
4           , 135   , 442   , 1029  , 1951  , 2670  , 2547  , 2368  , 1695  ,␣
↪1763  , 520   , 221
5           , 61    , 172   , 394   , 905   , 1429  , 1232  , 969   , 683   , 623␣
↪  , 235   , 94
6           , 25    , 81    , 153   , 352   , 511   , 459   , 372   , 280   , 280␣
↪  , 113   , 49
7           , 24    , 33    , 63    , 144   , 279   , 242   , 219   , 115   , 157␣
↪  , 80    , 16
```

3. **Distribution of household sizes**:

```
household_size , percent
1              , 0.2781590909877753
2              , 0.3443313103056699
3              , 0.15759535523004006
4              , 0.13654311541644018
5              , 0.050887858718118274
6              , 0.019738368167953997
7              , 0.012744901174002305
```

4. **Household contact matrix** specifying the number/weight of contacts by age bin:

```
        0-10        , 10-20       , 20-30
0-10    0.659867911 , 0.503965302 , 0.214772978
10-20   0.314776879 , 0.895460015 , 0.412465791
20-30   0.132821425 , 0.405073038 , 1.433888594
```

   By default, SynthPops uses matrices from a study (Prem et al. 2017) that projected inferred age mixing patterns

from the POLYMOD study (Mossong et al. 2008) in Europe to other countries. SynthPops can take in user-specified contact matrices if other age mixing patterns are available for the household, school, and workplace settings (see the social contact data on Zenodo for other empirical contact matrices from survey studies).

In theory, the household contact matrix varies with household size, but in general data at that resolution is unavailable.

Workflow

Use these SynthPops functions to instantiate households as follows:

1. Call `generate_synthetic_population()` and provide the binned age bracket distribution data described above. This wrapper function calls the following functions:
    1. From the binned age distribution, `get_age_n()` creates samples of ages from the binned distribution, and then normalizes to create a single-year distribution. This distribution can therefore be gathered using whatever age bins are present in any given dataset.
    2. `generate_household_sizes_from_fixed_pop_size()` generates empty households with known size based on the distribution of household sizes.
    3. `generate_all_households()` contains the core implementation and constructs households with individuals of different ages living together. It takes in the remaining data sources above, and then does the following:
        - Calls `generate_living_alone()` to populate households with 1 person (either from data on those living alone or, if unavailable, from the adult age distribution).
        - Calls `generate_larger_households()` repeatedly with with different household sizes to populate those households, first sampling the age of a reference person and then their household contacts as outlined above.

5.1.2 School contact layer

The *school contact layer* represents all of the pairwise connections between people in schools, including both students and teachers. Schools are special in that:

- Enrollment rates by age determine the probability of individual being a student given their age.
- Staff members such as teachers are chosen from individuals determined to be in the adult labor force.
- The current methods in SynthPops treat student and worker status as mutually exclusive. Many young adults may be both students and workers, part time or full time in either status. The ability to select individuals to participate in both activities will be introduced in a later version of the model.

Data needed

The following data is required for schools:

1. **School size distribution**:

```
school_size , percent
0-50        , 0.2
51-100      , 0.1
101-300     , 0.3
```

2. **Enrollment by age** specifying the percentage of people of each age attending school. See `get_school_enrollment_rates()`, but note that this mainly implements parsing a Seattle-specific data file to produce the following data structure, which could equivalently be read directly from a file:

```
age , percent
0   , 0
1   , 0
2   , 0
3   , 0.529
4   , 0.529
5   , 0.95
6   , 0.95
7   , 0.95
8   , 0.95
9   , 0.95
10  , 0.987
11  , 0.987
12  , 0.987
13  , 0.987
```

3. **School contact matrix** specifying the number/weight of contacts by age bin. This is similar to the household contact matrix. For example:

```
        0-10        , 10-20       , 20-30
0-10    0.659867911 , 0.503965302 , 0.214772978
10-20   0.314776879 , 0.895460015 , 0.412465791
20-30   0.132821425 , 0.405073038 , 1.433888594
```

4. **Employment rates by age**, which is used when determining who is in the labor force, and thus which adults are available to be chosen as teachers:

```
Age , Percent
16  , 0.496
17  , 0.496
18  , 0.496
19  , 0.496
20  , 0.838
21  , 0.838
22  , 0.838
```

5. **Student teacher ratio**, which is the average ratio for the location. Methods to use a distribution or vary the ratio for different types of schools may come in later developments of the model:

```
student_teacher_ratio=30
```

Typically, contact matrices describing age-specific mixing patterns in schools include the interactions between students and their teachers. These patterns describe multiple types of schools, from possibly preschools to universities.

Workflow

Use these SynthPops functions to implement the school contact layer as follows:

1. `get_uids_in_school()` uses the enrollment rates to determine which people attend school. This then provides the number of students needing to be assigned to schools.
2. `generate_school_sizes()` generates schools according to the school size distribution until there are enough places for every student to be assigned a school.

3. `send_students_to_school()` assigns specific students to specific schools.
    - This function is similar to households in that a reference student is selected, and then the contact matrix is used to fill the remaining spots in the school.
    - Some particulars in this function deal with ensuring a teacher/adult is less likely to be selected as a reference person, and restricting the age range of sampled people relative to the reference person so that a primary school age reference person will result in the rest of the school being populated with other primary school age children
4. `get_uids_potential_workers()` selects teachers by first getting a pool of working age people that are not students.
5. `get_workers_by_age_to_assign()` further filters this population by employment rates resulting in a collection of people that need to be assigned workplaces.
6. In `assign_teachers_to_work()`, for each school, work out how many teachers are needed according to the number of students and the student-teacher ratio, and sample those teachers from the pool of adult workers. A minimum and maximum age for teachers can be provided to select teachers from a specified range of ages (this can be used to account for the additional years of education needed to become a teacher in many places).

5.1.3 Workplace contact layer

The *workplace contact layer* represents all of the pairwise connections between people in workplaces, except for teachers working in schools. After some workers are assigned to the *school contact layer* as teachers, all remaining workers are assigned to workplaces. Workplaces are special in that there is little/no age structure so workers of all ages may be present in every workplace.

Again, note that work and school are currently exclusive, because the people attending schools are removed from the list of eligible workers. This doesn't necessarily need to be the case though. In fact, we know that in any countries and cultures around the world, people take on multiple roles as both students and workers, either part-time or full-time in one or both activities.

Data required

The following data are required for generating the workplace contact layer:

1. **Workplace size distribution** - again, this gets normalized so can be specified as absolute counts or as normalized values:

```
work_size_bracket , size_count
1-4               , 2947
5-9               , 992
10-19             , 639
20-49             , 430
50-99             , 140
100-249           , 83
250-499           , 26
500-999           , 13
1000-1999         , 12
```

2. **Work contact matrix** specifying the number/weight of contacts by age bin. This is similar to the household contact matrix. For example:

```
        20-30       , 30-40       , 40-50
20-30   0.659867911 , 0.503965302 , 0.214772978
```

(continues on next page)

(continued from previous page)

```
30-40   0.314776879 , 0.895460015 , 0.412465791
40-50   0.132821425 , 0.405073038 , 1.433888594
```

Workflow

1. `generate_workplace_sizes()` generates workplace sizes according to the workplace size distribution until the number of workers is reached.
2. `assign_rest_of_workers()` populates workplaces just like for households and schools: randomly selecting the age of a reference person, and then sampling the rest of the workplace using the contact matrix.

5.1.4 Input data

Locations

SynthPops input data is organized around the concept of a *location*. Each location can have its own set of values for each of the input data fields or parameters.

Location hierarchy

Every location optionally has a parent location. The child location inherits all of the data field values from the parent. The child location can override the values inherited from the parent.

Input parameters

location_name The name of the location. This needs to be the same as the name of the file, leaving off the ".json" suffix.

```
"Senegal", "Senegal-Dakar", "usa", "usa-Washington", "usa-Washington-seattle_metro"
```

data_provenance_notices A list of strings. Each string in the list describes the provenance of some portion, or all, of the data in the file.

```
["This data originally comes from X, and co., 2015.", "Long term care facility (LTCF)␣
↪data source is XYZ."]
```

reference_links A list of strings. Each string in the list is a reference for some portion, or all, of the data in the file.

```
["https://github.com/awesomedata/awesome-public-datasets", "https://ingeniumcanada.
↪org/collection-research/open-data"]
```

citations A list of strings. Each string in the list is a citation for some portion, or all, of the data in the file.

```
["https://doi.org/10.1038/s41467-020-20544-y", "American Community Survey 2018:␣
↪Seattle-Tacoma-Bellevue, WA"]
```

notes A list of strings. Each string in the list is a note describing something about the dataset.

```
["Field X, row N, is missing from the source data and assumed to be default value Y.",
↪ "Data field Z is mising from the source data and assumed to have distribution A."]
```

parent The name of the parent location file, including the ".json" suffix.

```
"Senegal.json"
```

population_age_distribution_16

The 16-bracket version of population age distribution. A list of tuples of the form [min_age, max_age, percentage]. See next section for more info.

```
[
[0, 4, 0.0605381173520116],
[5, 9, 0.060734396722304],
...
[70, 74, 0.0312168948061224],
[75, 100, 0.0504085424578719]
]
```

population_age_distribution_18

The 18-bracket version of population age distribution. A list of tuples of the form [min_age, max_age, percentage]. See next section for more info.

```
[
[0, 4, 0.0605381173520116],
[5, 9, 0.060734396722304],
...
[80, 84, 0.0140175336124184],
[85, 100, 0.0166478127732105]
]
```

population_age_distribution_20

The 20-bin version of population age distribution. A list of tuples of the form [min_age, max_age, percentage]. See next section for more info.

```
[
[0, 4, 0.0605381173520116],
[5, 9, 0.060734396722304],
...
[90, 94, 0.00436],
[95, 100, 0.00236]
]
```

employment_rates_by_age

Employment rate by age. A list of tuples of the form [age, percentage].

```
[
[16, 0.3],
...
[25, 0.861],
...
[42, 0.838],
...
[68, 0.294],
...
[100, 0.061]
]
```

enrollment_rates_by_age

School enrollment rate by age. A list of tuples of the form [age, percentage].

```
[
...
[3, 0.529],
...
[10, 0.987],
...
[17, 0.977],
...
[24, 0.409],
...
[33, 0.113],
...
[48, 0.027],
...
[100, 0.0]
]
```

household_head_age_brackets

Age brackets for the household head age distribution. A list of tuples of the form [age_min, age_max].

```
[
[15, 19],
[20, 24],
[25, 29],
[30, 34],
[35, 39],
[40, 44],
[45, 49],
[50, 54],
[55, 59],
[60, 64],
[65, 69],
[70, 74],
[75, 79],
[80, 100]
]
```

household_head_age_distribution_by_family_size

A table providing the distribution of the age of the household head (sometimes referred to as the reference person), as a function of family size. Each row in this table specifies the distribution for a given family size. The family size is the first entry in the row. The remaining entries are, for each household head age bracket (see last table entry), the number or percentage of households with a household head in that age bracket.

```
[
[1, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0],
[2, 163.0, 999.0, 2316.0, 2230.0, 1880.0, 1856.0, 2390.0, 3118.0, 9528.0, 9345.0,␣
↪5584.0],
[3, 115.0, 757.0, 1545.0, 1907.0, 2066.0, 1811.0, 2028.0, 2175.0, 3311.0, 1587.0, 588.
↪0],
[4, 135.0, 442.0, 1029.0, 1951.0, 2670.0, 2547.0, 2368.0, 1695.0, 1763.0, 520.0, 221.
↪0],
[5, 61.0, 172.0, 394.0, 905.0, 1429.0, 1232.0, 969.0, 683.0, 623.0, 235.0, 94.0],
[6, 25.0, 81.0, 153.0, 352.0, 511.0, 459.0, 372.0, 280.0, 280.0, 113.0, 49.0],
[7, 24.0, 33.0, 63.0, 144.0, 279.0, 242.0, 219.0, 115.0, 157.0, 80.0, 16.0],
```

(continues on next page)

(continued from previous page)

```
[8, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
]
```

household_size_distribution

Specifies the distribution of household sizes. A list of tuples of the form [household_size, percentage].

```
[
[1, 0.2802338920169473],
[2, 0.3425558454571084],
[3, 0.154678770225653],
[4, 0.1261686577488611],
[5, 0.0589023321064863],
[6, 0.0228368983653579],
[7, 0.0146236040795857]
]
```

ltcf_resident_to_staff_ratio_distribution

Specifies the distribution of the ratio of long term care facility residents to staff. A list of tuples of the form [ratio_low, ratio_high, percentage].

```
[
...
[6.0, 6.0, 0.0227272727272727],
...
[9.0, 9.0, 0.25],
...
[14.0, 14.0, 0.0909090909090909]
]
```

ltcf_num_residents_distribution

Specifies the distribution of number of long term care facility residents in a facility. A list of tuples of the form [num_low, num_high, percentage].

```
[
...
[40.0, 59.0, 0.1343283582089552],
...
[120.0, 139.0, 0.1194029850746268],
...
[200.0, 219.0, 0.0149253731343283],
...
[300.0, 319.0, 0.0298507462686567],
...
[520.0, 539.0, 0.0149253731343283],
...
[680.0, 699.0, 0.0]
]
```

ltcf_num_staff_distribution

Specifies the distribution of number of long term care facility staff in a facility. A list of tuples of the form [num_low, num_high, percentage].

```
[
[0, 19,0.014925373134328358],
```

(continues on next page)

(continued from previous page)

```
...
[60, 79,0.1044776119402985],
...
[140, 159,0.11940298507462686],
...
[260, 279,0.04477611940298507],
...
[460, 479,0.014925373134328358],
...
[680, 699,0.0]
]
```

ltcf_use_rate_distribution

Specifies the distribution of percentage of population of a given age that uses long term care facilities. A list of tuples of the form [age, percentage].

```
[
...
[57.0, 0.0],
...
[63.0, 0.01014726],
...
[72.0, 0.00992606],
...
[84.0, 0.06078108],
...
[91.0, 0.18420189],
...
[100.0, 0.18420189]
]
```

school_size_brackets

Specifies the school size (number of students) brackets associated with the school size distribution data. A list of tuples of the form [school_size_low, school_size_hi].

```
[
[20, 50],
[51, 100],
[101, 300],
[301, 500],
[501, 700],
[701, 900],
[901, 1100],
[1101, 1300],
[1301, 1500],
[1501, 1700],
[1701, 1900],
[1901, 2100],
[2101, 2300],
[2301, 2700]
]
```

school_size_distribution

Specifies the percentage of schools for each school_size_bracket (see last table entry). A list of percentages, one for each entry in school_size_brackets.

```
[0.02752293577981651, 0.009174311926605502, 0.20183486238532117, 0.39449541284403683,␣
↪0.19266055045871566, 0.045871559633027505, 0.05504587155963302, 0.
↪036697247706422007, 0.009174311926605502, 0.0, 0.02752293577981651, 0.0, 0.0, 0.0]
```

school_size_distribution_by_type

Specifies the percentage of schools for each school_size_bracket, broken out by school type. A list of json objects with two keys 'school_type', and 'size_distribution'. The 'school_type' entry is a string. The 'size_distribution' entry is a list of percentages, one for each entry in school_size_brackets.

```
[{
"school_type": "ms",
"size_distribution": [0.0, 0.0, 0.0, 0.0, 0.4166666666666667, 0.16666666666666666, 0.
↪3333333333333333, 0.08333333333333333, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
}, {
"school_type": "hs",
"size_distribution": [0.06666666666666667, 0.06666666666666667, 0.13333333333333333,␣
↪0.0, 0.06666666666666667, 0.06666666666666667, 0.13333333333333333, 0.2, 0.
↪06666666666666667, 0.0, 0.2, 0.0, 0.0, 0.0]
}, {
"school_type": "uv",
"size_distribution": [0.10720338983050849, 0.06059322033898306, 0.15974576271186441,␣
↪0.27796610169491537, 0.22754237288135598, 0.07754237288135594, 0.024152542372881364,
↪ 0.016525423728813562, 0.013135593220338982, 0.013135593220338982, 0.
↪01016949152542373, 0.006355932203389832, 0.0046610169491525435, 0.
↪0012711864406779662]
}, {
"school_type": "pk",
"size_distribution": [0.0, 0.0, 0.22580645161290322, 0.6129032258064516, 0.
↪16129032258064516, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
}, {
"school_type": "es",
"size_distribution": [0.0, 0.0, 0.22580645161290322, 0.6129032258064516, 0.
↪16129032258064516, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
}]
```

school_types_by_age

Specifies the age ranges for each school type.

```
[{
"school_type": "pk",
"age_range": [3, 5]
}, {
"school_type": "es",
"age_range": [6, 10]
}, {
"school_type": "ms",
"age_range": [11, 13]
}, {
"school_type": "hs",
"age_range": [14, 17]
}, {
"school_type": "uv",
"age_range": [18, 100]
}]
```

workplace_size_counts_by_num_personnel

Specifies the count of workplaces broken down by number of workplace personnel.

```
[
[1, 4, 60050.0],
[5, 9, 19002.0],
[10, 19, 13625.0],
[20, 49, 9462.0],
[50, 99, 3190.0],
[100, 249, 1802.0],
[250, 499, 486.0],
[500, 999, 157.0],
[1000, 1999, 109.0]
]
```

16-, 18-, and 20-bracket versions of population age distributions.

The are different aggregations of the age distribution for a population for a variety of reasons. These kind of data come from sources like a national census website or survey sample and may be aggregated into age brackets (also referred to as groups or bins), or may be available for single years of age. The age brackets are also used to map other data such as age-specific contact matrices. Contact matrices of age mixing patterns are rarely available at a resolution of single years of age. Rather, they are most frequently available for age brackets. Currently, by default, SynthPops uses age-specific contact matrices aggregated to 16 age brackets and so we include the age distributions of locations aggregated to 16 age brackets, as well as other aggregations.

Specifically, for US sourced data we include the original US Census Bureau age distributions aggregated to 18 age brackets, and age distributions inferred for 20 age brackets from trend data to assist in infectious disease modeling of older ages. Where inferred or estimated, we include a note in the 'notes' field about the method used to infer or estimate the age distribution data.

Location File Format

```
todo
```

Example Input File

```
todo
```

CHAPTER 6

API reference

6.1 Subpackages

6.1.1 synthpops.people package

Submodules

synthpops.people.country_age_data module

This is the following file:

https://github.com/neherlab/covid19_scenarios/blob/master/src/assets/data/country_age_distribution.json

expressed as a Python file.

synthpops.people.household_size_data module

This is the following file (on average household size), expressed as a function:

https://population.un.org/household/exceldata/population_division_UN_Houseshold_Size_and_Composition_2019.xlsx

synthpops.people.loaders module

Load data

get_country_aliases()
Define aliases for countries with odd names in the data

map_entries(*json*, *location*)
Find a match between the JSON file and the provided location(s).

Parameters

- **json** (*list or dict*) – the data being loaded
- **location** (*list or str*) – the list of locations to pull from

show_locations(*location=None*, *output=False*)
Print a list of available locations.

Parameters

- **location** (*str*) – if provided, only check if this location is in the list
- **output** (*bool*) – whether to return the list (else print)

Examples:

```
sp.people.show_locations() # Print a list of valid locations
sp.people.show_locations('lithuania') # Check if Lithuania is a valid location
sp.people.show_locations('Viet-Nam') # Check if Viet-Nam is a valid location
```

New in version 1.10.0.

get_age_distribution(*location=None*)
Load age distribution for a given country or countries.

Parameters **location** (*str or list*) – name of the country or countries to load the age distribution for

Returns Numpy array of age distributions, or dict if multiple locations

Return type age_data (array)

New in version 1.10.0.

get_household_size(*location=None*)
Load average household size distribution for a given country or countries.

Parameters **location** (*str or list*) – name of the country or countries to load the age distribution for

Returns Size of household, or dict if multiple locations

Return type house_size (float)

New in version 1.10.0.

synthpops.people.makepop module

Defines functions for making the population.

make_people(*n=None*, *popdict=None*, *rand_seed=1*, *pop_type='synthpops'*, *location=None*, *save_pop=False*, *popfile=None*, *die=True*, *verbose=None*, ***kwargs*)
Make the actual people for the simulation. Usually called via sim.initialize(), but can be called directly by the user.

Parameters

- **n** (*int*) – the number of people to create
- **popdict** (*dict*) – if supplied, use this population dictionary instead of generating a new one
- **save_pop** (*bool*) – whether to save the population to disk

- **popfile** (*bool*) – if so, the filename to save to
- **die** (*bool*) – whether or not to fail if synthetic populations are requested but not available
- **verbose** (*bool*) – level of detail to print
- **kwargs** (*dict*) – passed to make_randpop() or make_synthpop()

Returns people

Return type people (*People*)

New in version 1.10.0.

make_randpop(*sim*, *use_age_data=True*, *use_household_data=True*, *sex_ratio=0.5*, *microstructure=False*)

Make a random population, with contacts.

This function returns a "popdict" dictionary, which has the following (required) keys:

- uid: an array of (usually consecutive) integers of length N, uniquely identifying each agent
- age: an array of floats of length N, the age in years of each agent
- sex: an array of integers of length N (not currently used, so does not have to be binary)
- contacts: list of length N listing the contacts; see make_random_contacts() for details
- layer_keys: a list of strings representing the different contact layers in the population; see make_random_contacts() for details

Parameters

- **sim** (*Sim*) – the simulation object
- **use_age_data** (*bool*) – whether to use location-specific age data
- **use_household_data** (*bool*) – whether to use location-specific household size data
- **sex_ratio** (*float*) – proportion of the population that is male (not currently used)
- **microstructure** (*bool*) – whether or not to use the microstructuring algorithm to group contacts

Returns a dictionary representing the population, with the following keys for a population of N agents with M contacts between them:

Return type popdict (dict)

New in version 1.10.0.

make_random_contacts(*pop_size*, *contacts*, *overshoot=1.2*, *dispersion=None*)

Make random static contacts.

Parameters

- **pop_size** (*int*) – number of agents to create contacts between (N)
- **contacts** (*dict*) – a dictionary with one entry per layer describing the average number of contacts per person for that layer
- **overshoot** (*float*) – to avoid needing to take multiple Poisson draws
- **dispersion** (*float*) – if not None, use a negative binomial distribution with this dispersion parameter instead of Poisson to make the contacts

Returns a list of length N, where each entry is a dictionary by layer, and each dictionary entry is the UIDs of the agent's contacts layer_keys (list): a list of layer keys, which is the same as the keys of the input "contacts" dictionary

Return type contacts_list (list)

New in version 1.10.0.

make_microstructured_contacts (*pop_size*, *contacts*)
Create microstructured contacts – i.e. for households

make_hybrid_contacts (*pop_size*, *ages*, *contacts*, *school_ages=None*, *work_ages=None*)
Create "hybrid" contacts – microstructured contacts for households and random contacts for schools and workplaces, both of which have extremely basic age structure. A combination of both make_random_contacts() and make_microstructured_contacts().

New in version 1.10.0.

parse_synthpop (*population*, *layer_mapping=None*, *community_contacts=0*)
Make a population using SynthPops, including contacts. Usually called automatically, but can also be called manually. Either a simulation object or a population must be supplied; if a population is supplied, transform it into the correct format; otherwise, create the population and then transform it.

Parameters

- **population** (*list*) – a pre-generated SynthPops population
- **layer_mapping** (*dict*) – a custom mapping from SynthPops layers to Covasim layers
- **community_contacts** (*int*) – create this many community contacts on average

New in version 1.10.0.

synthpops.people.people module

Alternate representation of a population as a People object. Originally based on the corresponding Covasim classes and functions.

class FlexPretty
Bases: sciris.sc_utils.prettyobj

A class that supports multiple different display options: namely obj.brief() for a one-line description and obj.disp() for a full description.

New in version 1.10.0.

disp (*output=False*)
Print or output verbose representation of the object

brief (*output=False*)
Print or output a brief representation of the object

class BasePeople
Bases: *synthpops.people.people.FlexPretty*

A class to handle all the boilerplate for people – note that as with the BaseSim vs Sim classes, everything interesting happens in the People class, whereas this class exists to handle the less interesting implementation details.

New in version 1.10.0.

keys ()
Get the keys that have been set

lock()
Lock the people object to prevent keys from being added

unlock()
Unlock the people object to allow keys to be added

summarize(*output=False*)
Print a summary of the people – same as brief

set(*key*, *value*, *die=True*)
Ensure sizes and dtypes match

get(*key*)
Convenience method – key can be string or list of strings

true(*key*)
Return indices matching the condition

false(*key*)
Return indices not matching the condition

defined(*key*)
Return indices of people who are not-nan

undefined(*key*)
Return indices of people who are nan

count(*key*)
Count the number of people for a given key

count_not(*key*)
Count the number of people who do not have a property for a given key

set_pars(*pars=None*)
Re-link the parameters stored in the people object to the sim containing it, and perform some basic validation.

layer_keys()
Get the available contact keys – try contacts first, then beta_layer

indices()
The indices of each people array

validate(*die=True*, *verbose=False*)

to_df()
Convert to a Pandas dataframe

to_arr()
Return as numpy array

person(*ind*)
Method to create person from the people

to_people()
Return all people as a list

from_people(*people*, *resize=True*)
Convert a list of people back into a People object

to_graph(*full_output=False*)
Convert all people to a networkx MultiDiGraph, including all properties of the people (nodes) and contacts (edges).

Parameters **full_output** (*bool*) – if true, return nodes and edges along with the graph object

init_contacts(*reset=False*)
Initialize the contacts dataframe with the correct columns and data types

add_contacts(*contacts*, *lkey=None*, *beta=None*)
Add new contacts to the array. See also contacts.add_layer().

make_edgelist(*contacts*)
Parse a list of people with a list of contacts per person and turn it into an edge list.

static remove_duplicates(*df*)
Sort the dataframe and remove duplicates – note, not extensively tested

class Person(*pars=None*, *uid=None*, *age=-1*, *sex=-1*, *contacts=None*)
Bases: `sciris.sc_utils.prettyobj`

Class for a single person. Note: this is largely deprecated since People is now based on arrays rather than being a list of people.

New in version 1.10.0.

class FlexDict
Bases: `dict`

A dict that allows more flexible element access: in addition to obj['a'], also allow obj[0]. Lightweight implementation of the Sciris odict class.

keys()

values()

items()

class Contacts(*layer_keys=None*)
Bases: *synthpops.people.people.FlexDict*

A simple (for now) class for storing different contact layers.

New in version 1.10.0.

add_layer(***kwargs*)
Small method to add one or more layers to the contacts. Layers should be provided as keyword arguments.

Example:

```
hospitals_layer = cv.Layer(label='hosp')
sim.people.contacts.add_layer(hospitals=hospitals_layer)
```

pop_layer(**args*)
Remove the layer(s) from the contacts.

Example:

```
sim.people.contacts.pop_layer('hospitals')
```

Note: while included here for convenience, this operation is equivalent to simply popping the key from the contacts dictionary.

to_graph()
Convert all layers to a networkx MultiDiGraph

Example:

```
import networkx as nx
sim = cv.Sim(pop_size=50, pop_type='hybrid').run()
G = sim.people.contacts.to_graph()
nx.draw(G)
```

class Layer(*label=None*, ***kwargs*)

Bases: *synthpops.people.people.FlexDict*

A small class holding a single layer of contact edges (connections) between people.

The input is typically three arrays: person 1 of the connection, person 2 of the connection, and the weight of the connection. Connections are undirected; each person is both a source and sink.

This class is usually not invoked directly by the user, but instead is called as part of the population creation.

Parameters

- **p1** (*array*) – an array of N connections, representing people on one side of the connection
- **p2** (*array*) – an array of people on the other side of the connection
- **beta** (*array*) – an array of weights for each connection
- **label** (*str*) – the name of the layer (optional)
- **kwargs** (*dict*) – other keys copied directly into the layer

Note that all arguments (except for label) must be arrays of the same length, although not all have to be supplied at the time of creation (they must all be the same at the time of initialization, though, or else validation will fail).

Examples:

```
# Generate an average of 10 contacts for 1000 people
n = 10_000
n_people = 1000
p1 = np.random.randint(n_people, size=n)
p2 = np.random.randint(n_people, size=n)
beta = np.ones(n)
layer = cv.Layer(p1=p1, p2=p2, beta=beta, label='rand')

# Convert one layer to another with extra columns
index = np.arange(n)
self_conn = p1 == p2
layer2 = cv.Layer(**layer, index=index, self_conn=self_conn, label=layer.label)
```

New in version 1.10.0.

members

Return sorted array of all members

meta_keys()

Return the keys for the layer's meta information – i.e., p1, p2, beta

validate()

Check the integrity of the layer: right types, right lengths

pop_inds(*inds*)

"Pop" the specified indices from the edgelist and return them as a dict. Returns in the right format to be used with layer.append().

Parameters **inds** (*int, array, slice*) – the indices to be removed

append(*contacts*)
Append contacts to the current layer.

Parameters **contacts** (*dict*) – a dictionary of arrays with keys p1,p2,beta, as returned from layer.pop_inds()

to_df()
Convert to dataframe

from_df(*df*, *keys=None*)
Convert from a dataframe

to_graph()
Convert to a networkx DiGraph

Example:

```
import networkx as nx
sim = cv.Sim(pop_size=20, pop_type='hybrid').run()
G = sim.people.contacts['h'].to_graph()
nx.draw(G)
```

find_contacts(*inds*, *as_array=True*)
Find all contacts of the specified people

For some purposes (e.g. contact tracing) it's necessary to find all of the contacts associated with a subset of the people in this layer. Since contacts are bidirectional it's necessary to check both P1 and P2 for the target indices. The return type is a Set so that there is no duplication of indices (otherwise if the Layer has explicit symmetric interactions, they could appear multiple times). This is also for performance so that the calling code doesn't need to perform its own unique() operation. Note that this cannot be used for cases where multiple connections count differently than a single infection, e.g. exposure risk.

Parameters

- **inds** (*array*) – indices of people whose contacts to return
- **as_array** (*bool*) – if true, return as sorted array (otherwise, return as unsorted set)

Returns a set of indices for pairing partners

Return type contact_inds (array)

Example: If there were a layer with - P1 = [1,2,3,4] - P2 = [2,3,1,4] Then find_contacts([1,3]) would return {1,2,3}

update(*people*, *frac=1.0*)
Regenerate contacts on each timestep.

This method gets called if the layer appears in `sim.pars['dynam_lkeys']`. The Layer implements the update procedure so that derived classes can customize the update e.g. implementing overdispersion/other distributions, random clusters, etc.

Typically, this method also takes in the `people` object so that the update can depend on person attributes that may change over time (e.g. changing contacts for people that are severe/critical).

Parameters **frac** (*float*) – the fraction of contacts to update on each timestep

class People(*pars*, *strict=False*, ***kwargs*)
Bases: *synthpops.people.people.BasePeople*

A class to perform all the operations on the people. This class is usually not invoked directly, but instead is created automatically by the sim. The only required input argument is the population size, but typically the full parameters dictionary will get passed instead since it will be needed before the People object is initialized.

Note that this class handles the mechanics of updating the actual people, while BasePeople takes care of housekeeping (saving, loading, exporting, etc.). Please see the BasePeople class for additional methods.

Parameters

- **pars** (*dict*) – the sim parameters, e.g. sim.pars – alternatively, if a number, interpreted as pop_size
- **strict** (*bool*) – whether or not to only create keys that are already in self.meta.person; otherwise, let any key be set
- **kwargs** (*dict*) – the actual data, e.g. from a popdict, being specified

::Examples:

```
ppl1 = cv.People(2000)

sim = cv.Sim()
ppl2 = cv.People(sim.pars)
```

New in version 1.10.0.

plot(*bins=None*, *width=1.0*, *alpha=0.6*, *fig_args=None*, *axis_args=None*, *plot_args=None*, *do_show=None*, *fig=None*)

Plot statistics of the population – age distribution, numbers of contacts, and overall weight of contacts (number of contacts multiplied by beta per layer).

Parameters

- **bins** (*arr*) – age bins to use (default, 0-100 in one-year bins)
- **width** (*float*) – bar width
- **font_size** (*float*) – size of font
- **alpha** (*float*) – transparency of the plots
- **fig_args** (*dict*) – passed to pl.figure()
- **axis_args** (*dict*) – passed to pl.subplots_adjust()
- **plot_args** (*dict*) – passed to pl.plot()
- **do_show** (*bool*) – whether to show the plot
- **fig** (*fig*) – handle of existing figure to plot into

plot_graph()

Convert to networkx and draw. WARNING: extremely slow for more than ~100 people!

Example:

```
pop = sp.Pop(n=50)
pop.to_people().plot_graph()
```

story(*uid*, **args*)

Print out a short history of events in the life of the specified individual.

Parameters

- **uid** (*int/list*) – the person or people whose story is being regaled
- **args** (*list*) – these people will tell their stories too

Example:

```
sim = cv.Sim(pop_type='hybrid', verbose=0)
sim.run()
sim.people.story(12)
sim.people.story(795)
```

synthpops.people.state_age_data module

This data is translated from the US Census CSV to JSON/Python format.

synthpops.people.utils module

Default values and mathematical utilities

find_contacts
Numba for Layer.find_contacts()

A set is returned here rather than a sorted array so that custom tracing interventions can efficiently add extra people. For a version with sorting by default, see Layer.find_contacts(). Indices must be an int64 array since this is what's returned by true() etc. functions by default.

choose
Choose a subset of items (e.g., people) without replacement.

Parameters

- **max_n** (*int*) – the total number of items
- **n** (*int*) – the number of items to choose

Example:

```
choices = cv.choose(5, 2) # choose 2 out of 5 people with equal probability␣
↪(without repeats)
```

choose_r
Choose a subset of items (e.g., people), with replacement.

Parameters

- **max_n** (*int*) – the total number of items
- **n** (*int*) – the number of items to choose

Example:

```
choices = cv.choose_r(5, 10) # choose 10 out of 5 people with equal probability␣
↪(with repeats)
```

n_multinomial (*probs*, *n*)
An array of multinomial trials.

Parameters

- **probs** (*array*) – probability of each outcome, which usually should sum to 1
- **n** (*int*) – number of trials

Returns Array of integer outcomes

Example:

```
outcomes = cv.multinomial(np.ones(6)/6.0, 50)+1 # Return 50 die-rolls
```

poisson
A Poisson trial.

Parameters **rate** (*float*) – the rate of the Poisson process

Example:

```
outcome = cv.poisson(100) # Single Poisson trial with mean 100
```

n_poisson
An array of Poisson trials.

Parameters

- **rate** (*float*) – the rate of the Poisson process (mean)
- **n** (*int*) – number of trials

Example:

```
outcomes = cv.n_poisson(100, 20) # 20 Poisson trials with mean 100
```

n_neg_binomial (*rate*, *dispersion*, *n*, *step=1*)
An array of negative binomial trials. See cv.sample() for more explanation.

Parameters

- **rate** (*float*) – the rate of the process (mean, same as Poisson)
- **dispersion** (*float*) – dispersion parameter; lower is more dispersion, i.e. 0 = infinite, ∞ = Poisson
- **n** (*int*) – number of trials
- **step** (*float*) – the step size to use if non-integer outputs are desired

Example:

```
outcomes = cv.n_neg_binomial(100, 1, 50) # 50 negative binomial trials with mean␣
↪100 and dispersion roughly equal to mean (large-mean limit)
outcomes = cv.n_neg_binomial(1, 100, 20) # 20 negative binomial trials with mean␣
↪1 and dispersion still roughly equal to mean (approximately Poisson)
```

6.2 Submodules

6.2.1 synthpops.base module

The module contains frequently-used functions that do not neatly fit into other areas of the code base.

class LayerGroup (***kwargs*)
Bases: dict

A generic class for individual setting group and some methods to operate on each.

Parameters **kwargs** (*dict*) – data dictionary for the setting group

Notes

Settings currently supported include : households (H), schools (S), workplaces (W), and long term care facilities (LTCF).

Class constructor for an base empty setting group.

Parameters ****member_uids** (*np.array*) – ids of group members

set_layer_group(***kwargs*)
Set layer group values.

validate(*layer_str=''*)
Check that information supplied to make a household is valid and update to the correct type if necessary.

member_ages(*age_by_uid*, *subgroup_member_uids=None*)
Return the ages of members in the layer group given the pop object.

Parameters

- **age_by_uid** (*np.ndarray*) – mapping of age to uid
- **subgroup_member_uids** (*np.ndarray, list*) – subgroup of uids to return ages for

Returns ages of members in group or subgroup

Return type nd.ndarray

norm_dic(*dic*)
Normalize the dictionary `dic`.

Parameters **dic** (*dict*) – A dictionary with numerical values.

Returns A normalized dictionary.

norm_age_group(*age_dic*, *age_min*, *age_max*)
Create a normalized dictionary for the range `age_min` to `age_max`, inclusive.

Parameters

- **age_dic** (*dict*) – A dictionary with numerical values.
- **age_min** (*int*) – The minimum value of the range for the dictionary.
- **age_max** (*int*) – The maximum value of the range for the dictionary.

Returns A normalized dictionary for keys in the range `age_min` to `age_max`, inclusive.

get_index_by_brackets(*brackets*)
Create a dictionary mapping each item in the value arrays to the key. For example, if brackets are age brackets, then this function will map each age to the age bracket or bin that it belongs to, so that the resulting dictionary will give index_by_brackets[age_index] = age bracket of age_index.

Parameters **brackets** (*dict*) – A dictionary mapping bracket or bin keys to the array of values that belong to each bracket.

Returns A dictionary mapping indices to the brackets or bins each index belongs to.

Return type dict

get_age_by_brackets(*age_brackets*)
Create a dictionary mapping age to the age bracket it falls in.

Parameters **age_brackets** (*dict*) – A dictionary mapping age bracket keys to age bracket range.

Returns A dictionary of age bracket by age.

Example

```
age_brackets = sp.get_census_age_brackets(sp.datadir,state_location='Washington',
↪country_location='usa')
age_by_brackets = sp.get_age_by_brackets(age_brackets)
```

get_ids_by_age(*age_by_id*)
Get lists of IDs that map to each age.

Parameters **age_by_id** (*dict*) – A dictionary with the age of each individual by their ID.

Returns A dictionary listing IDs for each age from a dictionary that maps ID to age.

count_ages(*popdict*)
Create an age count from a population dictionary.

Parameters **popdict** (*dict*) – dictionary defining population

Returns Dictionary of the age count of the population.

Return type dict

get_aggregate_ages(*ages*, *age_by_brackets*)
Create a dictionary of the count of ages by age brackets.

Parameters

- **ages** (*dict*) – A dictionary of age count by single year.
- **age_by_brackets** (*dict*) – A dictionary mapping age to the age bracket range it falls within.

Returns A dictionary of aggregated age count for specified age brackets.

Example

```
aggregate_age_count = sp.get_aggregate_ages(age_count, age_by_brackets)
aggregate_matrix = symmetric_matrix.copy()
aggregate_matrix = sp.get_aggregate_matrix(aggregate_matrix, age_by_brackets)
```

get_aggregate_matrix(*matrix*, *age_by_brackets*)
Aggregate a symmetric matrix to fewer age brackets. Do not use for homogeneous mixing matrix.

Parameters

- **matrix** (*np.ndarray*) – A symmetric age contact matrix.
- **age_by_brackets** (*dict*) – A dictionary mapping age to the age bracket range it falls within.

Returns A symmetric contact matrix (`np.ndarray`) aggregated to age brackets.

Example

```
age_brackets = sp.get_census_age_brackets(sp.settings_config.datadir,state_
↪location='Washington',country_location='usa')
age_by_brackets = sp.get_age_by_brackets(age_brackets)

aggregate_age_count = sp.get_aggregate_ages(age_count, age_by_brackets)
aggregate_matrix = symmetric_matrix.copy()
aggregate_matrix = sp.get_aggregate_matrix(aggregate_matrix, age_by_brackets_dic)

asymmetric_matrix = sp.get_asymmetric_matrix(aggregate_matrix, aggregate_age_
↪count)
```

get_asymmetric_matrix(*symmetric_matrix*, *aggregate_ages*)

Get the contact matrix for the average individual in each age bracket.

Parameters

- **symmetric_matrix** (*np.ndarray*) – A symmetric age contact matrix.
- **aggregate_ages** (*dict*) – A dictionary mapping single year ages to age brackets.

Returns A contact matrix (`np.ndarray`) whose elements `M_ij` describe the contact frequency for the average individual in age bracket `i` with all possible contacts in age bracket `j`.

Example

```
age_brackets = sp.get_census_age_brackets(sp.datadir,state_location='Washington',
↪country_location='usa')
age_by_brackets = sp.get_age_by_brackets(age_brackets)

aggregate_age_count = sp.get_aggregate_ages(age_count, age_by_brackets)
aggregate_matrix = symmetric_matrix.copy()
aggregate_matrix = sp.get_aggregate_matrix(aggregate_matrix, age_by_brackets)

asymmetric_matrix = sp.get_asymmetric_matrix(aggregate_matrix, aggregate_age_
↪count)
```

get_bin_edges(*size_brackets*)

Get the bin edges for size brackets.

Parameters **size_brackets** (*dict*) – dictionary mapping bracket or bin number to an array of the range of sizes

Returns An array of the bin edges.

get_bin_labels(*size_brackets*)

Get the bin labels from the values contained within each bracket or bin.

Parameters **size_brackets** (*dict*) – dictionary mapping bracket or bin number to an array of the range of sizes

Returns A list of bin labels.

count_values(*dic*)

Counter of values in the dictionary. Keys in the returned dictionary are values from the input dictionary.

Parameters **dic** (*dict*) – dictionary with sortable values

Returns Dictionary of the count of values.

Return type dict

count_binned_values(*dic*, *bins=None*)
Binned counter of values in the dictionary. Indices are the bin indices from the input bins.

Parameters

- **dic** (*dict*) – dictionary with sortable and binnable values
- **bins** (*array*) – array of bin edges

Returns Array of the count of values binned

Return type array

binned_values_dist(*dic*, *bins=None*)
Binned distribution of values in the dictionary. Indices are the bin indices from the input bins.

Parameters

- **dic** (*dict*) – dictionary with sortable and binnable values
- **bins** (*array*) – array of bin edges

Returns Array of the binned distribution of values.

Return type array

6.2.2 synthpops.config module

This module sets the location of the data folder and other global settings.

To change the level of log messages displayed, use e.g.

sp.logger.setLevel('CRITICAL')

checkmem(*unit='mb'*, *fmt='0.2f'*, *start=0*, *to_string=True*)
For use with logger, check current memory usage

version_info()

set_location_defaults(*country_location=None*)

set_datadir(*root_dir*, *relative_path=None*)
Set the data folder and relative path to the user-specified location.

On startup, the datadir and rel_path are set to the conventions used to store data. datadir is the root directory to the data, and relative_path is a list of sub directories to the data –> to change the location of the data the user is able to supply a new root_dir and new relative path. If the user uses a similar directory path model that we use e.g. root_dir/demographics/contact... the user can change datadir without changing relative path, by passing in relative_path = None (default) – note, mostly deprecated but still functional if needed.

Parameters

- **root_dir** (*str*) – new root directory for the data folder to point to
- **relative_path** (*str*) – new relative path to the root_dir

Returns path to the updated settings.datadir

Return type str

set_nbrackets(*n*)
Set the number of census brackets – usually 16, 18 or 20.

validate_datadir(*verbose=True*)
Check that the data folder can be found.

6.2.3 synthpops.contact_networks module

This module generates the household, school, and workplace contact networks.

make_contacts(*pop*, *age_by_uid*, *homes_by_uids*, *students_by_uid_lists=None*, *teachers_by_uid_lists=None*, *non_teaching_staff_uid_lists=None*, *workplace_by_uid_lists=None*, *facilities_by_uid_lists=None*, *facilities_staff_uid_lists=None*, *use_two_group_reduction=False*, *average_LTCF_degree=20*, *with_school_types=False*, *school_mixing_type='random'*, *average_class_size=20*, *inter_grade_mixing=0.1*, *average_student_teacher_ratio=20*, *average_teacher_teacher_degree=3*, *average_student_all_staff_ratio=15*, *average_additional_staff_degree=20*, *school_type_by_age=None*, *workplaces_by_industry_codes=None*, *max_contacts=None*)

From microstructure objects (dictionary mapping ID to age, lists of lists in different settings, etc.), create a dictionary of individuals. Each key is the ID of an individual which maps to a dictionary for that individual with attributes such as their age, household ID (hhid), school ID (scid), workplace ID (wpid), workplace industry code (wpindcode) if available, and contacts in different layers.

Parameters

- **age_by_uid** (*dict*) – dictionary mapping id to age for all individuals in the population
- **homes_by_uids** (*list*) – A list of lists where each sublist is a household and the IDs of the household members.
- **schools_by_uids** (*list*) – A list of lists, where each sublist represents a school and the ids of the students and teachers within it
- **teachers_by_uids** (*list*) – A list of lists, where each sublist represents a school and the ids of the teachers within it
- **workplaces_by_uids** (*list*) – A list of lists, where each sublist represents a workplace and the ids of the workers within it
- **facilities_by_uids** (*list*) – A list of lists, where each sublist represents a skilled nursing or long term care facility and the ids of the residents living within it
- **facilities_staff_uids** (*list*) – A list of lists, where each sublist represents a skilled nursing or long term care facility and the ids of the staff working within it
- **non_teaching_staff_uids** (*list*) – None or a list of lists, where each sublist represents a school and the ids of the non teaching staff within it
- **use_two_group_reduction** (*bool*) – If True, create long term care facilities with reduced contacts across both groups
- **average_LTCF_degree** (*int*) – default average degree in long term care facilities
- **with_school_types** (*bool*) – If True, creates explicit school types.
- **school_mixing_type** (*str or dict*) – The mixing type for schools, 'random', 'age_clustered', or 'age_and_class_clustered' if string, and a dictionary of these by school type otherwise. 'random' means random graphs for each school, 'age_clustered' means random graphs but with students mostly mixing within the age/grade (inter_grade_mixing controls mixing between grades), 'age_and_grade_clustered' means students cohorted into classes with their own teachers.
- **average_class_size** (*float*) – The average classroom size.
- **inter_grade_mixing** (*float*) – The average fraction of mixing between grades in the same school for clustered school mixing types.
- **average_student_teacher_ratio** (*float*) – The average number of students per teacher.

- **average_teacher_teacher_degree** (*float*) – The average number of contacts per teacher with other teachers.
- **average_student_all_staff_ratio** (*float*) – The average number of students per staff members at school (including both teachers and non teachers).
- **average_additional_staff_degree** (*float*) – The average number of contacts per additional non teaching staff in schools.
- **school_type_by_age** (*dict*) – A dictionary of probabilities for the school type likely for each age.
- **workplaces_by_industry_codes** (*np.ndarray or None*) – array with workplace industry code for each workplace
- **trimmed_size_dic** (*dict*) – If supplied, trim contacts on creation rather than post hoc.

Returns A popdict of people with attributes. Dictionary keys are the IDs of individuals in the population and the values are a dictionary for each individual with their attributes, such as age, household ID (hhid), school ID (scid), workplace ID (wpid), workplace industry code (wpindcode) if available, and the IDs of their contacts in different layers. Different layers available are households ('H'), schools ('S'), and workplaces ('W'), and long term care facilities ('LTCF'). Contacts in these layers are clustered and thus form a network composed of groups of people interacting with each other. For example, all household members are contacts of each other, and everyone in the same school is considered a contact of each other. If use_two_group_reduction is True, then contracts within 'LTCF' are reduced from fully connected.

Notes

Methods to trim large groups of contacts down to better approximate a sense of close contacts (such as classroom sizes or smaller work groups are available via sp.trim_contacts() or sp.create_reduced_contacts_with_group_types(): see these methods for more details).

If with_school_types==False, completely random schools will be generated with respect to the average_class_size, but other parameters such as average_additional_staff_degree will not be used.

create_reduced_contacts_with_group_types(*popdict*, *group_1*, *group_2*, *setting*, *average_degree=20*, *p_matrix=None*, *force_cross_edges=True*)

Create contacts between members of group 1 and group 2, fixing the average degree, and the probability of an edge between any two groups controlled by p_matrix if provided. Forces inter group edge for each individual in group 1 with force_cross_groups equal to True. This means not everyone in group 2 will have a contact with group 1.

Parameters

- **group_1** (*list*) – list of ids for group 1
- **group_2** (*list*) – list of ids for group 2
- **average_degree** (*int*) – average degree across group 1 and 2
- **p_matrix** (*np.ndarray*) – probability matrix for edges between any two groups
- **force_cross_groups** (*bool*) – If True, force each individual to have at least one contact with a member from the other group

Returns Popdict with edges added for nodes in the two groups.

Notes

This method uses the Stochastic Block Model algorithm to generate contacts both between nodes in different groups

and for nodes within the same group. In the current version, fixing the average degree and p_matrix, the matrix of probabilities for edges between any two groups is not supported. Future versions may add support for this.

get_contact_counts_by_layer(*popdict*, *layer='S'*, *with_layer_ids=False*)

Method to count the number of contacts for individuals in the population based on their role in a layer and the role of their contacts. For example, in schools this method can distinguish the number of contacts between students, teachers, and non teaching staff in the population, as well as return the number of contacts between all individuals present in a school. In a population with a school layer and roles defined as students, teachers, and non teaching staff, this method will return the number of contacts or edges for sc_students, sc_teachers, and sc_staff to sc_student, sc_teacher, sc_staff, all_staff, all. all_staff is the combination of sc_teacher and sc_staff, and all is all kinds of people in schools.

Parameters

- **popdict** (*dict*) – popdict of a Pop object, Dictionary keys are the IDs of individuals in the population and the values are a dictionary
- **layer** (*str*) – name of the physial contact layer: H for households, S for schools, W for workplaces, C for community, etc.
- **with_layer_ids** (*bool*) – If True, return additional dictionary on contacts by layer group id

Returns A dictionary with keys = people_types (default to ['sc_student', 'sc_teacher', 'sc_staff']) and each value is a dictionary which stores the list of counts for each type of contact: default to ['sc_student', 'sc_teacher', 'sc_staff', 'all_staff', 'all'] for example: contact_counter['sc_teacher']['sc_teacher'] store the counts of each teacher's contacts or edges to other teachers. If with_layer_ids is True: additionally return a dictionary with keys = layer_id (for example: scid, wpid...), and value is list of contact contacts.

Return type If with_layer_ids is False

filter_people(*pop*, *ages=None*, *uids=None*)

Helper function to filter people based on their uid and age.

Parameters

- **pop** (*sp.Pop*) – population
- **ages** (*list or array*) – ages of people to include
- **uids** (*list or array*) – ids of people to include

Returns An array of the ids of people to include for further analysis.

Return type array

count_layer_degree(*pop*, *layer='H'*, *ages=None*, *uids=None*, *uids_included=None*)

Create a dataframe from the population of people in the layer, including their uid, age, degree, and the ages of contacts in the layer.

Parameters

- **pop** (*sp.Pop*) – population
- **layer** (*str*) – name of the physial contact layer: H for households, S for schools, W for workplaces, C for community or other
- **ages** (*list or array*) – ages of people to include

- **uids** (*list or array*) – ids of people to include
- **uids_included** (*list or None*) – pre-calculated mask of people to include

Returns A pandas DataFrame of people in the layer including uid, age, degree, and the ages of contacts in the layer.

Return type pandas.DataFrame

compute_layer_degree_description(*pop*, *layer='H'*, *ages=None*, *uids=None*, *uids_included=None*, *degree_df=None*, *percentiles=None*)

Compute a description of the statistics for the degree distribution by age for a layer in the population contact network. See pandas.Dataframe.describe() for more details on all of the statistics included by default.

Parameters

- **pop** (*sp.Pop*) – population
- **layer** (*str*) – name of the physial contact layer: H for households, S for schools, W for workplaces, C for community or other
- **ages** (*list or array*) – ages of people to include
- **uids** (*list or array*) – ids of people to include
- **uids_included** (*list or None*) – pre-calculated mask of people to include
- **degree_df** (*dataframe*) – pandas dataframe of people in the layer and their uid, age, degree, and ages of their contacts in the layer
- **percentiles** (*list*) – list of the percentiles to include as statistics

Returns A pandas DataFrame of the statistics for the layer degree distribution by age.

Return type pandas.DataFrame

random_graph_model(*uids*, *average_degree*, *seed=None*)

Generate edges for a group of individuals given their ids from an Erdos-Renyi random graph model given the expected average degree.

Parameters

- **uids** (*list, np.ndarray*) – a list or array of the ids of people in the graph
- **average_degree** (*float*) – the average degree in the generated graph

Returns Fast implementation of the Erdos-Renyi random graph model.

Return type nx.Graph

get_expected_density(*average_degree*, *n_nodes*)

Calculate the expected density of an undirected graph with no self-loops given graph properties. The expected density of an undirected graph with no self-loops is defined as the number of edges as a fraction of the number of maximal edges possible.

Reference: Newman, M. E. J. (2010). Networks: An Introduction (pp 134-135). Oxford University Press.

Parameters

- **average_degree** (*float*) – average expected degree
- **n_nodes** (*int*) – number of nodes in the graph

Returns The expected graph density.

Return type float

6.2.4 synthpops.data module

class PopulationAgeDistribution
Bases: jsonobject.api.JsonObject

Class for population age distribution with a specified number of bins.

num_bins

distribution

class SchoolSizeDistributionByType
Bases: jsonobject.api.JsonObject

Class for the school size distribution by school type.

school_type

size_distribution

class SchoolTypeByAge
Bases: jsonobject.api.JsonObject

Class for the school type by age range.

school_type

age_range

class Location
Bases: jsonobject.api.JsonObject

Class for the json object for the location containing data about the population to generate representative contact networks.

The general use case of this is to use a filepath, and the parent data is parsed from the filepath. DefaultProperty type handles either a scalar or json object. We allow a json object mainly for testing of inheriting from a parent specified directly in the json.

Most users will want to populate this with a relative or absolute file path.

Note: The structures for the population age distribution will be updated to be more flexible to take in a parameter for the number of age brackets to generate the population age distribution structure.

---

location_name

data_provenance_notices

reference_links

citations

notes

parent

population_age_distributions

employment_rates_by_age

enrollment_rates_by_age

household_head_age_brackets

household_head_age_distribution_by_family_size

household_size_distribution

ltcf_resident_to_staff_ratio_distribution

ltcf_num_residents_distribution

ltcf_num_staff_distribution

ltcf_use_rate_distribution

school_size_brackets

school_size_distribution

school_size_distribution_by_type

school_types_by_age

workplace_size_counts_by_num_personnel

get_list_properties()
Get the properties of the location data object as a list.

Returns A list of the properties of the location json object with data about the location.

Return type list

get_population_age_distribution(*nbrackets*)
Get the age distribution of the population aggregated to nbrackets age brackets. If the data doesn't contain a distribution with the requested number of brackets, an exception is raised.

Parameters **nbrackets** (*int*) – the number of age brackets the age distribution is aggregated to

Returns A list of the probability age distribution values indexed by the bracket number.

Return type list

populate_parent_data_from_file_path(*location*, *parent_file_path*)
Loading a location json object with necessary data fields filled from the parent location using the parent location file path.

Parameters

- **location** (*json*) – json object for the location data
- **parent_file_path** (*str*) – file path to the parent location

Returns The location json object with necessary data fields filled from the parent location.

Return type json

populate_parent_data_from_json_obj(*location*, *parent*)
Loading a location json object with necessary data fields filled from the parent location json.

Parameters

- **location** (*json*) – json object for the location data
- **parent** (*json*) – json object for the parent location

Returns The location json object with necessary data fields filled from the parent location.

Return type json

populate_parent_data(*location*)
Populate location json object with fields from the parent location if available.

Parameters **location** (*json*) – json data object for the location # parameter name change for more specificity

Returns The location json data object with data fields filled from the parent location.

Return type json

load_location_from_json(*json_obj*, *check_constraints=None*)
Load location data from json object with some checks made.

Parameters **json_obj** (*json*) – json object containing location data

Returns The json object with location data.

Return type json

load_location_from_json_str(*json_str*, *check_constraints=None*)
Load location data from json str with some checks made.

Parameters **json_str** (*str*) – string version of the json object

Returns The json object with location data.

Return type json

get_relative_path(*datadir*)
Get the relative path for the data folder.

Parameters **datadir** (*str*) – data folder path

Returns Relative path for the data folder.

Return type str

Notes

This method may not be necessary anymore...

get_location_attr(*location*, *property_name*)
Get the attribute from the json object containing location data given the associated property name.

Parameters

- **location** (*json*) – the json object with location data
- **property_name** (*str*) – the property name

Returns If property_name exists in the location json object, return [True, attribute]. Else, return [False, None].

load_location_from_filepath(*rel_filepath*, *check_constraints=None*)
Loads location data object from provided relative filepath where the file path is relative to defaults.settings.datadir.

Parameters **rel_filepath** (*str*) – relative file path for the location data

Returns The json object with location data.

Return type json

save_location_to_filepath(*location*, *abs_filepath*)
Saves json object with location data to provided absolute filepath.

Parameters

- **location** (*json*) – the json object with location data

- **abs_filepath** (*str*) – absolute file path to where the json is saved

Returns None.

check_location_constraints_satisfied(*location*)
Checks a number of constraints that need to be satisfied for the schema.

Parameters **location** (*json*) – the json object with location data

Returns None.

Raises

- RuntimeError with a description if one of the constraints is not
- satisfied.

are_location_constraints_satisfied(*location*)
Checks a number of constraints that need to be satisfied for the schema.

Parameters **location** (*json*) – the json object with location data

Returns [True, None] if all constraints are satisfied. [False, str] if a constraint is violated. The returned str is one of the error messages.

check_array_of_array_entry_lens_arr(*array_of_arrays*, *expected_len*)

check_array_of_arrays_entry_lens(*location*, *expected_len*, *property_name*)
Check that each array in an array of arrays has the expected length.

Parameters

- **location** (*json*) – the json object with location data
- **expected_len** (*int*) – the expected length of each sub array
- **property_name** (*str*) – the property name

Returns [True, None] if sub array length checks pass. [False, str] if sub array length checks fail. The returned str is the error message.

check_valid_probability_distributions(*property_name*, *valid_properties=None*)
Check that the property_name is a valid probability distribution.

Parameters

- **property_name** (*str*) – the property name
- **valid_properties** (*str or list*) – a list of the valid probability distributions

Returns None.

check_probability_distribution_sum_age_distributions(*location*, *arr*, *tolerance=0.01*, ***kwargs*)
Check that each population age distribution has a sum equal to 1 within some tolerance.

Parameters

- **location** (*json*) – the json object with location data
- **arr** (*list*) – the list of population age distributions
- **tolerance** (*float*) – difference from the sum of 1 tolerated
- **kwargs** (*dict*) – dictionary of values passed to np.isclose()

Returns [True, None] if the sum of the probability distribution is equal to 1 within the tolerance level. [False, str] else. The returned str is the error message with some information about the check.

check_probability_distribution_nonnegative_age_distributions(*location*, *arr*)
Check that each population age distribution has all non negative values.

Parameters

- **location** (*json*) – the json object with location data
- **arr** (*list*) – the list of population age distributions

Returns [True, None] if the sum of the probability distribution is equal to 1 within the tolerance level. [False, str] else. The returned str is the error message with some information about the check.

check_probability_distribution_sum(*location*, *property_name*, *tolerance=0.01*, *valid_properties=None*, ***kwargs*)
Check that fields representing probability distributions have sums equal to 1 within some tolerance.

Parameters

- **location** (*json*) – the json object with location data
- **property_name** (*str*) – the property name
- **tolerance** (*float*) – difference from the sum of 1 tolerated
- **valid_properties** (*str or list*) – a list of the valid probability distributions
- **kwargs** (*dict*) – dictionary of values passed to np.isclose()

Returns [True, None] if the sum of the probability distribution is equal to 1 within the tolerance level. [False, str] else. The returned str is the error message with some information about the check.

check_probability_distribution_nonnegative(*location*, *property_name*, *valid_properties=None*)
Check that fields representing probability distributions have all non negative values.

Parameters

- **location** (*json*) – the json object with location data
- **property_name** (*str*) – the property name
- **valid_properties** (*str or list*) – a list of the valid probability distributions

Returns [True, None] if the values of the probability distribution are all non negative. [False, str] else. The returned str is the error message with some information about the check.

check_all_probability_distribution_sums(*location*, *tolerance=0.01*, *die=False*, *verbose=False*, ***kwargs*)
Checks that each probability distribution available to a location has a sum close to 1.

Parameters

- **location** (*json*) – the json object with location data
- **tolerance** (*float*) – difference from the sum of 1 tolerated
- **die** (*bool*) – raise an exception if the check fails
- **verbose** (*bool*) – print a warning if the check fails
- **kwargs** (*dict*) – dictionary of values passed to np.isclose()

Returns List of checks and a list of associated error messages.

Return type list, list

check_all_probability_distribution_nonnegative(*location*, *die=False*, *verbose=True*)
Run checks that a field representing probabilty distributions has all non negative values.

Parameters

- **location** (*json*) – json object with the location data
- **die** (*bool*) – raise an exception if the check fails
- **verbose** (*bool*) – print a warning if the check fails

Returns List of checks and a list of associated error messages.

Return type list, list

check_location_name(*location*)
Check the location json data object has a string.

Parameters **location** (*json*) – the json object with location data

Returns [True, str] if the location json has a str value in the location_name field. Returned str specifies the location_name. [False, str] if the location json does not have a str value in the location_name field.

check_population_age_distributions(*location*)
Check that the population age distributions are self-consistent in the number of brackets, and each sub array has length 3.

Parameters **location** (*json*) – the json object with location data

Returns [True, None] if checks pass. [False, str] if checks fail.

check_employment_rates_by_age(*location*)
Check that the employment rates by age is an array of arrays, where each sub array has length 2.

Parameters **location** (*json*) – the json object with location data

Returns [True, None] if checks pass. [False, str] if checks fail.

check_enrollment_rates_by_age(*location*)
Check that the enrollment rates by age is an array of arrays, where each sub array has length 2.

Parameters **location** (*json*) – the json object with location data

Returns [True, None] if checks pass. [False, str] if checks fail.

check_household_head_age_brackets(*location*)
Check that the household head age brackets is an array of arrays, where each sub array has length 2.

Parameters **location** (*json*) – the json object with location data

Returns [True, None] if checks pass. [False, str] if checks fail.

check_household_head_age_distributions_by_family_size(*location*)
Check that the conditional household head age distribution by household size is an array with length equal to the number of household head age brackets.

Parameters **location** (*json*) – the json object with location data

Returns [True, None] if checks pass. [False, str] if checks fail.

check_household_size_distribution(*location*)
Check that the household size distribution is an array of arrays, where each sub array has length 2.

Parameters **location** (*json*) – the json object location data

Returns [True, None] if checks pass. [False, str] if checks fail.

check_ltcf_resident_to_staff_ratio_distribution(*location*)
Check that the long term care facility resident to staff ratio distribution is an array of arrays, where each sub array has length 3.

Parameters **location** (*json*) – the json object location data

Returns [True, None] if checks pass. [False, str] if checks fail.

check_ltcf_num_residents_distribution(*location*)
Check that the long term care facility resident size distribution is an array of arrays, where each sub array has length 3.

Parameters **location** (*json*) – the json object location data

Returns [True, None] if checks pass. [False, str] if checks fail.

check_ltcf_num_staff_distribution(*location*)
Check that the long term care facility staff size distribution is an array of arrays, where each sub array has length 3.

Parameters **location** (*json*) – the json object location data

Returns [True, None] if checks pass. [False, str] if checks fail.

check_school_size_brackets(*location*)
Check that the school size distribution brackets is an array of arrays, where each sub array has length 2.

Parameters **location** (*json*) – the json object location data

Returns [True, None] if checks pass. [False, str] if checks fail.

check_school_size_distribution(*location*)

check_school_size_distribution_by_type(*location*)
Check that the school size distribution by school type is an array of arrays, where each sub array has length 3.

Parameters **location** (*json*) – the json object location data

Returns [True, None] if checks pass. [False, str] if checks fail.

check_school_types_by_age(*location*)
Check that the school types by age range is an array of arrays, where each sub array has length 2.

Parameters **location** (*json*) – the json object location data

Returns [True, None] if checks pass. [False, str] if checks fail.

check_workplace_size_counts_by_num_personnel(*location*)
Check that the workplace size count is an array of arrays, where each sub array has length 3.

Parameters **location** (*json*) – the json object location data

Returns [True, None] if checks pass. [False, str] if checks fail.

convert_df_to_json_array(*df*, *cols*, *int_cols=None*)
Convert desired data from a pandas dataframe into a json array.

Parameters

- **df** (*pandas dataframe*) – the dataframe with data
- **cols** (*list*) – list of the columns to convert to the json array format
- **int_cols** (*str or list*) – a str or list of columns to convert to integer values

Returns An array version of the pandas dataframe to be added to synthpops json data objects.

Return type array

6.2.5 synthpops.data_distributions module

Read in data distributions.

get_relative_path(*datadir*)
Get the path relative for the datadir.

Parameters **datadir** (*str*) – path to a specified data directory

Returns A path relative to a specified data directory datadir

Return type str

get_nbrackets()
Return the default number of age brackets.

calculate_which_nbrackets_to_use(*location_data*, *nbrackets=None*)
Calculate the number of age brackets to use by default.

Parameters **nbrackets** (*int*) – the number of age brackets to use

Returns The number of age brackets to use.

Return type int

sanitize_location(*location*)
Process and return a valid name for a location.

Parameters **location** (*str*) – name of the location

Returns A processed location name.

Return type str

calculate_location_filename(*location*, *state_location*, *country_location*)
Process a location filename.

Parameters

- **location** (*string*) – name of the location
- **state_location** (*string*) – name of the state the location is in
- **country_location** (*string*) – name of the country the location is in

Returns A filename for where the location data reside.

Return type str

calculate_location_filepath(*location*, *state_location*, *country_location*)
Process a location filepath.

Parameters

- **location** (*string*) – name of the location
- **state_location** (*string*) – name of the state the location is in
- **country_location** (*string*) – name of the country the location is in

Returns A filename for where the location data reside.

Return type str

load_location(*specific_location*, *state_location*, *country_location*, *revert_to_default=None*)
Loading json object for the location data.

Parameters

- **specific_location** (*string*) – name of the location
- **state_location** (*string*) – name of the state the location is in
- **country_location** (*string*) – name of the country the location is in
- **revert_to_default** (*bool*) – If True, try to first find location specific data to return otherwise use default data specified by the default location

Returns A filename for where the location data reside.

Return type str

read_age_bracket_distr(*datadir=None*, *location=None*, *state_location=None*, *country_location=None*, *nbrackets=None*, *file_path=None*, *use_default=False*)

A dict of the age distribution by age brackets. If use_default, then we'll first try to look for location specific data and if that's not available we'll use default data from settings.location, settings.state_location, settings.country_location. This may not be appropriate for the population under study so it's best to provide as much data as you can for the specific population.

Parameters

- **datadir** (*string*) – file path to the data directory
- **location** (*string*) – name of the location
- **state_location** (*string*) – name of the state the location is in
- **country_location** (*string*) – name of the country the location is in
- **file_path** (*string*) – file path to user specified age bracket distribution data
- **use_default** (*bool*) – if True, try to first use the other parameters to find data specific to the location under study, otherwise returns default data drawing from the settings.location, settings.state_location, settings.country_location.

Returns A dictionary of the age distribution by age bracket. Keys map to a range of ages in that age bracket.

Return type dict

get_smoothed_single_year_age_distr(*datadir=None*, *location=None*, *state_location=None*, *country_location=None*, *nbrackets=None*, *file_path=None*, *use_default=False*, *window_length=7*)

A smoothed dict of the age distribution by single years. If use_default, then we'll first try to look for location specific data and if that's not available we'll use default data from settings.location, settings.state_location, settings.country_location. This may not be appropriate for the population under study so it's best to provide as much data as you can for the specific population. Using moving windows to smooth out the age distribution.

Parameters

- **datadir** (*string*) – file path to the data directory
- **location** (*string*) – name of the location
- **state_location** (*string*) – name of the state the location is in
- **country_location** (*string*) – name of the country the location is in
- **file_path** (*string*) – file path to user specified age bracket distribution data
- **use_default** (*bool*) – If True, try to first use the other parameters to find data specific to the location under study, otherwise returns default data drawing from the settings.location, settings.state_location, settings.country_location.

- **window_length** (*int*) – length of window, in units of years, over which to average or smooth out age distribution

Returns A dictionary of the age distribution by age bracket. Keys map to a range of ages in that age bracket.

Return type dict

get_household_size_distr(*datadir=None*, *location=None*, *state_location=None*, *country_location=None*, *file_path=None*, *use_default=False*)

A dictionary of the distribution of household sizes. If you don't give the file_path, then supply the location, state_location, and country_location strings. If use_default, then we'll first try to look for location specific data and if that's not available we'll use default data from settings.location, settings.state_location, settings.country_location. This may not be appropriate for the population under study so it's best to provide as much data as you can for the specific population.

Parameters

- **datadir** (*string*) – file path to the data directory
- **location** (*string*) – name of the location
- **state_location** (*string*) – name of the state the location is in
- **country_location** (*string*) – name of the country the location is in
- **file_path** (*string*) – file path to user specified household size distribution data
- **use_default** (*bool*) – if True, try to first use the other parameters to find data specific to the location under study, otherwise returns default data drawing from settings.location, settings.state_location, settings.country_location.

Returns A dictionary of the household size distribution data. Keys map to the household size as an integer, values are the percent of households of that size.

Return type dict

get_head_age_brackets(*datadir=None*, *location=None*, *state_location=None*, *country_location=None*, *file_path=None*, *use_default=False*)

Get a dictionary of head age brackets either from the file_path directly, or using the other parameters to figure out what the file_path should be. If use_default, then we'll first try to look for location specific data and if that's not available we'll use default data from settings.location, settings.state_location, settings.country_location. This may not be appropriate for the population under study so it's best to provide as much data as you can for the specific population.

Parameters

- **datadir** (*string*) – file path to the data directory
- **location** (*string*) – name of the location
- **state_location** (*string*) – name of the state
- **country_location** (*string*) – name of the country the state_location is in
- **file_path** (*string*) – file path to user specified head age brackets data
- **use_default** (*bool*) – if True, try to first use the other parameters to find data specific to the location under study, otherwise returns default data drawing from the settings.location, settings.state_location, settings.country_location.

Returns A dictionary of the age brackets for head of household distribution data. Keys map to the age bracket as an integer, values are the percent of households which head of household in that age bracket.

Return type dict

get_head_age_by_size_distr(*datadir=None*, *location=None*, *state_location=None*, *country_location=None*, *file_path=None*, *use_default=False*)

Create an array of head of household age bracket counts (column) given by size (row). If use_default, then we'll first try to look for location specific data and if that's not available we'll use default data from the settings.location, settings.state_location, settings.country_location. This may not be appropriate for the population under study so it's best to provide as much data as you can for the specific population.

Parameters

- **datadir** (*string*) – file path to the data directory
- **location** (*string*) – name of the location
- **state_location** (*string*) – name of the state
- **country_location** (*string*) – name of the country the state_location is in
- **file_path** (*string*) – file path to user specified age of the head of the household by household size distribution data
- **use_default** (*bool*) – if True, try to first use the other parameters to find data specific to the location under study, otherwise returns default data drawing from settings.location, settings.state_location, settings.country_location.

Returns An array where each row s represents the age distribution of the head of households for households of size s-1.

Return type ndarray

get_census_age_brackets(*datadir=None*, *location=None*, *state_location=None*, *country_location=None*, *file_path=None*, *use_default=False*, *nbrackets=None*)

Get census age brackets: depends on the country or source of the age distribution and the contact pattern data. If use_default, then we'll first try to look for location specific data and if that's not available we'll use default data from settings.location, settings.state_location, settings.country_location. This may not be appropriate for the population under study so it's best to provide as much data as you can for the specific population.

Parameters

- **datadir** (*string*) – file path to the data directory
- **location** (*string*) – name of the location
- **state_location** (*string*) – name of the state
- **country_location** (*string*) – name of the country the state_location is in
- **file_path** (*string*) – file path to user specified census age brackets
- **use_default** (*bool*) – if True, try to first use the other parameters to find data specific to the location under study, otherwise returns default data drawing from settings.location, settings.state_location, settings.country_location.

Returns A dictionary of the range of ages that map to each age bracket.

Return type dict

get_contact_matrix(*datadir*, *setting_code*, *sheet_name=None*, *file_path=None*, *delimiter=' '*, *header=None*)

Get setting specific age contact matrix given sheet name to use. If file_path is given, then delimiter and header should also be specified.

Parameters

- **datadir** (*string*) – file path to the data directory

- **setting_code** (*string*) – name of the physial contact setting: H for households, S for schools, W for workplaces, C for community or other
- **sheet_name** (*string*) – name of the sheet in the excel file with contact patterns
- **file_path** (*string*) – file path to user specified age contact matrix
- **delimiter** (*string*) – delimter for the contact matrix file
- **header** (*int*) – row number for the header of the file

Returns Matrix of contact patterns where each row i is the average contact patterns for an individual in age bracket i and the columns represent the age brackets of their contacts. The matrix element i,j is then the contact rate, number, or frequency for the average individual in age bracket i with all of their contacts in age bracket j in that physical contact setting.

Return type ndarray

get_contact_matrices(*datadir=None*, *sheet_name=None*, *file_path_dic=None*, *delimiter=' '*, *header=None*, *use_default=False*)

Create a dict of setting specific age contact matrices. If use_default, then we'll first try to look for location specific data and if that's not available we'll use default data from settings.sheet_name. This may not be appropriate for the population under study so it's best to provide as much data as you can for the specific population.

Parameters

- **datadir** (*string*) – file path to the data directory
- **setting_code** (*string*) – name of the physial contact setting: H for households, S for schools, W for workplaces, C for community or other
- **sheet_name** (*string*) – name of the sheet in the excel file with contact patterns
- **file_path_dic** (*string*) – dictionary to file paths of user specified age contact matrix, where keys are "H", "S", "W", and "C".
- **delimiter** (*string*) – delimter for the contact matrix file
- **header** (*int*) – row number for the header of the file

Returns A dictionary of the different contact matrices for each population, given by the sheet name. Keys map to the different possible physical contact settings for which data are available.

Return type dict

get_school_enrollment_rates(*datadir=None*, *location=None*, *state_location=None*, *country_location=None*, *file_path=None*, *use_default=False*)

Get dictionary of enrollment rates by age. If use_default, then we'll first try to look for location specific data and if that's not available we'll use default data from settings.location, settings.state_location, settings.country_location. This may not be appropriate for the population under study so it's best to provide as much data as you can for the specific population.

Parameters

- **datadir** (*string*) – file path to the data directory
- **location** (*string*) – name of the location
- **state_location** (*string*) – name of the state the location is in
- **country_location** (*string*) – name of the country the location is in
- **file_path** (*string*) – file path to user specified school enrollment by age data

- **use_default** (*bool*) – if True, try to first use the other parameters to find data specific to the location under study, otherwise returns default data drawing from settings.location, settings.state_location, settings.country_location.

Returns A dictionary of school enrollment rates by age.

Return type dict

get_school_size_brackets(*datadir=None*, *location=None*, *state_location=None*, *country_location=None*, *file_path=None*, *use_default=False*)

Get school size brackets: depends on the source/location of the data. If use_default, then we'll first try to look for location specific data and if that's not available we'll use default data from settings.location, settings.state_location, settings.country_location. This may not be appropriate for the population under study so it's best to provide as much data as you can for the specific population.

Parameters

- **datadir** (*string*) – file path to the data directory
- **location** (*string*) – name of the location
- **state_location** (*string*) – name of the state the location is in
- **country_location** (*string*) – name of the country the location is in
- **file_path** (*string*) – file path to user specified school size brackets data
- **use_default** (*bool*) – if True, try to first use the other parameters to find data specific to the location under study, otherwise returns default data drawing from settings.location, settings.state_location, settings.country_location.

Returns A dictionary of school size brackets.

Return type dict

get_school_size_distr_by_brackets(*datadir=None*, *location=None*, *state_location=None*, *country_location=None*, *file_path=None*, *use_default=False*)

Get distribution of school sizes by size bracket or bin. If use_default, then we'll first try to look for location specific data and if that's not available we'll use default data from settings.location, settings.state_location, settings.country_location. This may not be appropriate for the population under study so it's best to provide as much data as you can for the specific population.

Parameters

- **datadir** (*string*) – file path to the data directory
- **location** (*string*) – name of the location
- **state_location** (*string*) – name of the state the location is in
- **country_location** (*string*) – name of the country the location is in
- **file_path** (*string*) – file path to user specified school size distribution data
- **use_default** (*bool*) – if True, try to first use the other parameters to find data specific to the location under study, otherwise returns default data drawing from settings.location, settings.state_location, settings.country_location.

Returns A dictionary of the distribution of school sizes by bracket.

Return type dict

get_default_school_type_age_ranges()

Define and return default school types and the age range for each.

Returns A dictionary of default school types and the age range for each.

Return type dict

get_default_school_types_distr_by_age()
Define and return default probabilities of school type for each age.

Returns A dictionary of default probabilities for the school type likely for each age.

Return type dict

get_default_school_types_by_age_single()
Define and return default school type by age by assigning the school type with the highest probability.

Returns A dictionary of default school type by age.

Return type dict

get_default_school_size_distr_brackets()
Define and return default school size distribution brackets.

Returns A dictionary of school size brackets.

Return type dict

get_default_school_size_distr_by_type()
Define and return default school size distribution for each school type. The school size distributions are binned to size groups or brackets.

Returns A dictionary of school size distributions binned by size groups or brackets for each type of default school.

Return type dict

get_school_type_age_ranges(*datadir=None*, *location=None*, *state_location=None*, *country_location=None*, *file_path=None*, *use_default=False*)
Get a dictionary of the school types and the age range for each for the location specified.

Parameters

- **datadir** (*string*) – file path to the data directory
- **location** (*string*) – name of the location
- **state_location** (*string*) – name of the state the location is in
- **country_location** (*string*) – name of the country the location is in
- **file_path** (*string*) – file path to user specified distribution data
- **use_default** (*bool*) – if True, try to first use the other parameters to find data specific to the location under study, otherwise returns default data drawing from Seattle, Washington.

Returns A dictionary of default school types and the age range for each.

Return type dict

get_school_size_distr_by_type(*datadir=None*, *location=None*, *state_location=None*, *country_location=None*, *file_path=None*, *use_default=False*)
Get the school size distribution by school types. If use_default, then we'll try to look for location specific data first, and if that's not available we'll use default data from the set default locations (see sp.defaults.py). This may not be appropriate for the population under study so it's best to provide as much data as you can for the specific population.

Parameters

- **datadir** (*string*) – file path to the data directory
- **location** (*string*) – name of the location

- **state_location** (*string*) – name of the state the location is in
- **country_location** (*string*) – name of the country the location is in
- **file_path** (*string*) – file path to user specified distribution data
- **use_default** (*bool*) – if True, try to first use the other parameters to find data specific to the location under study, otherwise returns default data drawing from settings.location, settings.state_location, settings.country_location

Returns A dictionary of school size distributions binned by size groups or brackets for each type of default school.

Return type dict

get_employment_rates(*datadir=None*, *location=None*, *state_location=None*, *country_location=None*, *file_path=None*, *use_default=False*)

Get employment rates by age. If use_default, then we'll first try to look for location specific data and if that's not available we'll use default data from settings.location, settings.state_location, settings.country_location. This may not be appropriate for the population under study so it's best to provide as much data as you can for the specific population.

Parameters

- **datadir** (*string*) – file path to the data directory
- **location** (*string*) – name of the location
- **state_location** (*string*) – name of the state the location is in
- **country_location** (*string*) – name of the country the location is in, which should be the 'usa'
- **file_path** (*string*) – file path to user specified employment by age data
- **use_default** (*bool*) – if True, try to first use the other parameters to find data specific to the location under study, otherwise returns default data drawing from settings.location, settings.state_location, settings.country_location.

Returns A dictionary of employment rates by age.

Return type dict

get_workplace_size_brackets(*datadir=None*, *location=None*, *state_location=None*, *country_location=None*, *file_path=None*, *use_default=False*)

Get workplace size brackets. If use_default, then we'll first try to look for location specific data and if that's not available we'll use default data from settings.location, settings.state_location, settings.country_location. This may not be appropriate for the population under study so it's best to provide as much data as you can for the specific population.

Parameters

- **datadir** (*string*) – file path to the data directory
- **location** (*string*) – name of the location
- **state_location** (*string*) – name of the state the location is in
- **country_location** (*string*) – name of the country the location is in, which should be the 'usa'
- **file_path** (*string*) – file path to user specified workplace size brackets data
- **use_default** (*bool*) – if True, try to first use the other parameters to find data specific to the location under study, otherwise returns default data drawing from settings.location, settings.state_location, settings.country_location.

Returns A dictionary of workplace size brackets.

Return type dict

get_workplace_size_distr_by_brackets(*datadir=None*, *location=None*, *state_location=None*, *country_location=None*, *file_path=None*, *use_default=False*)

Get the distribution of workplace size by brackets. If use_default, then we'll first try to look for location specific data and if that's not available we'll use default data from settings.location, settings.state_location, settings.country_location. This may not be appropriate for the population under study so it's best to provide as much data as you can for the specific population.

Parameters

- **datadir** (*string*) – file path to the data directory
- **location** (*string*) – name of the location
- **state_location** (*string*) – name of the state the location is in
- **country_location** (*string*) – name of the country the location is in
- **file_path** (*string*) – file path to user specified workplace size distribution data
- **use_default** (*bool*) – if True, try to first use the other parameters to find data specific to the location under study, otherwise returns default data drawing from settings.location, settings.state_location, settings.country_location.

Returns A dictionary of the distribution of workplace sizes by bracket.

Return type dict

get_state_postal_code(*state_location*, *country_location*)

Get the state postal code.

Parameters

- **state_location** (*string*) – name of the state
- **country_location** (*string*) – name of the country the state is in

Returns A postal code for the state_location.

Return type str

get_long_term_care_facility_residents_distr(*datadir=None*, *location=None*, *state_location=None*, *country_location=None*, *file_path=None*, *use_default=False*)

Get size distribution of residents per facility for Long Term Care Facilities.

Parameters

- **datadir** (*string*) – file path to the data directory
- **location** (*string*) – name of the location
- **state_location** (*string*) – name of the state the location is in
- **country_location** (*string*) – name of the country the location is in
- **file_path** (*string*) – file path to user specified LTCF resident size distribution data
- **use_default** (*bool*) – if True, try to first use the other parameters to find data specific to the location under study, otherwise returns default data drawing from settings.location, settings.state_location, settings.country_location.

Returns A dictionary of the distribution of residents per facility for Long Term Care Facilities.

Return type dict

get_long_term_care_facility_residents_distr_brackets(*datadir=None*, *location=None*, *state_location=None*, *country_location=None*, *file_path=None*, *use_default=False*)

Get size bins for the distribution of residents per facility for Long Term Care Facilities.

Parameters

- **datadir** (*string*) – file path to the data directory
- **location** (*string*) – name of the location
- **state_location** (*string*) – name of the state the location is in
- **country_location** (*string*) – name of the country the location is in, which should be the 'usa'
- **file_path** (*string*) – file path to user specified LTCF resident size brackets data
- **use_default** (*bool*) – if True, try to first use the other parameters to find data specific to the location under study, otherwise returns default data drawing from settings.location, settings.state_location, settings.country_location.

Returns A dictionary of size brackets or bins for residents per facility.

Return type dict

get_long_term_care_facility_resident_to_staff_ratios_distr(*datadir=None*, *location=None*, *state_location=None*, *country_location=None*, *file_path=None*, *use_default=False*)

Get size distribution of resident to staff ratios per facility for Long Term Care Facilities.

Parameters

- **datadir** (*string*) – file path to the data directory
- **location** (*string*) – name of the location
- **state_location** (*string*) – name of the state the location is in
- **country_location** (*string*) – name of the country the location is in
- **file_path** (*string*) – file path to user specified resident to staff ratio distribution data
- **use_default** (*bool*) – if True, try to first use the other parameters to find data specific to the location under study, otherwise returns default data drawing from settings.location, settings.state_location, settings.country_location.

Returns A dictionary of the distribution of residents per facility for Long Term Care Facilities.

Return type dict

get_long_term_care_facility_resident_to_staff_ratios_brackets(*datadir=None*, *location=None*, *state_location=None*, *country_location=None*, *file_path=None*, *use_default=False*)

Get size bins for the distribution of resident to staff ratios per facility for Long Term Care Facilities.

Parameters

- **datadir** (*string*) – file path to the data directory
- **location** (*string*) – name of the location
- **state_location** (*string*) – name of the state the location is in
- **country_location** (*string*) – name of the country the location is in, which should be the 'usa'
- **file_path** (*string*) – file path to user specified resident to staff ratio brackets data
- **use_default** (*bool*) – if True, try to first use the other parameters to find data specific to the location under study, otherwise returns default data drawing from settings.location, settings.state_location, settings.country_location.

Returns A dictionary of size brackets or bins for resident to staff ratios per facility.

Return type dict

get_long_term_care_facility_use_rates(*datadir=None*, *location=None*, *state_location=None*, *country_location=None*, *file_path=None*, *use_default=False*)

Get Long Term Care Facility use rates by age for a state.

Parameters

- **datadir** (*str*) – file path to the data directory
- **location_alias** (*str*) – more commonly known name of the location
- **state_location** (*str*) – name of the state the location is in
- **country_location** (*str*) – name of the country the location is in
- **file_path** (*string*) – file path to user specified gender by age bracket distribution data
- **use_default** (*bool*) – if True, try to first use the other parameters to find data specific to the location under study, otherwise returns default data drawing from settings.location, settings.state_location, settings.country_location.

Returns A dictionary of the Long Term Care Facility usage rates by age.

Return type dict

Note: Currently only available for the United States.

6.2.6 synthpops.defaults module

Defaults for synthpops files and data types.

default_datadir_path()
Return the path to synthpops internal data folder.

reset_settings_by_key(*key*, *value*)
Reset a key in the globally available settings dictionary with a new value.

Returns None

reset_settings(*new_config*)
Reset multiple keys in the globally available settings dictionary based on a new dictionary of values.

Parameters **new_config** (*dict*) – a dictionary with new values mapped to keys

Returns None.

reset_default_settings()
By default, synthpops uses default settings for Seattle, Washington, USA. After having changed the values in the settings dictionary, calling this method can easily reset the settings dictionary to the values for Seattle, Washington, USA.

Returns None

6.2.7 synthpops.households module

Functions for generating households

class Household(*hhid=None*, *reference_uid=None*, *reference_age=None*, ***kwargs*)
Bases: *synthpops.base.LayerGroup*

A class for individual households and methods to operate on each.

Parameters **kwargs** (*dict*) – data dictionary of the household

Class constructor for empty household.

Parameters

- ****hhid** (*int*) – household id
- ****member_uids** (*np.array*) – ids of household members
- ****reference_uid** (*int*) – id of the reference person
- ****reference_age** (*int*) – age of the reference person

validate()
Check that information supplied to make a household is valid and update to the correct type if necessary.

get_household(*pop*, *hhid*)
Return household with id: hhid.

Parameters

- **pop** (*sp.Pop*) – population
- **hhid** (*int*) – household id number

Returns A populated household.

Return type sp.Household

add_household(*pop*, *household*)
Add a household to the list of households.

Parameters

- **pop** (*sp.Pop*) – population
- **household** (*sp.Household*) – household with at minimum the hhid, member_uids, member_ages, reference_uid, and reference_age.

initialize_empty_households(*pop*, *n_households=None*)
Array of empty households.

Parameters

- **pop** (*sp.Pop*) – population
- **n_households** (*int*) – the number of households to initialize

populate_households(*pop*, *households*, *age_by_uid*)
Populate all of the households. Store each household at the index corresponding to it's hhid.

Parameters

- **pop** (*sp.Pop*) – population
- **households** (*list*) – list of lists where each sublist represents a household and contains the ids of the household members
- **age_by_uid** (*dict*) – dictionary mapping each person's id to their age

generate_household_size_count_from_fixed_pop_size(*N*, *hh_size_distr*)
Given a number of people and a household size distribution, generate the number of homes of each size needed to place everyone in a household.

Parameters

- **N** (*int*) – The number of people in the population.
- **hh_size_distr** (*dict*) – The distribution of household sizes.

Returns An array with the count of households of size s at index s-1.

assign_uids_by_homes(*homes*, *id_len=16*, *use_int=True*)
Assign IDs to everyone in order by their households.

Parameters

- **homes** (*array*) – The generated synthetic ages of household members.
- **id_len** (*int*) – The length of the UID.
- **use_int** (*bool*) – If True, use ints for the uids of individuals; otherwise use strings of length 'id_len'.

Returns A copy of the generated households with IDs in place of ages, and a dictionary mapping ID to age.

generate_age_count(*n*, *age_distr*)
Generate a stochastic count of people for each age given the age distribution (age_distr) and number of people to generate (n).

Parameters

- **n** (*int*) – number of people to generate
- **age_distr** (*list or np.ndarray*) – single year age distribution

Returns A dictionary with the count of people to generate for each age given an age distribution and the number of people to generate.

Return type dict

generate_age_count_multinomial(*n*, *age_distr*)

Generate a stochastic count of people for each age given the age distribution (age_distr) and number of people to generate (n).

Parameters

- **n** (*int*) – number of people to generate
- **age_distr** (*list or np.ndarray*) – single year age distribution

Returns A dictionary with the count of people to generate for each age given an age distribution and the number of people to generate.

Return type dict

generate_household_head_ages(*household_sizes*, *hha_by_size*, *hha_brackets*, *ages_left_to_assign*)

Generate the head of household ages conditional on household size and the expected ages of people in the population.

Parameters

- **household_sizes** (*np.array*) – Array of household sizes to be generated
- **hha_by_size** (*matrix*) – A matrix in which each row contains the age distribution of the reference person for household size s at index s-1.
- **hha_brackets** (*dict*) – The age brackets for the heads of household.
- **ages_left_to_assign** (*dic*) – The counter of ages for the generated population left to place in a residence

Returns An array of head of household ages, updated counter of the ages in the population left to place in a residence.

generate_household_sizes(*hh_sizes*)

Create a list of the household sizes in random order so that as individuals are placed by age into homes running out of specific ages is not systemically an issue for any given household size unless certain sizes greatly outnumber households of other sizes.

Parameters **hh_sizes** (*array*) – The count of household size s at index s-1.

Returns An array of household sizes to be generated and place people into households.

Return type Np.array

generate_larger_households_fixed_ages(*larger_hh_size_array*, *larger_hha_chosen*, *hha_brackets*, *cm_age_brackets*, *cm_age_by_brackets*, *household_matrix*, *ages_left_to_assign*, *homes_dic*)

Assign people to households larger than one person (excluding special residences like long term care facilities or agricultural workers living in shared residential quarters).

Parameters

- **hh_sizes** (*array*) – The count of household size s at index s-1.
- **hha_by_size** (*matrix*) – A matrix in which each row contains the age distribution of the reference person for household size s at index s-1.
- **hha_brackets** (*dict*) – The age brackets for the heads of household.
- **cm_age_brackets** (*dict*) – The age brackets for the contact matrix.
- **cm_age_by_brackets** (*dict*) – A dictionary mapping age to the age bracket range it falls within.

- **household_matrix** (*dict*) – The age-specific contact matrix for the household ontact setting.
- **ages_left_to_assign** (*dict*) – Age count of people left to place in households larger than one person.

Returns A dictionary of households by age indexed by household size.

Return type dict

generate_all_households_fixed_ages(*n_remaining*, *hh_sizes*, *hha_by_size*, *hha_brackets*, *cm_age_brackets*, *cm_age_by_brackets*, *contact_matrices*, *ages_left_to_assign*)

Generate the ages of those living in households together. First create households of people living alone, then larger households. For households larger than 1, a reference individual's age is sampled conditional on the household size, while all other household members have their ages sampled conditional on the reference person's age and the age mixing contact matrix in households for the population under study. Fix the count of ages in the population before placing individuals in households so that the age distribution of the generated population is fixed to closely match the age distribution from data on the population.

Parameters

- **n_remaining** (*int*) – The number of people in the population left to place in a residence.
- **hh_sizes** (*array*) – The count of household size s at index s-1.
- **hha_by_size_counts** (*matrix*) – A matrix in which each row contains the age distribution of the reference person for household size s at index s-1.
- **hha_brackets** (*dict*) – The age brackets for the heads of household.
- **cm_age_brackets** (*dict*) – The dictionary mapping age bracket keys to age bracket range matching the household contact matrix.
- **cm_age_by_brackets** (*dict*) – The dictionary mapping age to the age bracket range it falls within matching the household contact matrix.
- **contact_matrices** (*dict*) – The dictionary of the age-specific contact matrix for different physical contact settings.
- **ages_left_to_assign** (*dict*) – Age count of people left to place in households larger than one person.

Returns An array of all households where each household is a row and the values in the row are the ages of the household members. The first age in the row is the age of the reference individual. Households are randomly shuffled by size.

generate_larger_households_infer_ages(*size*, *larger_household_sizes*, *heads_of_larger_households*, *hha_brackets*, *cm_age_brackets*, *cm_age_by_brackets*, *household_matrix*, *adjusted_age_dist*, *p=0.15*)

Generate ages of those living in households of greater than one individual. Reference individual is sampled conditional on the household size. All other household members have their ages sampled conditional on the reference person's age and the age mixing contact matrix in households for the population under study.

Parameters

- **size** (*int*) – The household size.
- **hh_sizes** (*array*) – The count of household size s at index s-1.
- **hha_by_size_counts** (*matrix*) – A matrix in which each row contains the age distribution of the reference person for household size s at index s-1.

- **hha_brackets** (*dict*) – The age brackets for the heads of household.
- **cm_age_brackets** (*dict*) – The dictionary mapping age bracket keys to age bracket range matching the household contact matrix.
- **cm_age_by_brackets** (*dict*) – The dictionary mapping age to the age bracket range it falls within matching the household contact matrix.
- **household_matrix** (*dict*) – Age-specific contact matrix for contacts in the household setting.
- **single_year_age_distr** (*dict*) – The age distribution.

Returns An array of households for size `size` where each household is a row and the values in the row are the ages of the household members. The first age in the row is the age of the reference individual.

generate_all_households_infer_ages(*n*, *n_remaining*, *hh_sizes*, *hha_by_size*, *hha_brackets*, *cm_age_brackets*, *cm_age_by_brackets*, *contact_matrices*, *adjusted_age_dist*, *ages_left_to_assign*)

Generate the ages of those living in households together. First create households of people living alone, then larger households. For households larger than 1, a reference individual's age is sampled conditional on the household size, while all other household members have their ages sampled conditional on the reference person's age and the age mixing contact matrix in households for the population under study.

Parameters

- **n** (*int*) – The number of people in the population.
- **n_remaining** (*int*) – The number of people in the population left to place in a residence.
- **hh_sizes** (*array*) – The count of household size s at index s-1.
- **hha_by_size_counts** (*matrix*) – A matrix in which each row contains the age distribution of the reference person for household size s at index s-1.
- **hha_brackets** (*dict*) – The age brackets for the heads of household.
- **cm_age_brackets** (*dict*) – The dictionary mapping age bracket keys to age bracket range matching the household contact matrix.
- **cm_age_by_brackets** (*dict*) – The dictionary mapping age to the age bracket range it falls within matching the household contact matrix.
- **contact_matrices** (*dict*) – The dictionary of the age-specific contact matrix for different physical contact settings.
- **ages_left_to_assign** (*dict*) – Age count of people left to place in households larger than one person.

Returns An array of all households where each household is a row and the values in the row are the ages of the household members. The first age in the row is the age of the reference individual. Households are randomly shuffled by size.

Note: This method is not guaranteed to model the population age distribution well automatically. The method called inside, generate_larger_households_infer_ages uses the method ltcf_resample_age to fit Seattle, Washington populations with long term care facilities generated. For a method that matches the age distribution well for populations in general, please use generate_all_households_fixed_ages.

The following contains an example of how you may resample from an age range that is over produced and instead sample ages from an age range that is under produced in your population. This kind of customization may be necessary when your age mixing matrix and the population you are interested in modeling differ in important but subtle ways. For example, generally household age mixing matrices reflect mixing patterns for

households composed of families. This means household age mixing matrices do not generally cover college or university aged individuals living together. Without this customization, this algorithm tends to under produce young adults. This method also has a tendency to underproduce the elderly, and does not explicitly model the elderly living in nursing homes. Customizations like this should be considered in context of the specific population and culture you are trying to model. In some cultures, it is common to live in non-family households, while in others family households are the most common and include multi-generational family households. If you are unsure of how to proceed with customizations please take a look at the references listed in the overview documentation for more information.

---

get_all_households(*homes_dic*)
Get all households in a list, randomly assorted.

Parameters **homes_dic** (*dict*) – A dictionary of households by age indexed by household size

Returns A random ordering of households with the ages of the individuals.

Return type list

get_household_sizes(*popdict*)
Get household sizes for each household in the popdict.

Parameters **popdict** (*dict*) – population dictionary

Returns Dictionary of the generated household size for each household.

Return type dict

get_household_heads(*popdict*)
Get the id of the head of each household.

Parameters **popdict** (*dict*) – population dictionary

Returns Dictionary of the id of the head of the household for each household.

Return type dict

Note: In static populations the id of the head of the household is the minimum id of the household members. With vital dynamics turned on and populations growing or changing households over time, this method will need to change and the household head or reference person will need to be specified at creation and when those membership events occur.

---

get_household_head_ages_by_size(*pop*)
Calculate the count of households by size and the age of the head of the household, assuming the minimal household members id is the id of the head of the household.

Parameters **pop** (*sp.Pop*) – population object

Returns An array with rows as household size and columns as household head age brackets.

Return type np.ndarray

get_generated_household_size_distribution(*household_sizes*)
Get household size distribution.

Parameters **household_sizes** (*dict*) – size of each generated household

Returns Dictionary of the generated household size distribution.

Return type dict

6.2.8 synthpops.ltcfs module

Modeling Seattle Metro Long Term Care Facilities

generate_ltcfs(*n*, *with_facilities*, *loc_pars*, *expected_age_dist*, *ages_left_to_assign*)

Generate residents living in long term care facilities and their ages.

Parameters

- **n** (*int*) – The number of people to generate in the population
- **with_facilities** (*bool*) – If True, create long term care facilities, currently only available for locations in the US.
- **loc_pars** (*dict*) – A dictionary of location parameters
- **expected_age_dist** (*dict*) – The expected age distribution
- **ages_left_to_assign** (*dic*) – The counter of ages for the generated population left to place in a residence

assign_facility_staff(*datadir*, *location*, *state_location*, *country_location*, *ltcf_staff_age_min*, *ltcf_staff_age_max*, *facilities*, *workers_by_age_to_assign_count*, *potential_worker_uids_by_age*, *potential_worker_uids*, *facilities_by_uids*, *age_by_uid*, *use_default=False*)

Assign Long Term Care Facility staff to the generated facilities with residents.

Parameters

- **datadir** (*string*) – The file path to the data directory.
- **location** – name of the location
- **state_location** (*string*) – name of the state the location is in
- **country_location** (*string*) – name of the country the location is in
- **ltcf_staff_age_min** (*int*) – Long term care facility staff minimum age.
- **ltcf_staff_age_max** (*int*) – Long term care facility staff maximum age.
- **facilities** (*list*) – A list of lists where each sublist is a facility with the resident ages
- **workers_by_age_to_assign_count** (*dict*) – A dictionary mapping age to the count of employed individuals of that age.
- **potential_worker_uids** (*dict*) – dictionary of potential workers mapping their id to their age
- **facilities** – A list of lists where each sublist is a facility with the resident IDs
- **age_by_uid** (*dict*) – dictionary mapping id to age for all individuals in the population
- **use_default** (*bool*) – If True, try to first use the other parameters to find data specific to the location under study; otherwise, return default data drawing from default_location, default_state, default_country.

Returns A list of lists with the facility staff IDs for each facility.

Return type list

remove_ltcf_residents_from_potential_workers(*facilities_by_uids*, *potential_worker_uids*, *potential_worker_uids_by_age*, *workers_by_age_to_assign_count*, *age_by_uid*)

Remove facilities residents from potential workers

Parameters

- **facilities_by_uids** (*list*) – A list of lists, where each sublist represents a skilled nursing or long term care facility and the ids of the residents living within it
- **potential_worker_uids** (*dict*) – dictionary of potential workers mapping their id to their age
- **potential_worker_uids_by_age** (*dict*) – dictionary mapping age to the list of worker ids with that age
- **workers_by_age_to_assign_count** (*dict*) – dictionary of the count of workers left to assign by age
- **age_by_uid_dic** (*dict*) – dictionary mapping id to age for all individuals in the population

Returns Updated dictionaries for potential worker ids, lists of potential worker ids mapped to age, and the number of workers left to assign by age.

ltcf_resample_age(*exp_age_distr*, *a*)

Resampling younger ages to better match data

Parameters

- **exp_age_distr** (*dict*) – age distribution
- **age** (*int*) – age as an integer

Returns Resampled age as an integer.

Notes

This is not always necessary, but is mostly used to smooth out sharp edges in the age distribution when spsamp.resample_age() produces too many of one year and under produces the surrounding ages. For example, new borns (0 years old) may be over produced, and 1 year olds under produced, so this function can be customized to correct for that. It is currently customized to model well the age distribution for Seattle, Washington.

get_ltcf_sizes(*popdict*, *keys_to_exclude=[]*)

Get long term care facility sizes, including both residents and staff.

Parameters

- **popdict** (*dict*) – population dictionary
- **keys_to_exclude** (*list*) – possible keys to exclude for roles in long term care facilities. See notes.

Returns Dictionary of the size for each long term care facility generated.

Return type dict

Notes

keys_to_exclude is an empty list by default, but can contain the different long term care facility roles: 'ltcf_res' for residents and 'ltcf_staff' for staff. If either role is included in the parameter keys_to_exclude, then individuals with that value equal to 1 will not be counted.

class LongTermCareFacility(*ltcfid=None*, *resident_uids=array([], dtype=int64)*, *staff_uids=array([], dtype=int64)*, ***kwargs*)

Bases: *synthpops.base.LayerGroup*

A class for individual long term care facilities and methods to operate on each.

Parameters **kwargs** (*dict*) – data dictionary of the long term care facility

Class constructor for empty long term care facility (ltcf).

Parameters

- ****ltcfid** (*int*) – ltcf id
- ****resident_uids** (*np.array*) – ids of ltcf members
- ****staff_uids** (*np.array*) – ages of ltcf members

validate()
Check that information supplied to make a long term care facility is valid and update to the correct type if necessary.

member_uids
residents and staff.

Returns ltcf member ids

Return type np.ndarray

Type Return ids of all ltcf members

member_ages(*age_by_uid*)
Return ages of all ltcf members: residents and staff.

Parameters **age_by_uid** (*np.ndarray*) – mapping of age to uid

Returns ltcf member ages

Return type np.ndarray

resident_ages(*age_by_uid*)
Return ages of ltcf residents.

Parameters **age_by_uid** (*np.ndarray*) – mapping of age to uid

Returns ltcf resident ages

Return type np.ndarray

staff_ages(*age_by_uid*)
Return ages of ltcf staff.

Parameters **age_by_uid** (*np.ndarray*) – mapping of age to uid

Returns ltcf staff ages

Return type np.ndarray

get_ltcf(*pop*, *ltcfid*)
Return ltcf with id: ltcfid.

Parameters

- **pop** (*sp.Pop*) – population
- **ltcfid** (*int*) – ltcf id number

Returns A populated ltcf.

Return type sp.LongTermCareFacility

add_ltcf(*pop*, *ltcf*)
Add a ltcf to the list of ltcfs.

Parameters

- **pop** (*sp.Pop*) – population
- **ltcf** (*sp.LongTermCareFacility*) – ltcf with at minimum the ltcfid, resident_uids and staff_uids.

initialize_empty_ltcfs(*pop*, *n_ltcfs=None*)
Array of empty ltcfs.

Parameters

- **pop** (*sp.Pop*) – population
- **n_ltcfs** (*int*) – the number of ltcfs to initialize

populate_ltcfs(*pop*, *resident_lists*, *staff_lists*)
Populate all of the ltcfs. Store each ltcf at the index corresponding to it's ltcfid.

Parameters

- **pop** (*sp.Pop*) – population
- **residents_list** (*list*) – list of lists where each sublist represents a ltcf and contains the ids of the residents
- **staff_lists** (*list*) – list of lists where each sublist represents a ltcf and contains the ids of the staff

6.2.9 synthpops.plotting module

This module provides plotting methods including methods to plot the age-specific contact matrix in different contact layers.

class plotting_kwargs(**args*, ***kwargs*)
Bases: sciris.sc_odict.objdict

A class to set and operate on plotting kwargs throughout synthpops.

Parameters **kwargs** (*dict*) – dictionary of plotting parameters to be used.

Class constructor for plotting_kwargs.

initialize()
Initialize plot settings.

set_font(**args*, ***font*)
Set font styles.

default_plotting_kwargs()
Define default plotting kwargs to be used in plotting methods.

set_figure_display_size(**args*, ***kwargs*)
Update plotting kwargs with new calculated display sizes.

Parameters **kwargs** (*sc.objdict*) – new values to update with

Returns Updated kwargs and recalculating the display sizes.

set_default_pop_pars()
Check if method has some key pop parameters to call on data. If not, use defaults and warn user of their use and value.

make_title(*suffix=None*, *override=False*)
Create the title for the figure depending on the location information and if there already exists a preset title_prefix.

Parameters

- **suffix** (*str*) – title suffix
- **override** (*bool*) – If True, override the title_prefix already stored in self and create a new one.

Returns None.

restore_defaults()
Reset matplotlib defaults.

update_defaults(*method_defaults*, *kwargs*)
Update defaults with method defaults and kwargs.

axis
Dictionary of axis settings.

calculate_contact_matrix(*population*, *density_or_frequency='density'*, *layer='H'*)
Calculate the symmetric age-specific contact matrix from the connections for all people in the population. density_or_frequency sets the type of contact matrix calculated.

When density_or_frequency is set to 'frequency' each person is assumed to have a fixed amount of contact with others they are connected to in a setting so each person will split their contact amount equally among their connections. This means that if a person has links to 4 other individuals then 1/4 will be added to the matrix element matrix[age_i][age_j] for each link, where age_i is the age of the individual and age_j is the age of the contact. This follows the mass action principle such that increased density or number of people a person is in contact with leads to decreased per-link or connection contact rate.

When density_or_frequency is set to 'density' the amount of contact each person has with others scales with the number of people they are connected to. This means that a person with connections to 4 individuals has a higher total contact rate than a person with connection to 3 individuals. For this definition if a person has links to 4 other individuals then 1 will be added to the matrix element matrix[age_i][age_j] for each contact. This follows the 'pseudo'mass action principle such that the per-link or connection contact rate is constant.

Parameters

- **population** (*dict*) – A dictionary of a population with attributes.
- **density_or_frequency** (*str*) – option for the type of contact matrix calculated.
- **layer** (*str*) – name of the physial contact setting, see notes.

Returns Symmetric age specific contact matrix.

Return type np.ndarray

Note: H for households, S for schools, W for workplaces, C for community or other, and 'LTCF' for long term care facilities.

---

plot_contacts(*pop*, ***kwargs*)
Plot the age mixing matrix for a specific contact layer.

Parameters

- **pop** (*pop object*) – population, either synthpops.pop.Pop or dict
- ****layer** (*str*) – name of the physial contact layer: H for households, S for schools, W for workplaces, C for community or other
- ****aggregate_flag** (*bool*) – If True, plot the contact matrix for aggregate age brackets, else single year age contact matrix.

- ****logcolors_flag** (*bool*) – If True, plot heatmap in logscale
- ****density_or_frequency** (*str*) – If 'density', then each contact counts for 1/(group size -1) of a person's contact in a group, elif 'frequency' then count each contact. This means that more people in a group leads to higher rates of contact/exposure.
- ****state_location** (*string*) – name of the state the location is in
- ****country_location** (*string*) – name of the country the location is in
- ****cmap** (*str or Matplotlib cmap*) – colormap
- ****fontsize** (*int*) – base font size
- ****rotation** (*int*) – rotation for x axis labels
- ****title_prefix** (*str*) – optional title prefix for the figure
- ****fig** (*matplotlib.figure*) – If supplied, use this figure instead of generating one
- ****ax** (*matplotlib.axes*) – If supplied, use these axes instead of generating one
- ****do_show** (*bool*) – If True, show the plot
- ****do_save** (*bool*) – If True, save the plot to disk

Returns Matplotlib figure.

plot_array (*expected*, *fig=None*, *ax=None*, ***kwargs*)

Plot histogram on a sorted array based by names. If names not provided the order will be used. If generate data is not provided, plot only the expected values. Note this can only be used with the limitation that data that has already been binned. Figure will be saved in figdir if given or else working directory.

Parameters

- **expected** (*array*) – Array of expected values
- **fig** (*matplotlib.figure*) – Matplotlib.figure object
- **ax** (*matplotlib.axis*) – Matplotlib.axes object
- ****xvalue** (*array*) – Array of values used in X-axis, must be the same length as expected
- ****generated** (*array*) – Array of values generated using a model
- ****names** (*list or dict*) – names to display on x-axis, default is set to the indexes of data
- ****figname** (*str*) – name to save figure to disk
- ****figdir** (*str*) – directory to save the plot if provided
- ****prefix** (*str*) – used to prefix the title of the plot
- ****fontsize** (*float*) – default fontsize
- ****color_1** (*str*) – color for expected data
- ****color_2** (*str*) – color for generated data
- ****expect_label** (*str*) – Label to show in the plot, default to "expected"
- ****value_text** (*bool*) – If True, display the values on top of the bar if specified
- ****rotation** (*float*) – rotation angle for xticklabels
- ****binned** (*bool*) – If True, data are binned. Else, if False, plot a simple histogram for expected data.

- ****do_show** (*bool*) – If True, show the plot
- ****do_save** (*bool*) – If True, save the plot to disk

Returns Matplotlib figure and axes.

plot_ages (*pop*, ***kwargs*)

Plot a comparison of the expected and generated age distribution.

Parameters

- **pop** (*pop object*) – population, either synthpops.pop.Pop, sp.people.People, or dict
- ****left** (*float*) – Matplotlib.figure.subplot.left
- ****right** (*float*) – Matplotlib.figure.subplot.right
- ****top** (*float*) – Matplotlib.figure.subplot.top
- ****bottom** (*float*) – Matplotlib.figure.subplot.bottom
- ****color_1** (*str*) – color for expected data
- ****color_2** (*str*) – color for data from generated population
- ****fontsize** (*float*) – Matplotlib.figure.fontsize
- ****figname** (*str*) – name to save figure to disk
- ****comparison** (*bool*) – If True, plot comparison to the generated population
- ****do_show** (*bool*) – If True, show the plot
- ****do_save** (*bool*) – If True, save the plot to disk

Returns Matplotlib figure and axes.

Note: If using pop with type sp.people.Pop or dict, args must be supplied for the location parameters to get the expected distribution.

Example:

```
pars = {'n': 10e3, 'location':'seattle_metro', 'state_location':'Washington',
↪'country_location':'usa'}
pop = sp.Pop(**pars)
fig, ax = pop.plot_age_distribution_comparison()

popdict = pop.to_dict()
kwargs = pars.copy()
kwargs['datadir'] = sp.datadir
fig, ax = sp.plot_age_distribution_comparison(popdict, **kwargs)
```

plot_household_sizes (*pop*, ***kwargs*)

Plot a comparison of the expected and generated household size distribution.

Parameters

- **pop** (*pop object*) – population, either synthpops.pop.Pop or dict
- ****left** (*float*) – Matplotlib.figure.subplot.left
- ****right** (*float*) – Matplotlib.figure.subplot.right
- ****top** (*float*) – Matplotlib.figure.subplot.top

- ****bottom** (*float*) – Matplotlib.figure.subplot.bottom
- ****color_1** (*str*) – color for expected data
- ****color_2** (*str*) – color for data from generated population
- ****fontsize** (*float*) – Matplotlib.figure.fontsize
- ****figname** (*str*) – name to save figure to disk
- ****comparison** (*bool*) – If True, plot comparison to the generated population
- ****do_show** (*bool*) – If True, show the plot
- ****do_save** (*bool*) – If True, save the plot to disk

Returns Matplotlib figure and axes.

Note: If using pop with type dict, args must be supplied for the location parameter to get the expected rates. sp.people.People pop type not yet supported.

Example:

```
pars = {'n': 10e3, 'location':'seattle_metro', 'state_location':'Washington',
↪'country_location':'usa'}
pop = sp.Pop(**pars)
fig, ax = pop.plot_household_sizes()

popdict = pop.to_dict()
kwargs = pars.copy()
kwargs['datadir'] = sp.datadir
fig, ax = sp.plot_household_sizes(popdict, **kwargs)
```

plot_ltcf_resident_sizes(*pop*, ***kwargs*)

Plot a comparison of the expected and generated ltcf resident sizes.

Parameters

- **pop** (*pop object*) – population, either synthpops.pop.Pop or dict
- ****left** (*float*) – Matplotlib.figure.subplot.left
- ****right** (*float*) – Matplotlib.figure.subplot.right
- ****top** (*float*) – Matplotlib.figure.subplot.top
- ****bottom** (*float*) – Matplotlib.figure.subplot.bottom
- ****color_1** (*str*) – color for expected data
- ****color_2** (*str*) – color for data from generated population
- ****fontsize** (*float*) – Matplotlib.figure.fontsize
- ****figname** (*str*) – name to save figure to disk
- ****comparison** (*bool*) – If True, plot comparison to the generated population
- ****do_show** (*bool*) – If True, show the plot
- ****do_save** (*bool*) – If True, save the plot to disk

Returns Matplotlib figure and axes.

Note: If using pop with type dict, args must be supplied for the location parameter to get the expected rates. sp.people.People pop type not yet supported.

Example:

```
pars = {'n': 10e3, 'location':'seattle_metro', 'state_location':'Washington',
↪'country_location':'usa'}
pop = sp.Pop(**pars)
fig, ax = pop.plot_ltcf_resident_sizes()

popdict = pop.to_dict()
kwargs = pars.copy()
kwargs['datadir'] = sp.datadir
fig, ax = sp.plot_ltcf_resident_sizes(popdict, **kwargs)
```

plot_enrollment_rates_by_age(*pop*, ***kwargs*)

Plot a comparison of the expected and generated school enrollment rates by age.

Parameters

- **pop** (*pop object*) – population, either synthpops.pop.Pop or dict
- ****left** (*float*) – Matplotlib.figure.subplot.left
- ****right** (*float*) – Matplotlib.figure.subplot.right
- ****top** (*float*) – Matplotlib.figure.subplot.top
- ****bottom** (*float*) – Matplotlib.figure.subplot.bottom
- ****color_1** (*str*) – color for expected data
- ****color_2** (*str*) – color for data from generated population
- ****fontsize** (*float*) – Matplotlib.figure.fontsize
- ****figname** (*str*) – name to save figure to disk
- ****comparison** (*bool*) – If True, plot comparison to the generated population
- ****do_show** (*bool*) – If True, show the plot
- ****do_save** (*bool*) – If True, save the plot to disk

Returns Matplotlib figure and axes.

Note: If using pop with type dict, args must be supplied for the location parameter to get the expected rates. sp.people.People pop type not yet supported.

Example:

```
pars = {'n': 10e3, 'location':'seattle_metro', 'state_location':'Washington',
↪'country_location':'usa'}
pop = sp.Pop(**pars)
fig, ax = pop.plot_enrollment_rates_by_age()

popdict = pop.to_dict()
kwargs = pars.copy()
kwargs['datadir'] = sp.datadir
fig, ax = sp.plot_enrollment_rates_by_age(popdict, **kwargs)
```

plot_employment_rates_by_age(*pop*, ***kwargs*)
Plot a comparison of the expected and generated employment rates by age.

Parameters

- **pop** (*pop object*) – population, either synthpops.pop.Pop or dict
- ****left** (*float*) – Matplotlib.figure.subplot.left
- ****right** (*float*) – Matplotlib.figure.subplot.right
- ****top** (*float*) – Matplotlib.figure.subplot.top
- ****bottom** (*float*) – Matplotlib.figure.subplot.bottom
- ****color_1** (*str*) – color for expected data
- ****color_2** (*str*) – color for data from generated population
- ****fontsize** (*float*) – Matplotlib.figure.fontsize
- ****figname** (*str*) – name to save figure to disk
- ****comparison** (*bool*) – If True, plot comparison to the generated population
- ****do_show** (*bool*) – If True, show the plot
- ****do_save** (*bool*) – If True, save the plot to disk

Returns Matplotlib figure and axes.

Note: If using pop with type dict, args must be supplied for the location parameter to get the expected rates. sp.people.People pop type not yet supported.

Example:

```
pars = {'n': 10e3, 'location':'seattle_metro', 'state_location':'Washington',
↪'country_location':'usa'}
pop = sp.Pop(**pars)
fig, ax = pop.plot_employment_rates_by_age()

popdict = pop.to_dict()
kwargs = pars.copy()
kwargs['datadir'] = sp.datadir
fig, ax = sp.plot_employment_rates_by_age(popdict, **kwargs)
```

plot_school_sizes(*pop*, ***kwargs*)
Plot a comparison of the expected and generated school size distribution for each type of school expected.

Parameters

- **pop** (*pop object*) – population, either synthpops.pop.Pop, or dict
- ****with_school_types** (*type*) – If True, plot school size distributions by type, else plot overall school size distributions
- ****keys_to_exclude** (*str or list*) – school types to exclude
- ****left** (*float*) – Matplotlib.figure.subplot.left
- ****right** (*float*) – Matplotlib.figure.subplot.right
- ****top** (*float*) – Matplotlib.figure.subplot.top
- ****bottom** (*float*) – Matplotlib.figure.subplot.bottom

- ****hspace** (*float*) – Matplotlib.figure.subplot.hspace
- ****subplot_height** (*float*) – height of subplot in inches
- ****subplot_width** (*float*) – width of subplot in inches
- ****screen_height_factor** (*float*) – fraction of the screen height to use for display
- ****location_text_y** (*float*) – height to add location text to figure
- ****fontsize** (*float*) – Matplotlib.figure.fontsize
- ****rotation** (*float*) – rotation angle for xticklabels
- ****cmap** (*str or Matplotlib cmap*) – colormap
- ****figname** (*str*) – name to save figure to disk
- ****comparison** (*bool*) – If True, plot comparison to the generated population
- ****do_show** (*bool*) – If True, show the plot
- ****do_save** (*bool*) – If True, save the plot to disk

Returns Matplotlib figure and axes.

Note: If using pop with type sp.people.Pop or dict, args must be supplied for the location parameters to get the expected distribution.

Example:

```
pars = {'n': 10e3, 'location'='seattle_metro', 'state_location'='Washington',
↪'country_location'='usa'}
pop = sp.Pop(**pars)
fig, ax = pop.plot_school_sizes_by_type()

popdict = pop.to_dict()
kwargs = pars.copy()
kwargs['datadir'] = sp.datadir
fig, ax = sp.plot_school_sizes(popdict, **kwargs)
```

plot_workplace_sizes(*pop*, ***kwargs*)

Plot a comparison of the expected and generated workplace sizes for workplaces outside of schools or long term care facilities.

Parameters

- **pop** (*pop object*) – population, either synthpops.pop.Pop or dict
- ****left** (*float*) – Matplotlib.figure.subplot.left
- ****right** (*float*) – Matplotlib.figure.subplot.right
- ****top** (*float*) – Matplotlib.figure.subplot.top
- ****bottom** (*float*) – Matplotlib.figure.subplot.bottom
- ****color_1** (*str*) – color for expected data
- ****color_2** (*str*) – color for data from generated population
- ****fontsize** (*float*) – Matplotlib.figure.fontsize
- ****figname** (*str*) – name to save figure to disk

- ****comparison** (*bool*) – If True, plot comparison to the generated population
- ****do_show** (*bool*) – If True, show the plot
- ****do_save** (*bool*) – If True, save the plot to disk

Returns Matplotlib figure and axes.

Note: If using pop with type dict, args must be supplied for the location parameter to get the expected rates. sp.people.People pop type not yet supported.

Example:

```
pars = {'n': 10e3, 'location':'seattle_metro', 'state_location':'Washington',
↪'country_location':'usa'}
pop = sp.Pop(**pars)
fig, ax = pop.plot_workplace_sizes()

popdict = pop.to_dict()
kwargs = pars.copy()
kwargs['datadir'] = sp.datadir
fig, ax = sp.plot_workplace_sizes(popdict, **kwargs)
```

plot_household_head_ages_by_size(*pop*, ***kwargs*)

Plot a comparison of the expected and generated age distribution of the household heads by the household size, presented as matrices. The age distribution of household heads is binned to match the expected data.

Parameters

- **pop** (*sp.Pop*) – population
- ****figname** (*str*) – name to save figure to disk
- ****figdir** (*str*) – directory to save the plot if provided
- ****title_prefix** (*str*) – used to prefix the title of the plot
- ****fontsize** (*float*) – Matplotlib.figure.fontsize
- ****cmap** (*str or Matplotlib cmap*) – colormap
- ****do_show** (*bool*) – If True, show the plot
- ****do_save** (*bool*) – If True, save the plot to disk

Returns Matplotlib figure and axes.

Example:

```
pars = {'n': 10e3, 'location': 'seattle_metro', 'state_location': 'Washington',
↪'country_location': 'usa'}
pop = sp.Pop(**pars)
fig, ax = plot_household_head_ages_by_size(pop)

kwargs = pars.copy()
kwargs['cmap'] = 'rocket'
fig, ax = plot_household_head_ages_by_size(pop, **kwargs)
```

plot_contact_counts(*contact_counter*, ***kwargs*)

Plot the number of contacts by contact types as a histogram. The contact_counter is a dictionary with keys = people_types (default to school layer ['sc_student', 'sc_teacher', 'sc_staff']) and each value is a dictionary

which stores the list of counts for each type of contact, for example ['sc_teacher', 'sc_student', 'sc_staff', 'all_staff', 'all'].

Parameters

- **contact_counter** (*dict*) – A dictionary with people_types as keys and value as list of counts for each type of contacts
- ****title_prefix** (*str*) – optional title prefix for the figure
- ****figname** (*str*) – name to save figure to disk
- ****fontsize** (*float*) – Matplotlib.figure.fontsize

Returns Matplotlib figure and axes of the histograms of contact distributions for the corresponding contact_counter.

6.2.10 synthpops.pop module

This module provides the main class for interacting with SynthPops, the Pop class.

class Pop(*n=None*, *max_contacts=None*, *ltcf_pars=None*, *school_pars=None*, *with_industry_code=False*, *with_facilities=False*, *use_default=False*, *use_two_group_reduction=True*, *average_LTCF_degree=20*, *ltcf_staff_age_min=20*, *ltcf_staff_age_max=60*, *with_school_types=False*, *school_mixing_type='random'*, *average_class_size=20*, *inter_grade_mixing=0.1*, *average_student_teacher_ratio=20*, *average_teacher_teacher_degree=3*, *teacher_age_min=25*, *teacher_age_max=75*, *with_non_teaching_staff=False*, *average_student_all_staff_ratio=15*, *average_additional_staff_degree=20*, *staff_age_min=20*, *staff_age_max=75*, *rand_seed=None*, *country_location=None*, *state_location=None*, *location=None*, *sheet_name=None*, *household_method='infer_ages'*, *smooth_ages=False*, *window_length=7*, *do_make=True*)

Bases: sciris.sc_utils.prettyobj

Make a full population network including both people (ages, sexes) and contacts. By default uses Seattle, Washington data. Note about the household methods available: 'infer_ages' and 'fixed_ages'.

If using 'infer_ages', then the ages of individuals in the population are generated by first placing individuals into households using the age of the head of households or reference individuals (always an adult), household age mixing patterns, household sizes, and the age distribution from data (census or other sources).

If using 'fixed_ages', then individuals are pre-assigned ages according to the age distribution and placed into households using the age of the head of households or reference individuals, household age mixing patterns, and household sizes.

Parameters

- **n** (*int*) – The number of people to create.
- **max_contacts** (*dict*) – A dictionary for maximum number of contacts per layer: keys must be "W" (work).
- **ltcf_pars** (*dict*) – If supplied, replace default LTCF parameters
- **school_pars** (*dict*) – if supplied, replace default school parameters
- **with_industry_code** (*bool*) – If True, assign industry codes for workplaces, currently only possible for cached files of populations in the US.
- **with_facilities** (*bool*) – If True, create long term care facilities, currently only available for locations in the US.

- **use_default** (*bool*) – If True, use default data from settings.location, settings.state, settings.country.
- **use_two_group_reduction** (*bool*) – If True, create long term care facilities with reduced contacts across both groups.
- **average_LTCF_degree** (*float*) – default average degree in long term care facilities.
- **ltcf_staff_age_min** (*int*) – Long term care facility staff minimum age.
- **ltcf_staff_age_max** (*int*) – Long term care facility staff maximum age.
- **with_school_types** (*bool*) – If True, creates explicit school types.
- **school_mixing_type** (*str or dict*) – The mixing type for schools, 'random', 'age_clustered', or 'age_and_class_clustered' if string, and a dictionary of these by school type otherwise.
- **average_class_size** (*float*) – The average classroom size.
- **inter_grade_mixing** (*float*) – The average fraction of edges rewired to create edges between grades in the same school when school_mixing_type is 'age_clustered'
- **average_student_teacher_ratio** (*float*) – The average number of students per teacher.
- **average_teacher_teacher_degree** (*float*) – The average number of contacts per teacher with other teachers.
- **teacher_age_min** (*int*) – The minimum age for teachers.
- **teacher_age_max** (*int*) – The maximum age for teachers.
- **with_non_teaching_staff** (*bool*) – If True, includes non teaching staff.
- **average_student_all_staff_ratio** (*float*) – The average number of students per staff members at school (including both teachers and non teachers).
- **average_additional_staff_degree** (*float*) – The average number of contacts per additional non teaching staff in schools.
- **staff_age_min** (*int*) – The minimum age for non teaching staff.
- **staff_age_max** (*int*) – The maximum age for non teaching staff.
- **rand_seed** (*int*) – Start point random sequence is generated from.
- **country_location** (*string*) – name of the country the location is in
- **state_location** (*string*) – name of the state the location is in
- **location** (*string*) – name of the location
- **sheet_name** (*string*) – sheet name where data is located
- **household_method** (*string*) – name of household generation method used; for details see above.
- **smooth_ages** (*bool*) – If True, use smoothed out age distribution.
- **window_length** (*int*) – length of window over which to average or smooth out age distribution
- **do_make** (*bool*) – whether to make the population

Returns A dictionary of the full population with ages, connections, and other attributes.

Return type network (dict)

generate()
Actually generate the network.

Returns A dictionary of the full population with ages, connections, and other attributes.

Return type network (dict)

set_layer_classes()
Add layer classes.

clean_up_layer_info()
Clean up temporary data from the pop object after storing them in specific layer classes.

pop_item(*key*)
Pop key from self.

to_dict()
Export to a dictionary – official way to get the popdict.

Example:

```
popdict = pop.to_dict()
```

to_json(*filename*, *indent=2*, ***kwargs*)
Export to a JSON file.

Example:

```
pop.to_json('my-pop.json')
```

save(*filename*, ***kwargs*)
Save population to an binary, gzipped object file.

Example:

```
pop.save('my-pop.pop')
```

static load(*filename*, **args*, ***kwargs*)
Load from disk from a gzipped pickle.

Parameters

- **filename** (*str*) – the name or path of the file to load from
- **kwargs** – passed to sc.loadobj()

Example:

```
pop = sp.Pop.load('my-pop.pop')
```

initialize_households_list()
Initialize a new households list.

initialize_empty_households(*n_households=None*)
Create a list of empty households.

Parameters **n_households** (*int*) – the number of households to initialize

populate_households(*households*, *age_by_uid*)
Populate all of the households. Store each household at the index corresponding to it's hhid.

Parameters

- **households** (*list*) – list of lists where each sublist represents a household and contains the ids of the household members
- **age_by_uid** (*dict*) – dictionary mapping each person's id to their age

get_household(*hhid*)
Return household with id: hhid.

Parameters **hhid** (*int*) – household id number

Returns A populated household.

Return type sp.Household

add_household(*household*)
Add a household to the list of households.

Parameters **household** (*sp.Household*) – household with at minimum the hhid, member_uids, member_ages, reference_uid, and reference_age.

initialize_workplaces_list()
Initialize a new workplaces list.

initialize_empty_workplaces(*n_workplaces=None*)
Create a list of empty workplaces.

Parameters **n_households** (*int*) – the number of workplaces to initialize

populate_workplaces(*workplaces*)
Populate all of the workplaces. Store each workplace at the index corresponding to it's wpid.

Parameters

- **workplaces** (*list*) – list of lists where each sublist represents a workplace and contains the ids of the workplace members
- **age_by_uid** (*dict*) – dictionary mapping each person's id to their age

get_workplace(*wpid*)
Return workplace with id: wpid.

Parameters **wpid** (*int*) – workplace id number

Returns A populated workplace.

Return type sp.Workplace

add_workplace(*workplace*)
Add a workplace to the list of workplaces.

Parameters **workplace** (*sp.Workplace*) – workplace with at minimum the wpid, member_uids, member_ages, reference_uid, and reference_age.

initialize_ltcfs_list()
Initialize a new ltcfs list.

initialize_empty_ltcfs(*n_ltcfs=None*)
Create a list of empty ltcfs.

Parameters **n_ltcfs** (*int*) – the number of ltcfs to initialize

populate_ltcfs(*resident_lists*, *staff_lists*)
Populate all of the ltcfs. Store each ltcf at the index corresponding to it's ltcfid.

Parameters

- **residents_list** (*list*) – list of lists where each sublist represents a ltcf and contains the ids of the residents
- **staff_lists** (*list*) – list of lists where each sublist represents a ltcf and contains the ids of the staff

get_ltcf(*ltcfid*)
Return ltcf with id: ltcfid.

Parameters **ltcfid** (*int*) – ltcf id number

Returns A populated ltcf.

Return type sp.LongTermCareFacility

add_ltcf(*ltcf*)
Add a ltcf to the list of ltcfs.

Parameters **ltcf** (*sp.LongTermCareFacility*) – ltcf with at minimum the ltcfid, resident_uids, staff_uids, resident_ages, staff_ages, reference_uid, and reference_age.

initialize_schools_list()
Initialize a new schools list.

initialize_empty_schools(*n_schools=None*)
Create a list of empty schools.

Parameters **n_schools** (*int*) – the number of schools to initialize

populate_schools(*student_lists*, *teacher_lists*, *non_teaching_staff_lists*, *age_by_uid*, *school_types=None*, *school_mixing_types=None*)
Populate all of the schools. Store each school at the index corresponding to it's scid.

Parameters

- **student_lists** (*list*) – list of lists where each sublist represents a school and contains the ids of the students
- **teacher_lists** (*list*) – list of lists where each sublist represents a school and contains the ids of the teachers
- **non_teaching_staff_lists** (*list*) – list of lists where each sublist represents a school and contains the ids of the non teaching staff
- **age_by_uid** (*dict*) – dictionary mapping each person's id to their age
- **school_types** (*list*) – list of the school types
- **school_mixing_types** (*list*) – list of the school mixing types

get_school(*scid*)
Return school with id: scid.

Parameters **scid** (*int*) – school id number

Returns A populated school.

Return type sp.School

add_school(*school*)
Add a school to the list of schools.

Parameters **school** (*sp.School*) – school

populate_all_classrooms(*schools_in_groups*)
Populate all of the classrooms in schools for each school that has school_mixing_type equal to 'age_and_class_clustered'. Each classroom will be indexed at id clid.

Parameters **schools_in_groups** (*dict*) – a dictionary representing each school in terms of student_groups and teacher_groups corresponding to classrooms

get_classroom(*scid*, *clid*)
Return classroom with id: clid from school with id: scid.

Parameters

- **scid** (*int*) – school id number
- **clid** (*int*) – classroom id number

Returns A populated classroom.

Return type sp.Classroom

compute_information()
Computing an advanced description of the population.

compute_summary()
Compute summaries and add to pop post generation.

summarize(*return_msg=False*)
Print and optionally return a brief summary string of the pop.

count_pop_ages()
Create an age count of the generated population post generation.

Returns Dictionary of the age count of the generated population.

Return type dict

get_household_sizes()
Create household sizes in the generated population post generation.

Returns Dictionary of household size by household id (hhid).

Return type dict

count_household_sizes()
Count of household sizes in the generated population.

Returns Dictionary of the count of household sizes.

Return type dict

get_household_heads()
Get the ids of the head of households in the generated population post generation.

get_household_head_ages()
Get the age of the head of each household in the generated population post generation.

count_household_head_ages(*bins=None*)
Count of household head ages in the generated population.

Parameters **bins** (*array*) – If supplied, use this to create a binned count of the household head ages. Otherwise, count discrete household head ages.

Returns Dictionary of the count of household head ages.

Return type dict

get_household_head_ages_by_size()
Get the count of households by size and the age of the head of the household, assuming the minimal household members id is the id of the head of the household.

Returns An array with row as household size and columns as household head age brackets.

Return type np.ndarray

get_ltcf_sizes(*keys_to_exclude=[]*)
Create long term care facility sizes in the generated population post generation.

Parameters **keys_to_exclude** (*list*) – possible keys to exclude for roles in long term care facilities. See notes.

Returns Dictionary of the size for each long term care facility generated.

Return type dict

Notes

keys_to_exclude is an empty list by default, but can contain the different long term care facility roles: 'snf_res' for residents and 'snf_staff' for staff. If either role is included in the parameter keys_to_exclude, then individuals with that value equal to 1 will not be counted.

count_ltcf_sizes(*keys_to_exclude=[]*)
Count of long term care facility sizes in the generated population.

Parameters **keys_to_exclude** (*list*) – possible keys to exclude for roles in long term care facilities. See notes.

Returns Dictionary of the count of long term care facility sizes.

Return type dict

Notes

keys_to_exclude is an empty list by default, but can contain the different long term care facility roles: 'snf_res' for residents and 'snf_staff' for staff. If either role is included in the parameter keys_to_exclude, then individuals with that value equal to 1 will not be counted.

count_enrollment_by_age()
Create enrollment count by age for students in the generated population post generation.

Returns Dictionary of the count of enrolled students by age in the generated population.

Return type dict

enrollment_rates_by_age
Enrollment rates by age for students in the generated population.

Returns Dictionary of the enrollment rates by age for students in the generated population.

Return type dict

count_enrollment_by_school_type(**args*, ***kwargs*)
Create enrollment sizes by school types in the generated population post generation.

Returns List of generated enrollment sizes by school type.

Return type list

count_employment_by_age()
Create employment count by age for workers in the generated population post generation.

Returns Dictionary of the count of employed workers by age in the generated population.

Return type dict

employment_rates_by_age
Employment rates by age for workers in the generated population.

Returns Dictionary of the employment rates by age for workers in the generated population.

Return type dict

get_workplace_sizes()
Create workplace sizes in the generated population post generation.

Returns Dictionary of workplace size by workplace id (wpid).

Return type dict

count_workplace_sizes()
Count of workplace sizes in the generated population.

Returns Dictionary of the count of workplace sizes.

Return type dict

get_contact_counts_by_layer(*layer='S'*, ***kwargs*)
Get the number of contacts by layer.

Returns Dictionary of the count of contacts in the layer for the different people types in the layer. See sp.contact_networks.get_contact_counts_by_layer() for method details.

Return type dict

to_people()
Convert to the alternative People representation of a population

plot_people(**args*, ***kwargs*)
Placeholder example of plotting the people in a population.

plot_contacts(**args*, ***kwargs*)
Plot matrices of the contacts for a given layer or layers.

plot_contact_counts(*contact_counter*, ***kwargs*)
Plot the number of contacts by contact types as a histogram.

Parameters

- **contact_counter** (*dict*) – A dictionary with people_types as keys and value as list of counts for each type of contacts
- ****title_prefix** (*str*) – optional title prefix for the figure
- ****figname** (*str*) – name to save figure to disk
- ****fontsize** (*float*) – Matplotlib.figure.fontsize

Returns Matplotlib figure and axes of the histograms of contact distributions for the corresponding contact_counter.

Examples:

```
pars = {'n': 10e3, 'location': 'seattle_metro', 'state_location': 'Washington
↪', 'country_location': 'usa'}
pop = sp.Pop(**pars)
layer = 'S'
contact_counter = pop.get_contact_counts_by_layer(layer=layer)
fig, ax = pop.plot_contact_counts(contact_counter)
```

plot_ages(***kwargs*)
Plot a comparison of the expected and generated age distribution.

Example:

```
pars = {'n': 10e3, 'location':'seattle_metro', 'state_location':'Washington',
↪'country_location':'usa'}
pop = sp.Pop(**pars)
fig, ax = pop.plot_ages()
```

plot_household_sizes(***kwargs*)
Plot a comparison of the expected and generated household size distribution.

Example:

```
pars = {'n': 10e3, 'location':'seattle_metro', 'state_location':'Washington',
↪'country_location':'usa'}
pop = sp.Pop(**pars)
fig, ax = pop.plot_household_sizes()
```

plot_household_head_ages_by_size(***kwargs*)
Plot a comparison of the expected and generated age distribution of the household heads by the household size.

Examples:

```
pars = {'n': 10e3, 'location':'seattle_metro', 'state_location':'Washington',
↪'country_location':'usa'}
pop = sp.Pop(**pars)
fig, ax = pop.plot_household_head_ages_by_size()

kwargs = pars.copy()
fig, ax = pop.plot_household_head_ages_by_size(**kwargs)
```

plot_ltcf_resident_sizes(***kwargs*)
Plot a comparison of the expected and generated ltcf resident sizes.

Examples:

```
pars = {'n': 10e3, 'location':'seattle_metro', 'state_location':'Washington',
↪'country_location':'usa'}
pop = sp.Pop(**pars)
fig, ax = pop.plot_ltcf_resident_sizes()
```

plot_enrollment_rates_by_age(***kwargs*)
Plot a comparison of the expected and generated enrollment rates by age.

Example:

```
pars = {'n': 10e3, 'location':'seattle_metro', 'state_location':'Washington',
↪'country_location':'usa'}
pop = sp.Pop(**pars)
fig, ax = pop.plot_enrollment_rates_by_age()
```

plot_employment_rates_by_age(***kwargs*)
Plot a comparison of the expected and generated employment rates by age.

Example:

```
pars = {'n': 10e3, 'location':'seattle_metro', 'state_location':'Washington',
↪'country_location':'usa'}
pop = sp.Pop(**pars)
fig, ax = pop.plot_employment_rates_by_age()
```

plot_school_sizes(**args*, ***kwargs*)
Plot a comparison of the expected and generated school size distributions by school type.

Example:

```
pars = {'n': 10e3, 'location':'seattle_metro', 'state_location':'Washington',
↪'country_location':'usa'}
pop = sp.Pop(**pars)
fig, ax = pop.plot_school_sizes()
```

plot_workplace_sizes(***kwargs*)
Plot a comparison of the expected and generated workplace sizes for workplaces that are not schools or long term care facilities.

Examples:

```
pars = {'n': 10e3, 'location':'seattle_metro', 'state_location':'Washington',
↪'country_location':'usa'}
pop = sp.Pop(**pars)
fig, ax = pop.plot_ltcf_resident_sizes()
```

make_population(**args*, ***kwargs*)
Interface to sp.Pop().to_dict(). Included for backwards compatibility.

generate_synthetic_population(**args*, ***kwargs*)
For backwards compatibility only.

6.2.11 synthpops.sampling module

Sample distributions, either from real world data or from uniform distributions.

set_seed(*seed=None*)
Reset the random seed – complicated because of Numba.

fast_choice(*weights*)
Choose an option – quickly – from the provided weights. Weights do not need to be normalized.

Reimplementation of random.choices(), removing everything inessential.

Example

fast_choice([0.1,0.2,0.3,0.2,0.1]) # might return 2

sample_single_dict(*distr_keys*, *distr_vals*)
Sample from a distribution.

Parameters **distr** (*dict or np.ndarray*) – distribution

Returns A single sampled value from a distribution.

sample_single_arr(*distr*)
Sample from a distribution.

Parameters **distr** (*dict or np.ndarray*) – distribution

Returns A single sampled value from a distribution.

resample_age(*age_dist_vals*, *age*)
Resample age from single year age distribution.

Parameters

- **single_year_age_distr** (*arr*) – age distribution, ordered by age
- **age** (*int*) – age as an integer

Returns Resampled age as an integer.

sample_from_range(*distr*, *min_val*, *max_val*)
Sample from a distribution from min_val to max_val, inclusive.

Parameters

- **distr** (*dict*) – distribution with integer keys
- **min_val** (*int*) – minimum of the range to sample from
- **max_val** (*int*) – maximum of the range to sample from

Returns A sampled number from the range min_val to max_val in the distribution distr.

check_dist(*actual*, *expected*, *std=None*, *dist='norm'*, *check='dist'*, *label=None*, *alpha=0.05*, *size=10000*, *verbose=True*, *die=False*, *stats=False*)
Check whether counts match the expected distribution. The distribution can be any listed in scipy.stats. The parameters for the distribution should be supplied via the "expected" argument. The standard deviation for a normal distribution is a special case; it can be supplied separately or calculated from the (actual) data.

Parameters

- **actual** (*int, float, or array*) – the observed value, or distribution of values
- **expected** (*int, float, tuple*) – the expected value; or, a tuple of arguments
- **std** (*float*) – for normal distributions, the standard deviation of the expected value (taken from data if not supplied)
- **dist** (*str*) – the type of distribution to use
- **check** (*str*) – what to check: 'dist' = entire distribution (default), 'mean' (equivalent to supplying np.mean(actual)), or 'median'
- **label** (*str*) – the name of the variable being tested
- **alpha** (*float*) – the significance level at which to reject the null hypothesis
- **size** (*int*) – the size of the sample from the expected distribution to compare with if distribution is discrete
- **verbose** (*bool*) – print a warning if the null hypothesis is rejected
- **die** (*bool*) – raise an exception if the null hypothesis is rejected
- **stats** (*bool*) – whether to return statistics

Returns whether null hypothesis is rejected, pvalue, number of samples, expected quintiles, observed quintiles, and the observed quantile.

Return type If stats is True, returns statistics

Examples:

```
sp.check_dist(actual=[3,4,4,2,3], expected=3, dist='poisson')
sp.check_dist(actual=[0.14, -3.37,  0.59, -0.07], expected=0, std=1.0, dist='norm
↪')
sp.check_dist(actual=5.5, expected=(1, 5), dist='lognorm')
```

check_normal (**args*, ***kwargs*)
Alias to check_dist(dist='normal')

check_poisson (**args*, ***kwargs*)
Alias to check_dist(dist='poisson')

check_truncated_poisson (*testdata*, *mu*, *lowerbound=None*, *upperbound=None*, *skipcheck=False*, ***kwargs*)
test if data fits in truncated poisson distribution between upperbound and lowerbound using kstest :param testdata: data to be tested :type testdata: array :param mu: expected mean for the poisson distribution :type mu: float :param lowerbound: lowerbound for truncation :type lowerbound: float :param upperbound: upperbound for truncation :type upperbound: float

Returns (bool) return True if statistic check passed, else return False

statistic_test (*expected*, *actual*, *test=<function chisquare>*, *verbose=True*, *die=False*, ***kwargs*)
Perform statistical checks for expected and actual data based on the null hypothesis that expected and actual distributions are identical. Throw assertion if the expected and actual data differ significantly based on the test selected. See https://docs.scipy.org/doc/scipy/reference/stats.html#statistical-tests.

Parameters

- **expected** (*array*) – the expected value; or, a tuple of arguments
- **actual** (*array*) – the observed value, or distribution of values
- **test** (*scipy.stats*) – scipy statistical tests functions, for example scipy.stats.chisquare
- **verbose** (*bool*) – print a warning if the null hypothesis is rejected
- **die** (*bool*) – raise an exception if the null hypothesis is rejected
- ****kwargs** (*dict*) – optional arguments for statistical tests

Returns None.

6.2.12 synthpops.schools module

This module generates school contacts by class and grade in flexible ways. Contacts can be clustered into classes and also mixed across the grade and across the school.

H. Guclu et. al (2016) shows that mixing across grades is low for public schools in elementary and middle schools. Mixing across grades is however higher in high schools.

Functions in this module are flexible to allow users to specify the inter-grade mixing (for 'age_clustered' school_mixing_type), and to choose whether contacts are clustered within a grade. Clustering contacts across different grades is not supported because there is no data to suggest that this happens commonly.

class School (*scid=None*, *sc_type=None*, *school_mixing_type=None*, *student_uids=array([]*, *dtype=int64)*, *teacher_uids=array([]*, *dtype=int64)*, *non_teaching_staff_uids=array([]*, *dtype=int64)*, ***kwargs*)
Bases: *synthpops.base.LayerGroup*

A class for individual schools and methods to operate on each.

Parameters **kwargs** (*dict*) – data dictionary of the school

Class constructor for an base empty setting group.

Parameters

- ****scid** (*int*) – id of the school
- ****sc_type** (*str*) – school type defined by grade/age ranges
- ****school_mixing_type** (*str*) – the mixing type of the school, 'random', 'age_clustered', or 'age_and_class_clustered' if str. Else, None. See sp.schools.add_school_edges() for more information.
- ****student_uids** (*np.array*) – ids of student members
- ****teacher_uids** (*np.array*) – ids of teacher members
- ****non_teaching_staff_uids** (*np.array*) – ids of non_teaching_staff members

validate()
Check that information supplied to make a school is valid and update to the correct type if necessary.

member_uids
students, teachers, and non teaching staff.

Returns school member ids

Return type np.ndarray

Type Return ids of all school members

member_ages(*age_by_uid*)
Return ages of all school members: students, teachers, and non teaching staff.

Parameters **age_by_uid** (*np.ndarray*) – mapping of age to uid

Returns school member ages

Return type np.ndarray

student_ages(*age_by_uid*)
Return student ages in the school.

Parameters **age_by_uid** (*np.ndarray*) – mapping of age to uid

Returns student ages in school

Return type np.ndarray

teacher_ages(*age_by_uid*)
Return teacher ages in the school.

Parameters **age_by_uid** (*np.ndarray*) – mapping of age to uid

Returns teacher ages in school

Return type np.ndarray

non_teaching_staff_ages(*age_by_uid*)
Return non-teaching staff ages in the school.

Parameters **age_by_uid** (*np.ndarray*) – mapping of age to uid

Returns non-teaching staff ages in school

Return type np.ndarray

get_classroom(*clid*)
Return the classroom indexed at clid if school_mixing_type is equal to 'age_and_class_clustered'.

Parameters **clid** (*int*) – classroom id number

Returns the classroom indexed at clid

Return type sp.Classroom

class Classroom(*clid=None*, *student_uids=array([]*, *dtype=int64)*, *teacher_uids=array([]*, *dtype=int64)*, ***kwargs*)

Bases: *synthpops.base.LayerGroup*

A class for individual classrooms and methods to operate on each.

Parameters **kwargs** (*dict*) – data dictionary of the classroom

Class constructor for an base empty setting group.

Parameters

- ****clid** (*int*) – id of the classroom
- ****student_uids** (*np.array*) – ids of student members
- ****teacher_uids** (*np.array*) – ids of teacher members

validate()

Check that information supplied to make a school is valid and update to the correct type if necessary.

member_uids

students and teachers.

Returns classroom member ids

Return type np.ndarray

Type Return ids of all classroom members

member_ages(*age_by_uid*)

Return ages of all classroom members: students and teachers.

Parameters **age_by_uid** (*np.ndarray*) – mapping of age to uid

Returns classroom member ages

Return type np.ndarray

student_ages(*age_by_uid*)

Return student ages in the classroom.

Parameters **age_by_uid** (*np.ndarray*) – mapping of age to uid

Returns student ages in classroom

Return type np.ndarray

teacher_ages(*age_by_uid*)

Return teacher ages in the classroom.

Parameters **age_by_uid** (*np.ndarray*) – mapping of age to uid

Returns teacher ages in classroom

Return type np.ndarray

get_school_type_labels()

count_enrollment_by_age(*popdict*)

Get enrollment count by age for students in the popdict.

Parameters **popdict** (*dict*) – population dictionary

Returns Dictionary of the count of enrolled students by age in popdict.

Return type dict

get_enrollment_rates_by_age(*enrollment_count_by_age*, *age_count*)
Get enrollment rates by age.

Parameters

- **enrollment_count_by_age** (*dict*) – dictionary of the count of enrolled students
- **age_count** (*dict*) – dictionary of the age count

Returns Dictionary of the enrollment rates by age.

Return type dict

count_enrollment_by_school_type(*popdict*, ***kwargs*)
Get enrollment sizes by school types in popdict.

Parameters

- **popdict** (*dict*) – population dictionary
- ****with_school_types** (*bool*) – If True, return enrollment by school types as defined in the popdict. Otherwise, combine all enrollment sizes for a school type of None.
- ****keys_to_exclude** (*list*) – school types to exclude

Returns Dictionary of generated enrollment sizes by school type.

Return type dict

get_generated_school_size_distributions(*enrollment_by_school_type*, *bins*)
Get school size distributions by type.

Parameters

- **enrollment_by_school_type** (*dict*) – generated enrollment sizes by school types
- **bins** (*list*) – school size bins

Returns Dictionary of generated school size distribution by school type.

Return type dict

6.2.13 synthpops.version module

6.2.14 synthpops.workplaces module

class Workplace(*wpid=None*, ***kwargs*)
Bases: *synthpops.base.LayerGroup*

A class for individual workplaces and methods to operate on each.

Parameters **kwargs** (*dict*) – data dictionary of the workplace

Class constructor for empty workplace.

Parameters

- ****wpid** (*int*) – workplace id
- ****member_uids** (*np.array*) – ids of workplace members

validate()
Check that information supplied to make a workplace is valid and update to the correct type if necessary.

get_workplace(*pop*, *wpid*)
Return workplace with id: wpid.

Parameters

- **pop** (*sp.Pop*) – population
- **wpid** (*int*) – workplace id number

Returns A populated workplace.

Return type sp.Workplace

add_workplace(*pop*, *workplace*)
Add a workplace to the list of workplaces.

Parameters

- **pop** (*sp.Pop*) – population
- **workplace** (*sp.Workplace*) – workplace with at minimum the wpid and member_uids.

initialize_empty_workplaces(*pop*, *n_workplaces=None*)
Array of empty workplaces.

Parameters

- **pop** (*sp.Pop*) – population
- **n_workplaces** (*int*) – the number of workplaces to initialize

populate_workplaces(*pop*, *workplaces*)
Populate all of the workplaces. Store each workplace at the index corresponding to it's wpid.

Parameters

- **pop** (*sp.Pop*) – population
- **workplaces** (*list*) – list of lists where each sublist represents a workplace and contains the ids of the workplace members

Notes

If number of workplaces (n) is fewer than existing workplaces, it will only replace the first n workplaces. Otherwise the existing workplaces will be overwritten by the input workplaces.

count_employment_by_age(*popdict*)
Get employment count by age for workers in the popdict. Workers can be in different possible layers: as staff in long term care facilities (LTCF), as teachers or staff in schools (S), or as workers in other workplaces (W).

Parameters **popdict** (*dict*) – population dictionary

Returns Dictionary of the count of employed people by age in popdict.

Return type dict

get_employment_rates_by_age(*employment_count_by_age*, *age_count*)
Get employment rates by age.

Parameters

- **employment_count_by_age** (*dict*) – dictionary of the count of employed people

- **age_count** (*dict*) – dictionary of the age count

Returns Dictionary of the employment rates by age.

Return type dict

get_workplace_sizes(*popdict*)
Get workplace sizes of regular workplaces in popdict. This only includes workplaces that are not long term care facilities (LTCF) or schools (S).

Parameters **popdict** (*dict*) – population dictionary

Returns Dictionary of the generated workplace sizes for each regular workplace.

Return type dict

get_generated_workplace_size_distribution(*workplace_sizes*, *bins*)
Get workplace size distribution.

Parameters

- **workplace_sizes** (*dict*) – generated workplace sizes by workplace id (wpid)
- **bins** (*list*) – workplace size bins

Returns Dictionary of generated workplace size distribution.

Return type dict

CHAPTER 7

Glossary

contact layers Each of the layers of the population network that is a representation of all of the pairwise connections between people in a given location, such as school, work, or households.

node In network science, the discrete object being represented. In SynthPops, nodes represent people and can have attributes like age assigned.

edge In network science, the interactions or connections between discrete objects. In SynthPops, edges represent interactions between people, with attributes like the setting in which the interactions take place (for example, household, school, or work). The relationship between the interaction setting and properties governing disease transmission, such as frequency of contact and risk associated with each contact, is mapped separately by Covasim or other *agent-based model*. SynthPops reports whether the edge exists or not.

agent-based model A type of simulation that models the actions and interactions of autonomous agents (both individual and collective entities such as organizations or groups).

time step A discrete number of hours or days in which the "simulation states" of all "simulation objects" (interventions, infections, immune systems, or individuals) are updated in a simulation. Each time step will complete processing before launching the next one. For example, a time step would process the migration data for populations moving between nodes via rail, airline, and road. The migration of individuals between nodes is the last step of the time step after updating states.

household contact layer The layer in the population network that represents all of the pairwise connections between people in households. All people must be part of the household contact layer, though some households may consist of a single person.

school contact layer The layer in the population network that represents all of the pairwise connections between people in schools. This includes both students and teachers. The school and workplace contact layers are mutually exclusive, someone cannot be both a student and a worker.

workplace contact layer The layer in the population network that represents all of the pairwise connections between people in workplaces excluding teachers in schools. The school and workplace contact layers are mutually exclusive, someone cannot be both a student and a worker.

location The location in which a set of SynthPops input data are valid. This is often geographic but could be administrative or specific to for example a sub-population within a geographic region. Locations are organized hierarchically. Locations are defined by a location file which contain data for the associated population. Child

locations can inherit input data values from their parent location. Supplementing the data in these files is encouraged if available.

Python Module Index

S

Index

D

E

F

G

H

I

K

L

M

N

P

R

S

T

U

V

W