

March 2, 2021



AN ABM FRAMEWORK FOR SIMULATING THE FLOW OF ANTI-MICROBIAL RESISTANCE IN DRUM PROJECT STUDY SITES.

Simon Alderton* & Chris Jewell
s.alderton@lancaster.ac.uk



DRUM

Drivers of Resistance in Uganda & Malawi

Contents

Contents	2
List of Figures	4
1 Overview	5
1 Summary	5
1.1 Usage	5
2 Code Structure & Workflow	6
3 Directory Tree Structure	8
4 \data	9
5 \docs	9
6 \drum	10
7 \envs	11
8 \tests	11
2 Data	12
1 Summary	12
2 Coverage	12
3 Synthetic Population	16
1 Initial Method	16
2 Config File	16
3 Method	17
4 Analysis	17
4.1 Ndirande STRATAA vs. Ndirande synthetic.	17
4.2 Example Synthetic Household Spatial Distributions.	23
4.3 Generating 100 populations for each site.	27
5 Issues/Review	31
4 Networks	32
1 Summary	32
2 Class Structure	32
3 Config File	32
4 Network Generation	33
5 Model	35
1 Summary	35
2 Config File	35
3 Merge networks	35
4 Schedule	36

6	Simulator	37
1	Summary	37
2	Config File	37
3	Initialise	38
4	Iterate	38
4.1	Seed infected population	38
5	Output	39
5.1	Infection Data	39
5.2	Network Information	39
7	Analysis	40
1	Summary	40
2	Config File	40
3	Analysis A	40
4	Analysis B	41
A	'Run' Script	42
1	drum.py	42
1.1	Config File	42
1.2	drum.py Code	42

List of Figures

1.1	Code Workflow	6
1.2	Inputs and outputs of the DRUM ABM modules.	7
2.1	Ndirande population/household location information.	13
2.2	Chileka population/household location information.	14
2.3	Chikwawa population/household location information.	15
3.1	STRATAA census vs. synthetic population, coloured by household size.	18
3.2	Age distributions for STRATAA, the Ndirande SWARM sample, and the Ndirande synthetic population.	19
3.3	Household size distributions for STRATAA, the Ndirande SWARM sample, and the Ndirande synthetic population.	20
3.4	Ethnicity distributions for STRATAA, the Ndirande SWARM sample, and the Ndirande synthetic population.	21
3.5	Employment distributions for STRATAA, the Ndirande SWARM sample, and the Ndirande synthetic population. N.B. STRATAA ‘empty-fields’ largely represent respondents under the age of 15. It is expected that these would translate to additional numbers in the ‘student’ or ‘unemployed’ categories.	22
3.6	Example Chikwawa Synthetic Household Spatial Distributions.	24
3.7	Example Chileka Synthetic Household Spatial Distributions.	25
3.8	Example Ndirande Synthetic Household Spatial Distributions.	26
3.9	Age distributions for 100 generated populations at each site.	28
3.10	Household size distributions for 100 generated populations at each site.	28
3.11	Adults per household - distributions for 100 generated populations at each site.	29
3.12	Adolescents per household - distributions for 100 generated populations at each site.	29
3.13	Children per household - distributions for 100 generated populations at each site.	30
3.14	Income per household - distributions for 100 generated populations at each site.	30
4.1	The class structure for synthetic population module.	33

PART 1

Overview

1 Summary

This document contains all the information relating to the agent-based modelling of individual level antimicrobial-resistance (AMR) contamination associated with the [Drivers of Resistance in Uganda and Malawi \(DRUM\)](#) project.

1.1 Usage

The code discussed throughout this report can be found [here](#) on the CHICAS code repository. It is currently private, so appropriate credentials are required for access.

At present the model can be run through a local copy of the cloned repository. In the future, it may be developed into a package, installable through PyPI.

2 Code Structure & Workflow

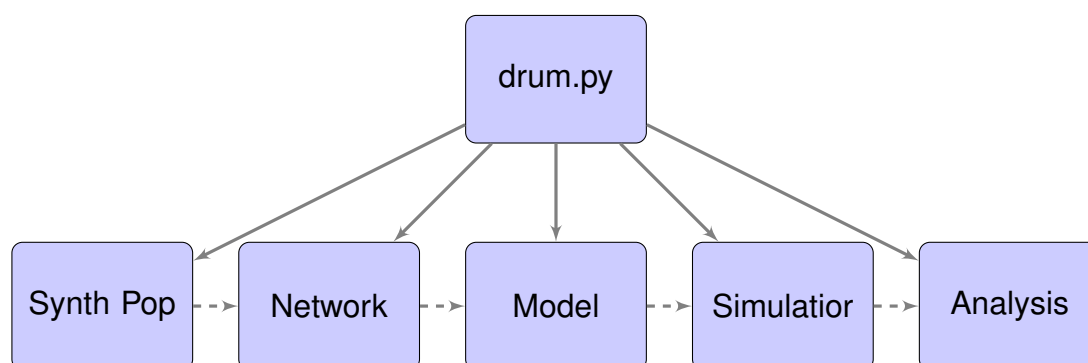


Figure 1.1: Code workflow. Solid arrows represent the execution of a module from `drum.py`, while dashed arrows represent the flow of data. Where the output of one module exists, the subsequent module in the sequence can be executed directly from `drum.py`.

The initial structure of the DRUM project code is simple (Fig. 1.1). A single main file, `drum.py`, is executed and runs one or more of the modules related to the generation of a synthetic population, a set of plausible networks, a model or simulation instance, or a set of analyses, depending on user input. This decision was made as, with growing complexity, keeping the whole project as a single module would be unmaintainable and demanding on resources. Secondly, where multiple parameter sets are to be modelled on a single network, it is unnecessary to produce the network again each time. Finally, the different modules have different requirements; both the network and analysis modules depend on capturing and presenting the high level of detail associated with an individual-based model. Conversely, the model itself needs to simulate modifications to this population as quickly as possible. As a result, the workflow will necessarily fluctuate in detail and complexity between modules.

More detail is provided in the flowchart in Fig. 1.2 which illustrates how the five modules of the DRUM ABM program are implemented, highlighting inputs and outputs. Subsequent sections discuss these inputs and processes in more detail.

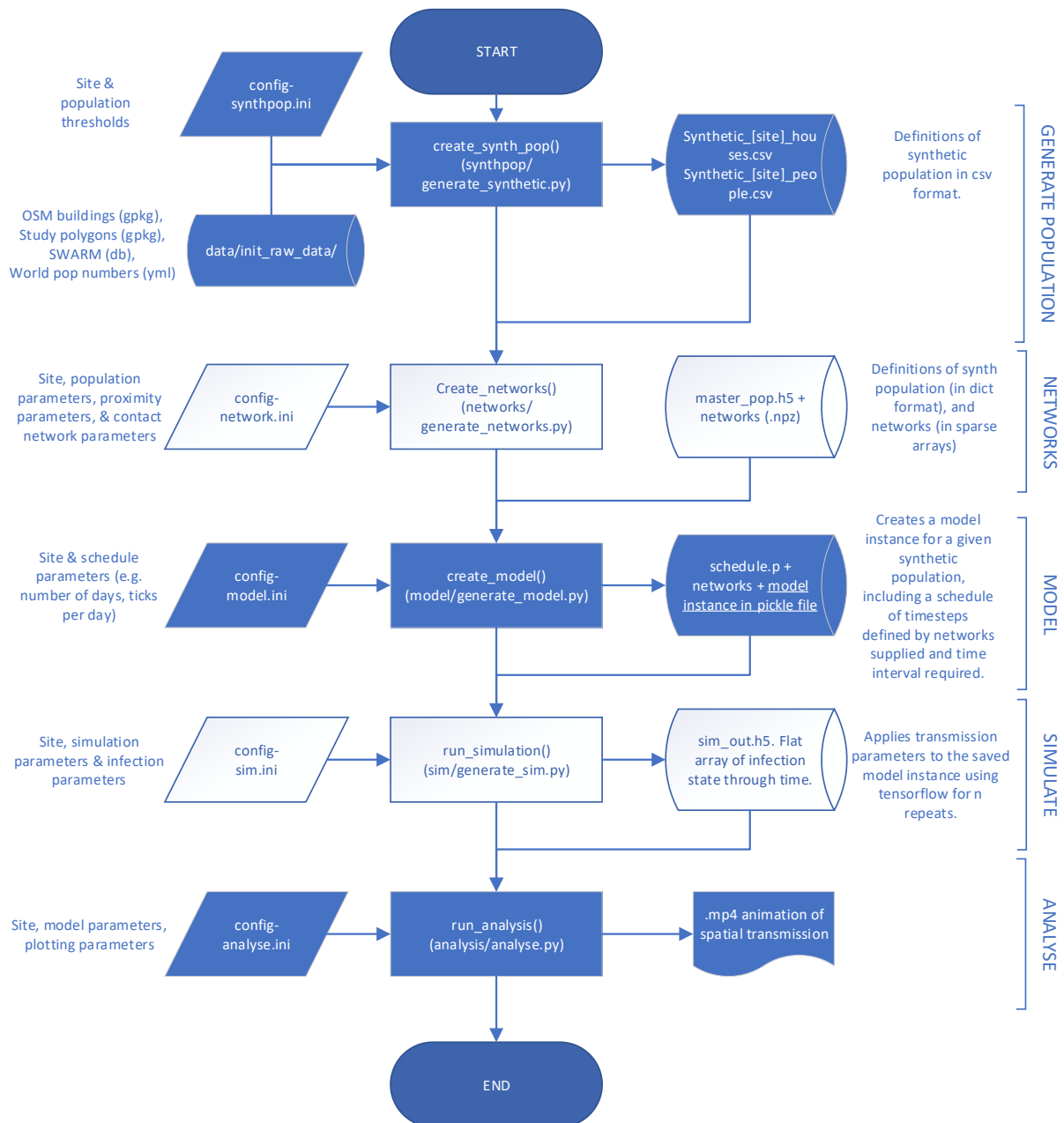


Figure 1.2: Inputs and outputs of the DRUM ABM modules.

3 Directory Tree Structure

```

drum-model
├── .git
├── data
│   ├── analysis
│   ├── init_raw_data
│   ├── models
│   ├── networks
│   ├── simulation
│   └── synthetic_population
├── docs
│   ├── doc_code
│   ├── doc_images
│   ├── doc_plots
│   └── latex_doc
├── drum
│   ├── analysis
│   ├── model
│   ├── networks
│   ├── objects
│   ├── sim
│   ├── synthpop
│   ├── utils
│   ├── config-analyse.ini
│   ├── config-master.ini
│   ├── config-model.ini
│   ├── config-network.ini
│   ├── config-sim.ini
│   ├── config-synthpop.ini
│   └── drum.py
├── envs
│   ├── drum-model_linux.yml
│   ├── drum-model_win.yml
│   └── README
├── tests
│   └── unit
├── .gitignore
└── README.md

```

The main folders in the directory tree are documented above, with more detail for each highlighted section in the following.

4 \data

```

data
├── analysis
│   └── At present, a single, site specific
│       └── .mp4 animation of the spatial spread
│           └── of contamination
├── init_raw_data
│   ├── osm_buildings
│   ├── study_polygons
│   ├── swarm.db
│   └── worldpop
├── models
│   └── Copies of 'networks' output, plus
│       └── schedule and model instance 'pickle'
│           └── files
├── networks
│   └── hdf5 file with population definition,
│       └── .npz stored networks
├── simulation
│   └── hdf5 file containing simulation output
├── synthetic_population
│   └── .csv files containing synthetic
│       └── population definitions

```

The data folder contains all of the raw data required for, and produced by, the DRUM code. The code requires/produces different elements depending on which modules are being run. For example, to produce a synthetic population, `osm_buildings`, `study_polygons`, `swarm.db` and `worldpop` data inputs are required, along with the user's choice of study site. Conversely, to run a simulation, only the contents of the model folder is required.

5 \docs

```

docs
├── doc_code
├── doc_images
├── doc_plots
└── latex_doc

```

The documentation tree contains images, example output, and the code to produce this report.

6 \drum

```

drum
├── analysis
│   ├── analyse.py
│   ├── analysis_a.py
│   └── analysis_b.py
├── model
│   ├── generate_model.py
│   └── model.py
├── networks
│   ├── generate_networks.py
│   ├── network.py
│   └── parallel_contact.py
├── objects
│   ├── household.py
│   └── individual.py
├── sim
│   ├── generate_sim.py
│   ├── simulator.py
│   └── tf_funcs.py
├── synthpop
│   └── generate_synthetic.py
├── utils
│   ├── misc_funcs.py
│   └── parse.py
├── config-analyse.ini
├── config-master.ini
├── config-model.ini
├── config-network.ini
├── config-sim.ini
├── config-synthpop.ini
└── drum.py

```

The drum folder contains a single `drum.py` file, which is used to run the whole workflow, based on the modules selected in the `config-master.ini` file. The folder also contains individual config files for each module, and seven sub-directories containing the scripts associated with each.

The analysis folder contains the `analyse.py` script, which contains the functionality to run analyses on simulation outputs (using the scripts in `analysis_a.py` and `analysis_b.py`) based on the user inputs in `config-analyse.ini`. `analysis_a.py` is used to process the simulation output, and `analysis_b.py` is used to visualise it.

The model folder contains the `generate_model.py` script which contains the functionality to build a model instance and simulation schedule, based on the user inputs in `config-model.ini`. The Model class is defined in `model.py`.

The `generate_networks.py` file in the networks folder contains the functionality to build contact networks for the synthetic population, based on user inputs defined in `config-network.ini`. A `NetworkGenerator` object (defined in `network.py`) is used to create the synthetic population objects (based on the csv files produced by the synthpop module), and build networks between individuals. Currently, this module creates:

1. A contact network based on the households.
2. A proximity network, recording the distance between each individual (N.B. distances beyond a user-defined maximum distance are ignored).

Finally, the `parallel_contact.py` file contains the standalone functionality to create a network for a subset of the population for a given activity, allowing us to build the networks using multiprocessing.

The `objects` folder contains the object definitions for the population. `individual.py` defines `Individual` and `Child` classes, and `household.py` defines the `Household` class.

The `generate_sim.py` in the `sim` folder contains the functionality to run a simulation based on the user inputs defined in `config-sim.ini` and a model instance. The `Simulator` class is defined in `simulator.py`, and supporting tensorflow functions in `tf_funcs.py`.

The functionality to create a synthetic population based on the data held in `data/init_raw_data/` is held in `synthpop/generate_synthetic.py`.

Finally, a `utils` folder contains any supporting functions that are general for the whole DRUM model. `misc_funcs.py` contains useful data manipulation functions, and `parse.py` contains a function specific to parsing data from config files.

7 \envs

```
envs
├── drum-model_linux.yml
├── drum-model_win.yml
└── README
```

This folder contains exported conda environments for both windows and linux, which contain prerequisites to run the model. There is also a `README` with instructions for installation.

8 \tests

```
tests
├── unit
```

At present the tests folder is empty.

PART 2

Data

1 Summary

The drum ABM takes four input datasets to allow the generation of a synthetic population, and subsequently the demographic networks, as inputs to the model itself.

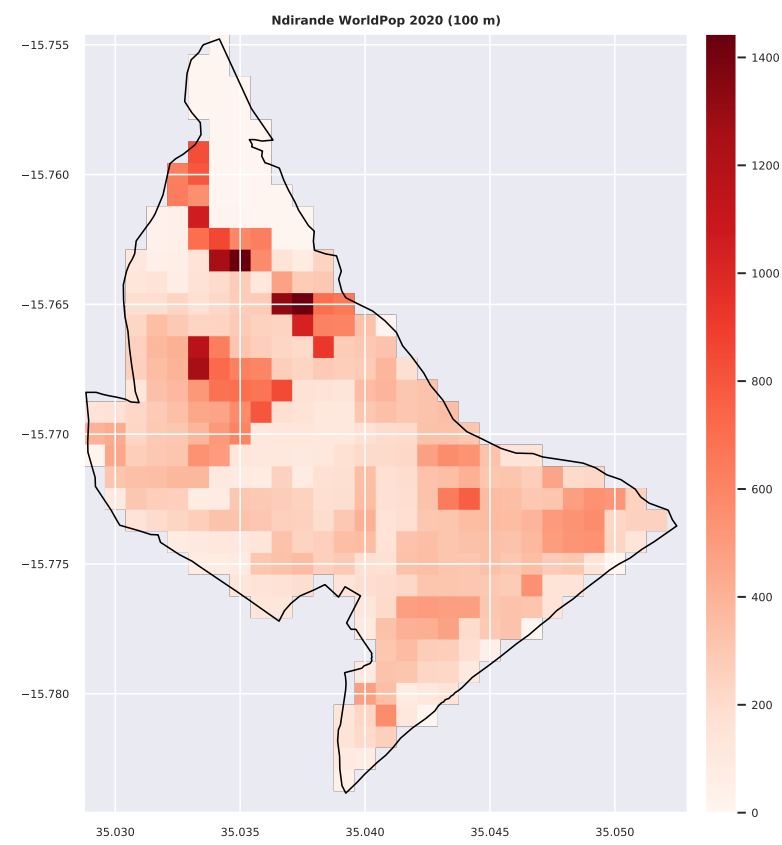
The directories for these inputs, which can be found in `data/init_raw_data/`, are:

- `osm_buildings`: A directory containing a `.gpkg` file with OSM buildings for each of the three study sites, filtered to those which could be considered residential based on OSM labels. The directory also contains a `.yaml` file with the number of filtered buildings in each site, and spatial plots of these buildings.
- `study_polygons`: A directory containing a `.gpkg` of the study defined polygons for Ndirande, Chileka, and Chikwawa.
- `swarmdb`: A copy of the latest `swarm.db` database, from which DRUM collected data can be incorporated into the model.
- `worldpop`: A directory containing raw 2020, 100 m resolution WorldPop `.tif` data, clipped to the three study sites. The directory also contains a `.yaml` file with the estimated total population of each site, and image representations of the data.

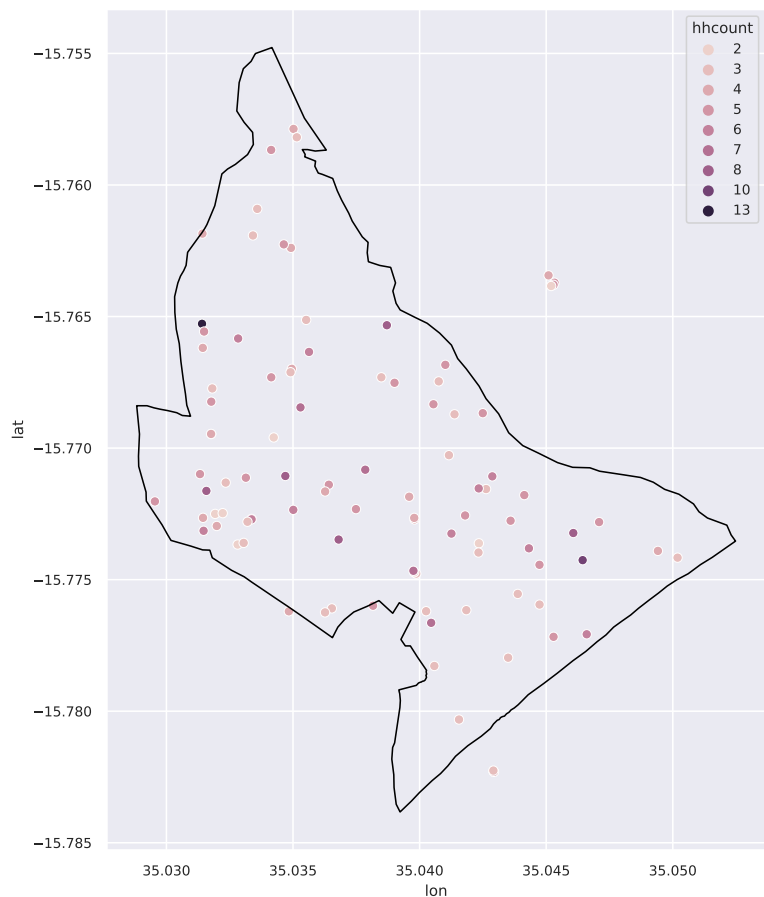
2 Coverage

Figures 2.1, 2.2, 2.3 illustrate this data for the Ndirande, Chileka and Chikwawa sites respectively.

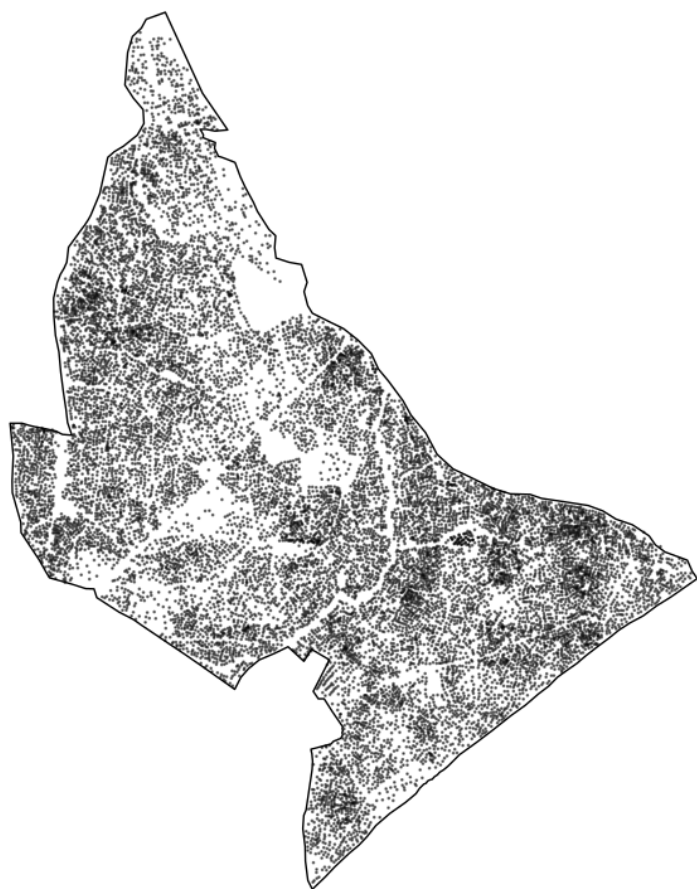
It is worth noting that there appears to be good, if not complete, coverage of buildings in the Ndirande and Chileka datasets, based purely on qualitative assessment. However, the Chikwawa OSM dataset is more ‘patchy’. 714 buildings would appear to be an under-representation of the current number, particularly when comparing to the WorldPop estimated population distribution, and the lack of polygons in the south-east of the study area for example. It looks like the digitising of buildings here has been more targeted, and less consistent. The majority of data available to date appears to have been collected in 2014, which would again suggest the dataset is incomplete.



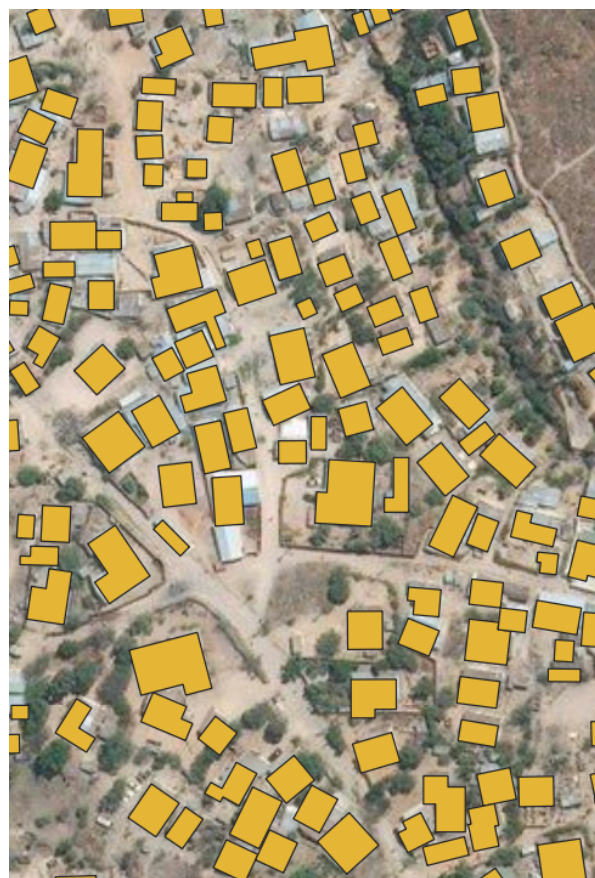
(a) WorldPop population estimate. $N = 108,323$.



(b) SWARM household sample. 98 houses, 304 people located.

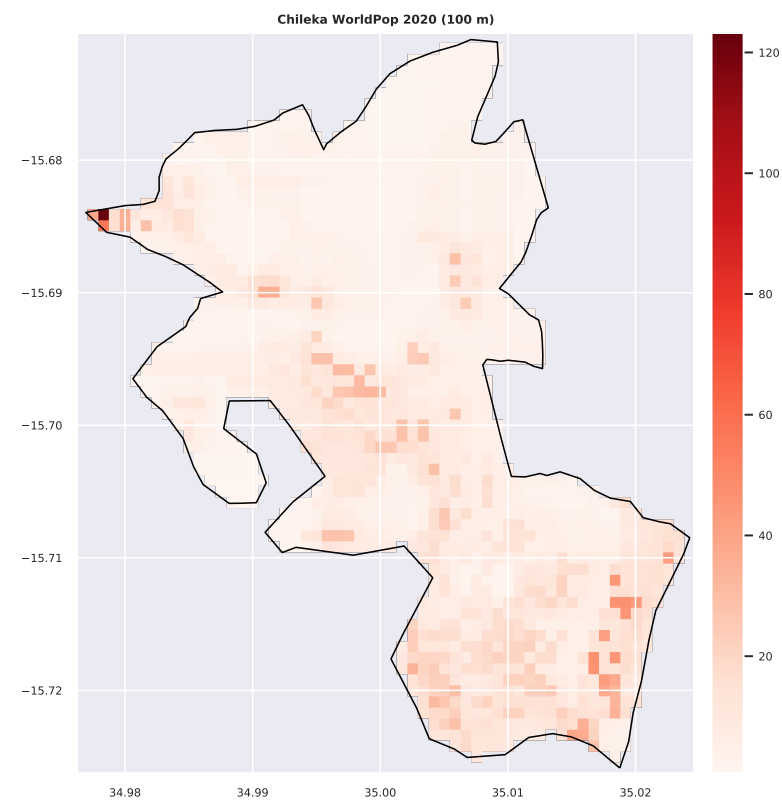


(c) Distribution of filtered OSM buildings. $N = 15,722$.

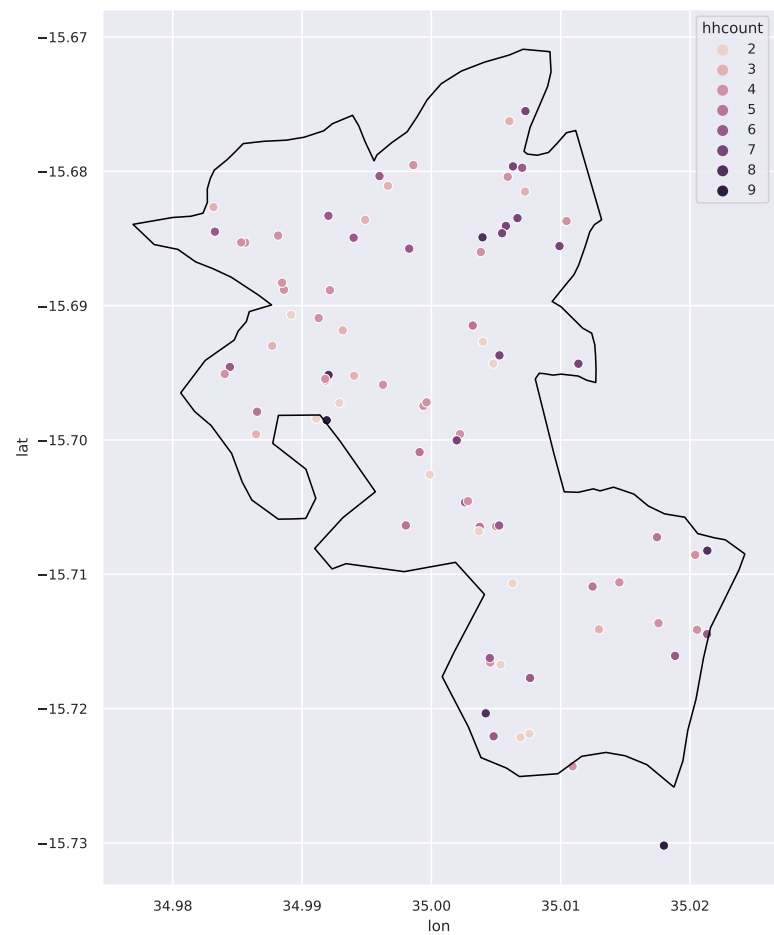


(d) Zoom of the OSM data coverage quality.

Figure 2.1: Ndirande population/household location information.



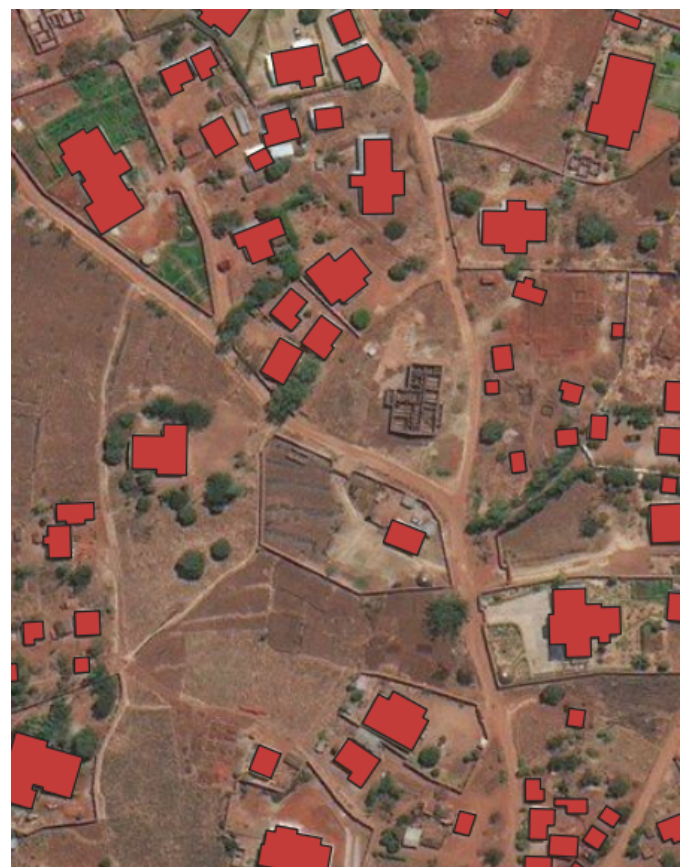
(a) WorldPop population estimate. N = 13,184.



(b) SWARM household sample. 84 houses, 323 people located.

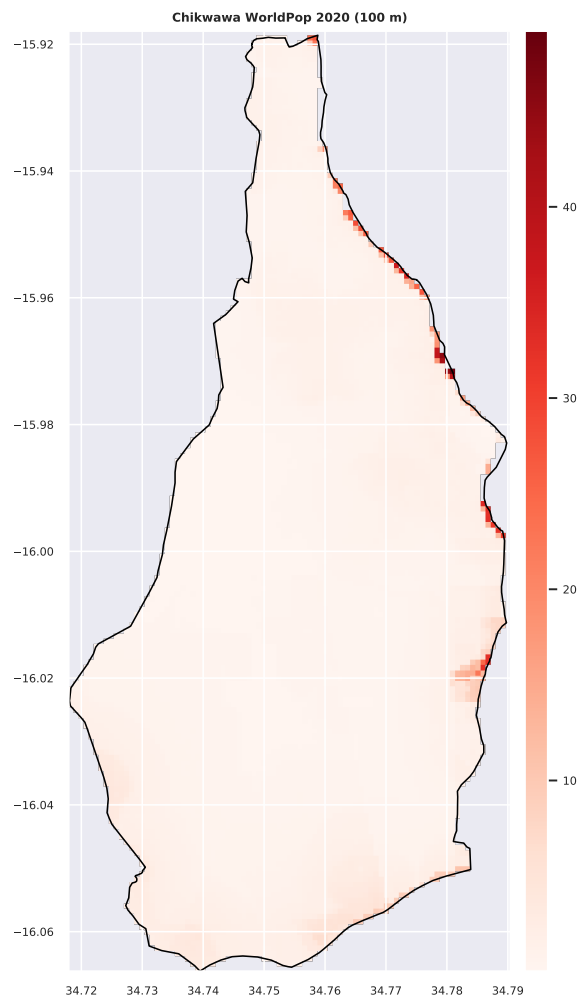


(c) Distribution of filtered OSM buildings. N = 7,902.

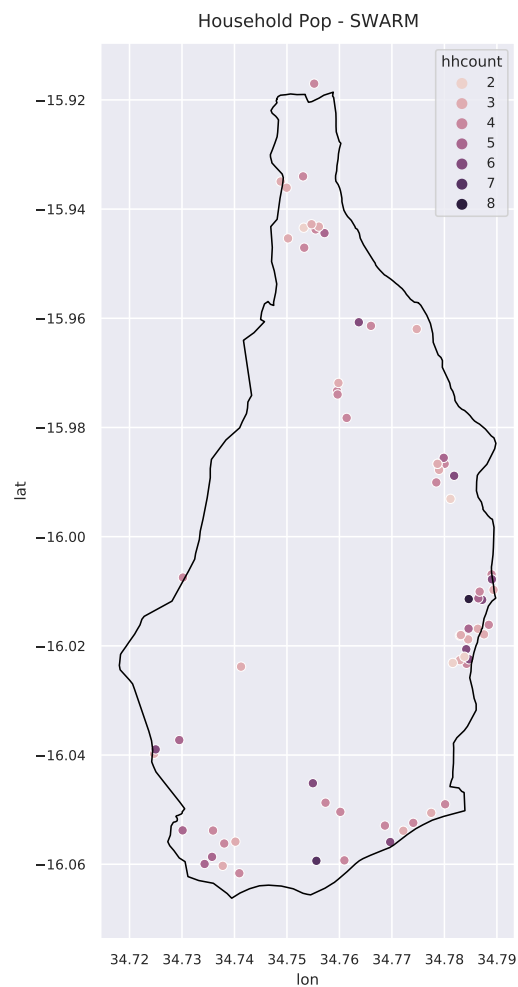


(d) Zoom of the OSM data coverage quality.

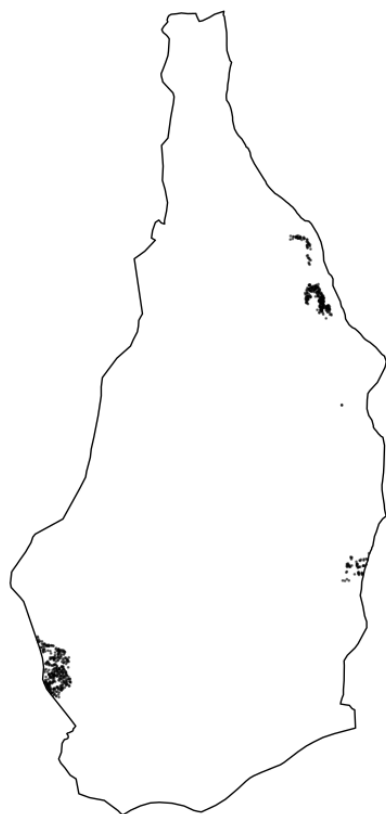
Figure 2.2: Chileka population/household location information.



(a) WorldPop population estimate. N = 9,509.



(b) SWARM household sample. 70 houses, 178 people located.



(c) Distribution of filtered OSM buildings. N = 714.



(d) Zoom of the OSM data coverage quality.

Figure 2.3: Chikwawa population/household location information.

PART 3

Synthetic Population

1 Initial Method

The initial method presented here is to produce a population for each study area from the SWARM household sample, with a size that corresponds to the WorldPop total estimated population for the site (if not the estimated distribution). Household locations are derived from the STRATAA census (Ndirande), OSM house data (Chileka), or re-sampled from the SWARM sample itself, with jitter (Chileka). SWARM households will be sampled with replacement, while households with missing people will source extra household members of the correct age category from households with the most similar household characteristics.

This method is described below, followed by an analysis of output, and discussion of issues.

2 Config File

The synthpop module requires a config file with the following parameters defined:

- *chikwawa* [Boolean]: Flag identifying whether to build the synthetic population for the Chikwawa study site.
- *chileka* [Boolean]: Flag identifying whether to build the synthetic population for the Chileka study site.
- *ndirande* [Boolean]: Flag identifying whether to build the synthetic population for the Ndirande study site.
- *chikwawa_pop* [Integer]: Population size to build for the Chikwawa study site (Currently maps to WorldPop estimates).
- *chileka_pop* [Integer]: Population size to build for the Chileka study site (Currently maps to WorldPop estimates).
- *ndirande_pop* [Integer]: Population size to build for the Ndirande study site (Currently maps to WorldPop estimates).
- *osm_building_size_filter_min* [Integer]: Minimum polygon area permitted for OSM household selection.

- *osm_building_size_filter_max* [Integer]: Maximum polygon area permitted for OSM household selection.
- *building_jitter* [Integer]: Maximum displacement for jitter application in metres.

3 Method

The workflow is as follows:

- **Load SWARM individual and household data:** Access the `swarm.db` database, loading currently locatable households and individuals into two pandas dataframes. Filter out column attributes that either aren't relevant, or provide insufficient/too much variability to be immediately useful in the model.
- **Parse config:** Parse the config file, identify which sites a population is being created for, and how big it will be, and initialise a for-loop through the subsequent functions, for each necessary site.
- **Filter loaded SWARM data:** Return filtered versions of the pandas household and individual dataframes, based on relevant study site.
- **Load buildings:** Load the buildings data that corresponds to the relevant study site.
- **Fill buildings:** Draw a household from the SWARM data (random, with replacement), and associate with a drawn building polygon (random, without replacement). When OSM buildings are exhausted, start drawing buildings from the growing synthetic population and apply a jitter to locate (currently hardcoded to be within 25 m in any direction). Continue until the threshold population is reached. Export these synthetic households as a `.csv`.
- **Fill 'complete' households:** For each synthetic household where the SWARM household population count matches the number of individuals sampled in the SWARM individual dataset (i.e. every household member was surveyed), add the inhabitants to the synthetic population.
- **Fill 'incomplete' households:** For each synthetic household where the SWARM household population count is greater than the number of individuals sampled in the SWARM individual dataset (i.e. every household member was surveyed), identify how many adults, children and adolescents are missing, and copy them from 'similar' households. Similarity is scored by comparing household-level responses in the dataset. Export these synthetic people as a `.csv`.

4 Analysis

4.1 Ndirande STRATAA vs. Ndirande synthetic.

Figures 3.1 to 3.5 show data for Ndirande. Distributions are compared between STRATAA (an Ndirande census study), the SWARM sample, and an instance of a synthesised population.

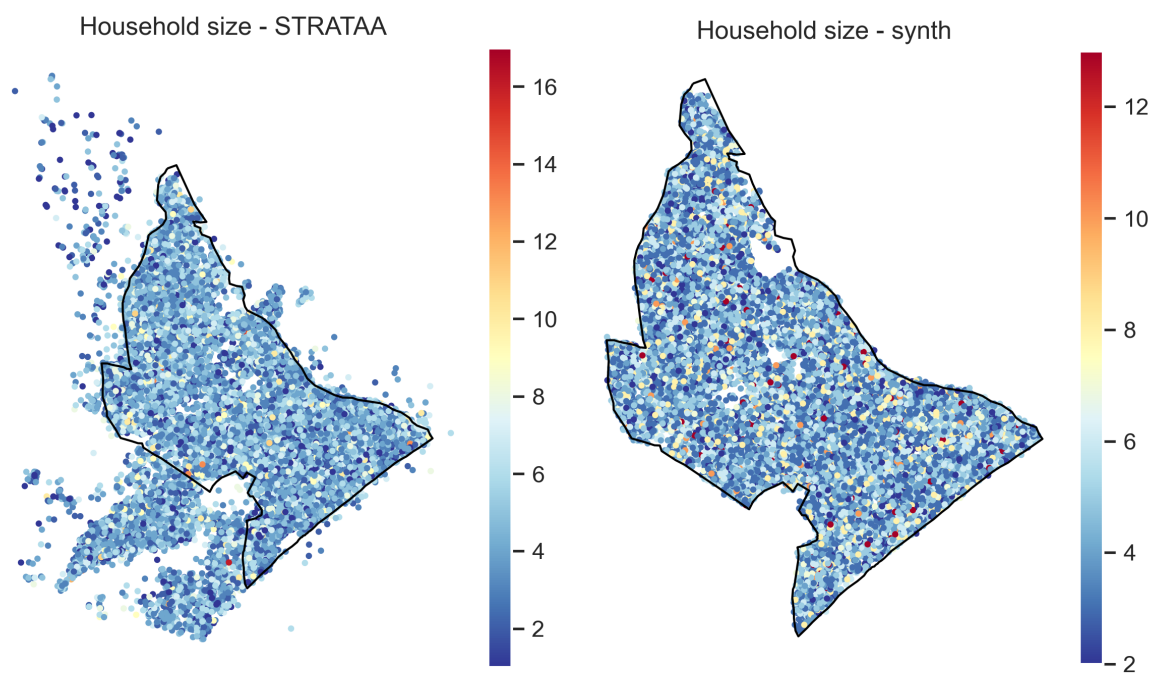


Figure 3.1: STRATAA census vs. synthetic population, coloured by household size.

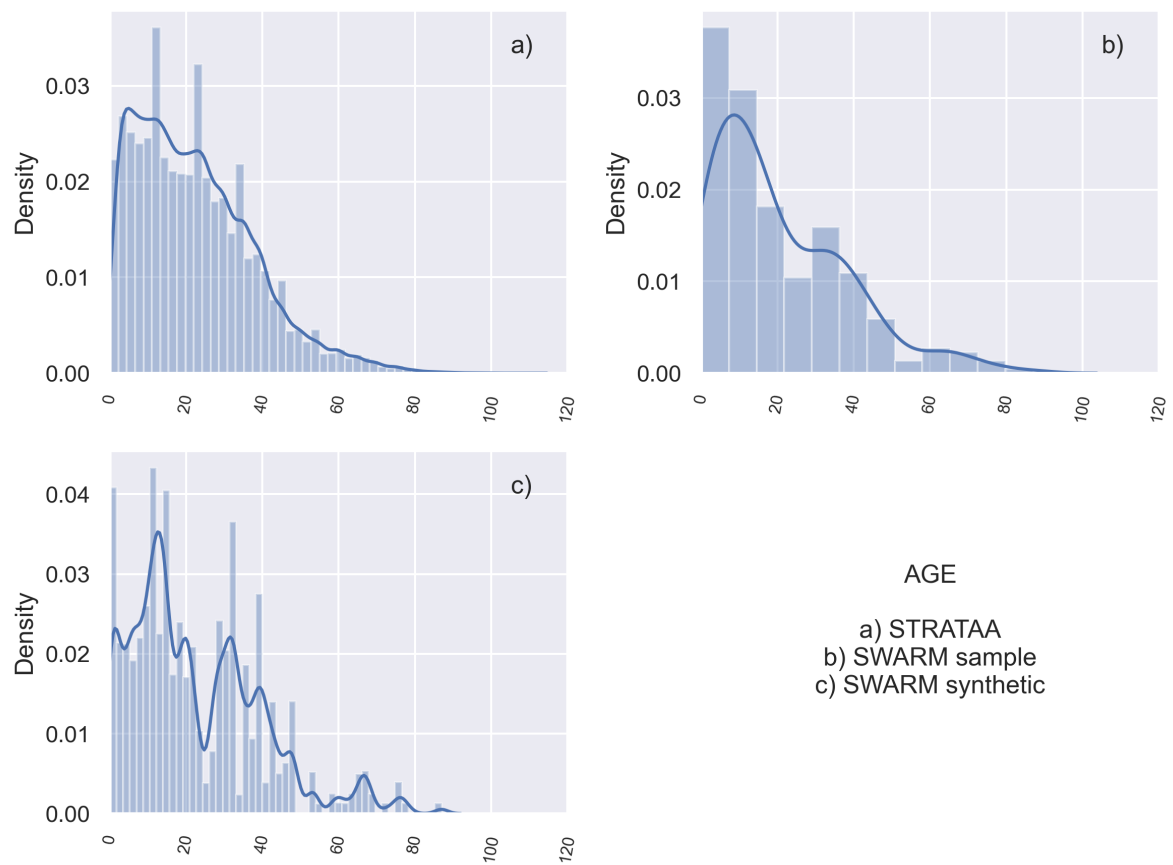


Figure 3.2: Age distributions for STRATAA, the Ndirande SWARM sample, and the Ndirande synthetic population.

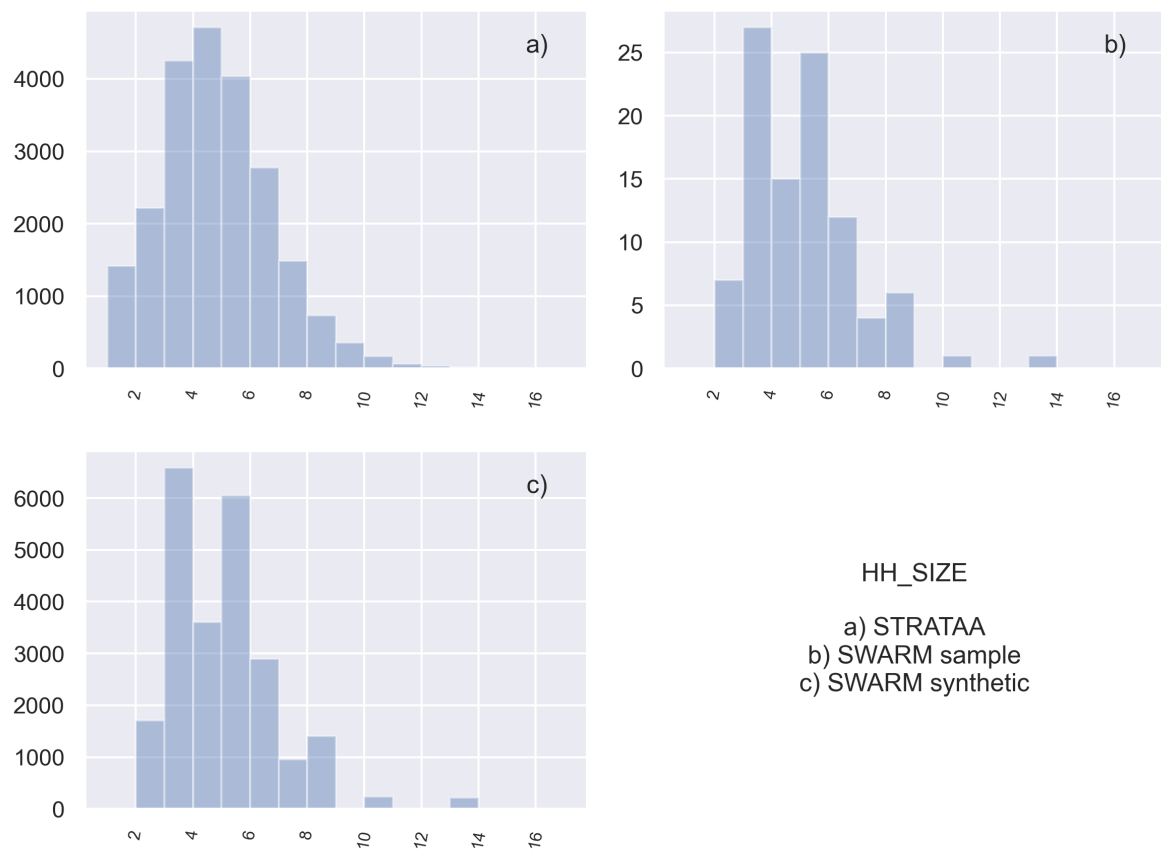


Figure 3.3: Household size distributions for STRATAA, the Ndirande SWARM sample, and the Ndirande synthetic population.

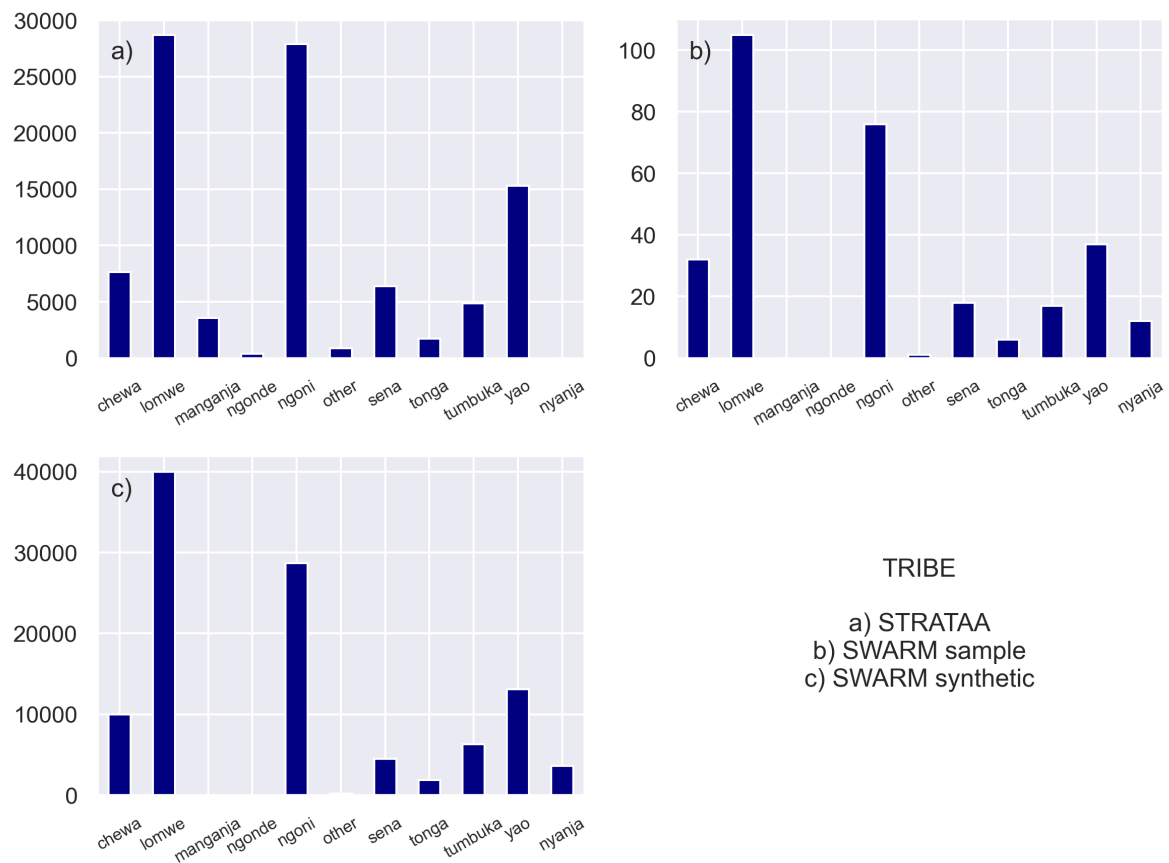


Figure 3.4: Ethnicity distributions for STRATAA, the Ndirande SWARM sample, and the Ndirande synthetic population.

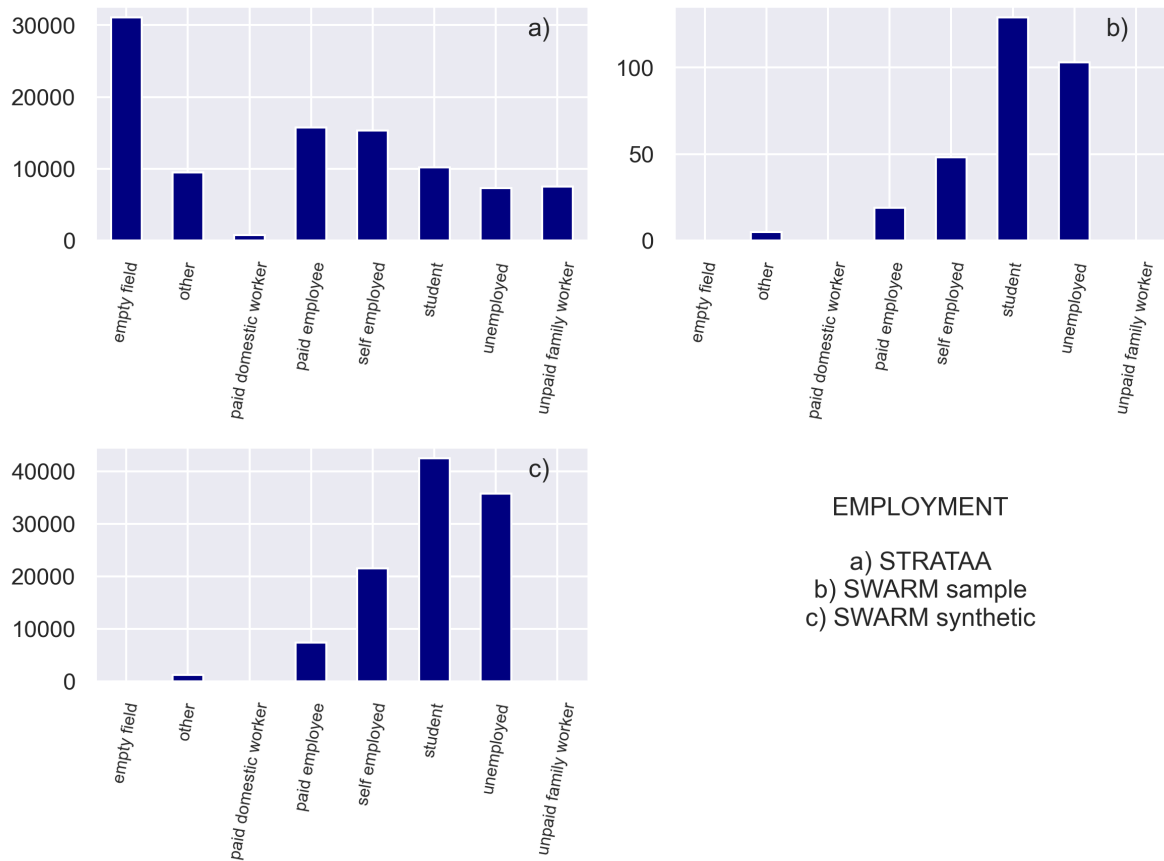


Figure 3.5: Employment distributions for STRATAA, the Ndirande SWARM sample, and the Ndirande synthetic population. **N.B. STRATAA ‘empty-fields’ largely represent respondents under the age of 15. It is expected that these would translate to additional numbers in the ‘student’ or ‘unemployed’ categories.**

4.2 Example Synthetic Household Spatial Distributions.

Figures 3.6 to 3.8 illustrate the spatial distributions of households in a single synthetic population generated for each study site.

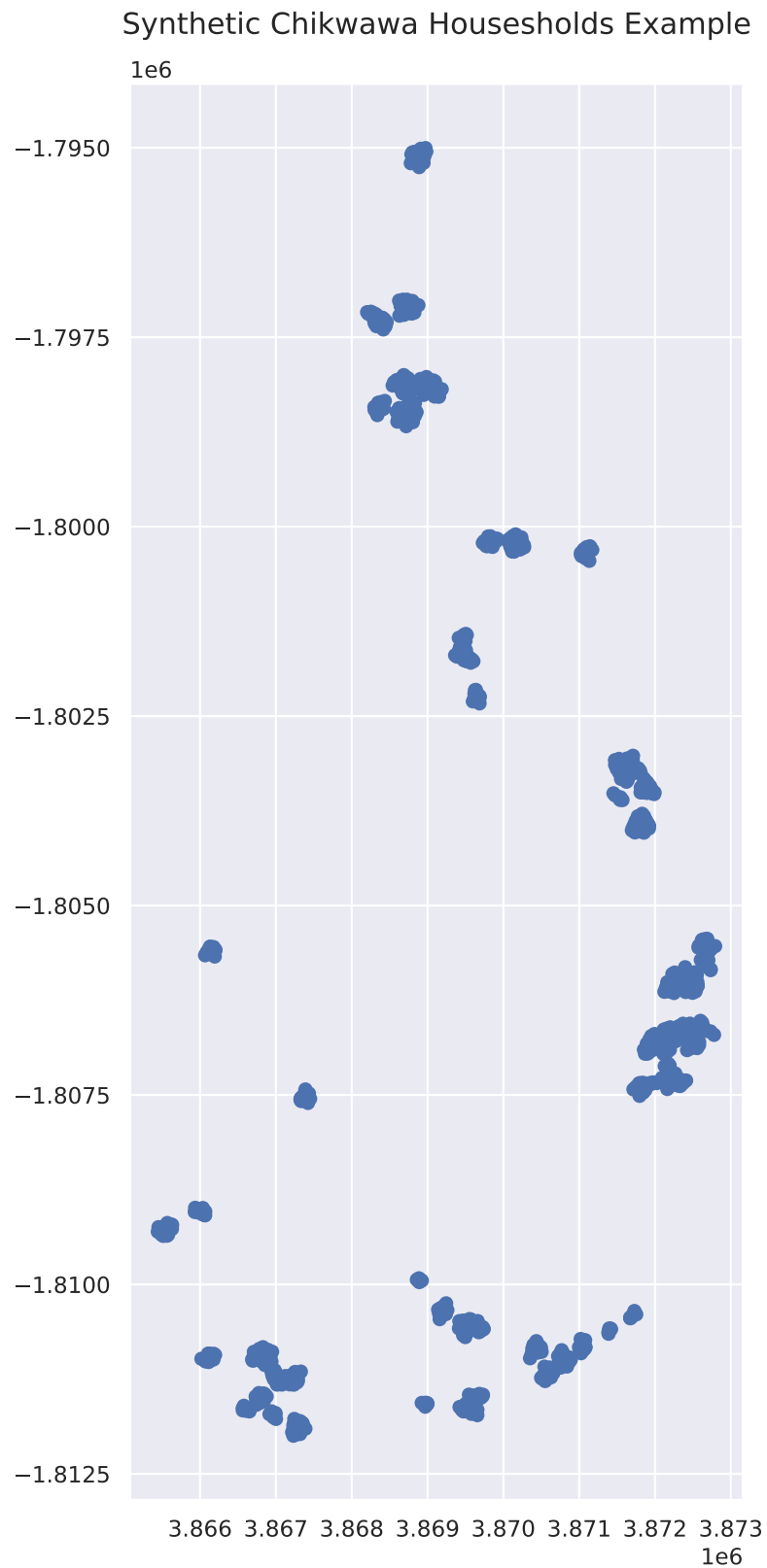


Figure 3.6: Example Chikwawa Synthetic Household Spatial Distributions.

Synthetic Chileka Households Example

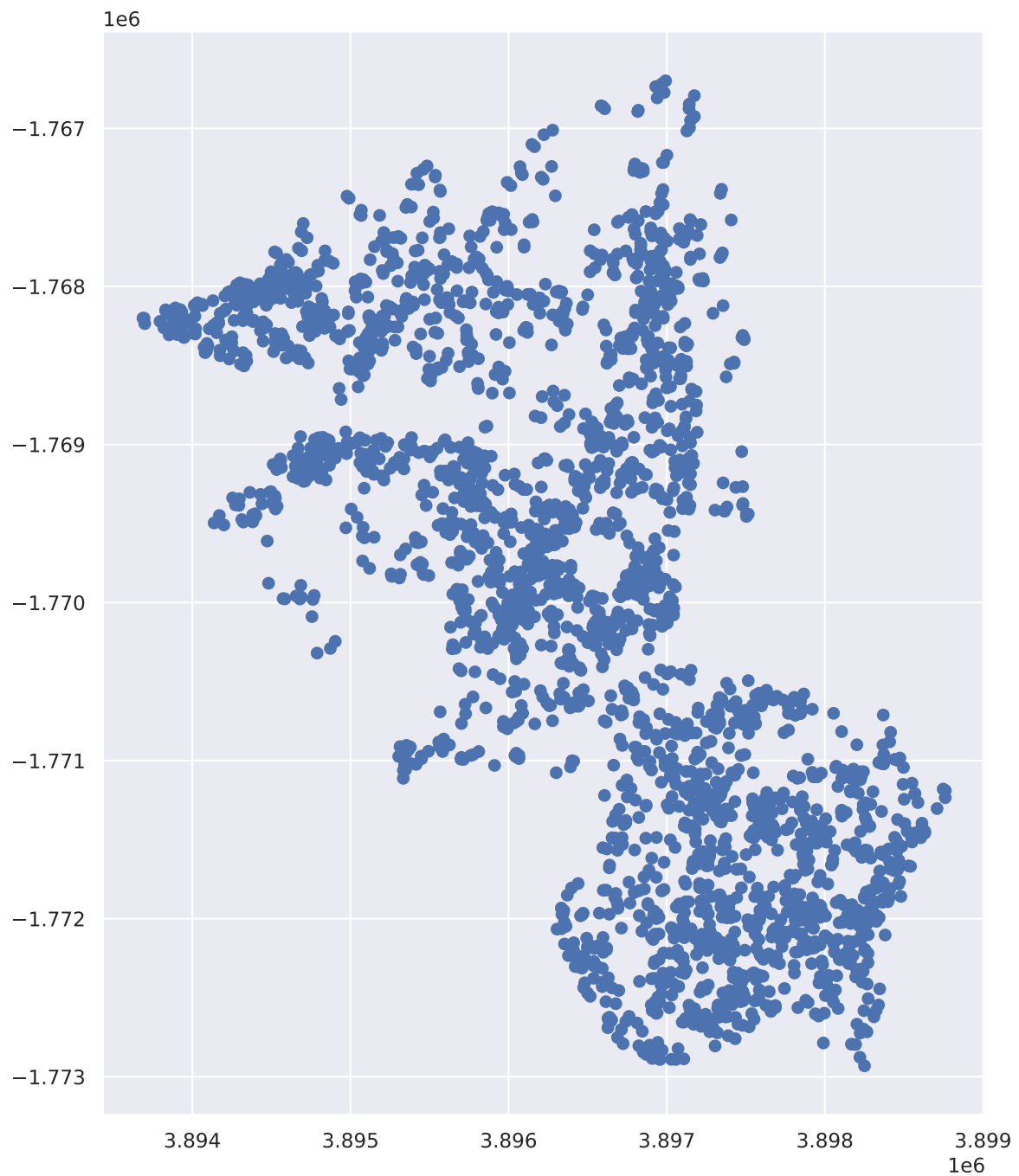


Figure 3.7: Example Chileka Synthetic Household Spatial Distributions.

Synthetic Ndirande Households Example

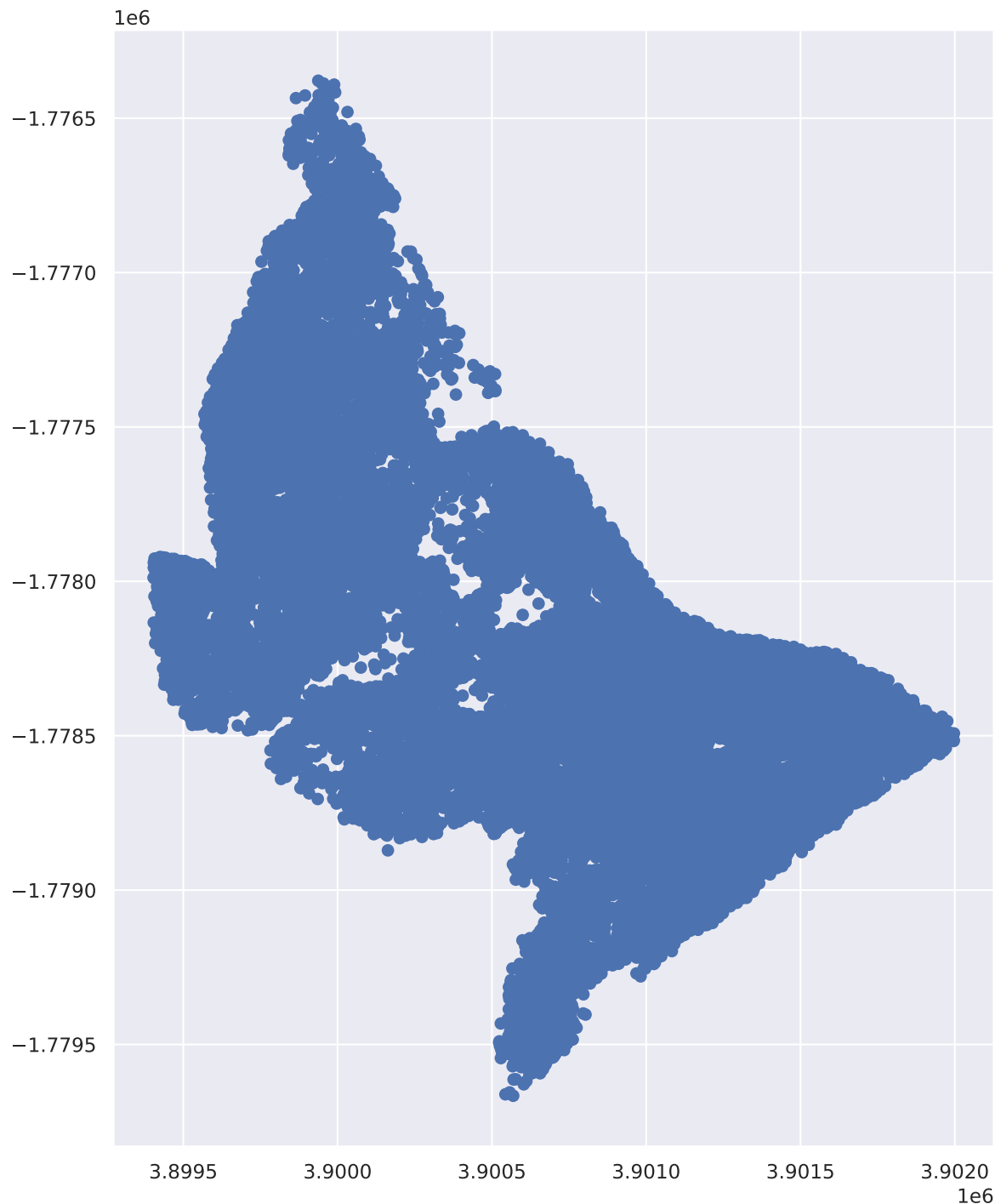
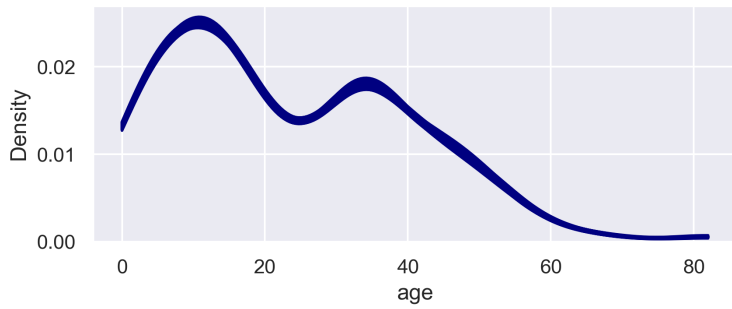


Figure 3.8: Example Ndirande Synthetic Household Spatial Distributions.

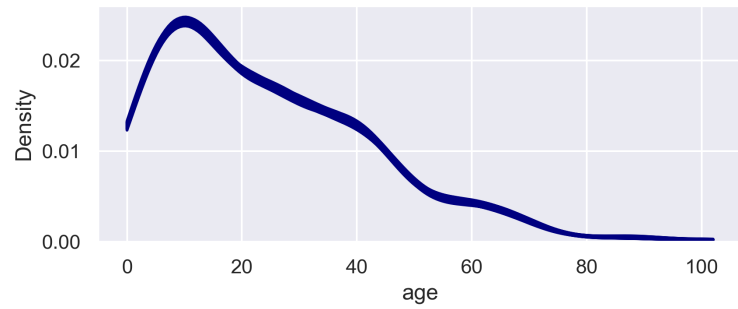
4.3 Generating 100 populations for each site.

Figures 3.9 to 3.14 show the distributions (plotted on top of each other) of a set of numeric parameters from 100 generated synthetic populations at each site.

age chikwawa - KDE for 100 generated populations.



age chileka - KDE for 100 generated populations.



age ndirande - KDE for 100 generated populations.

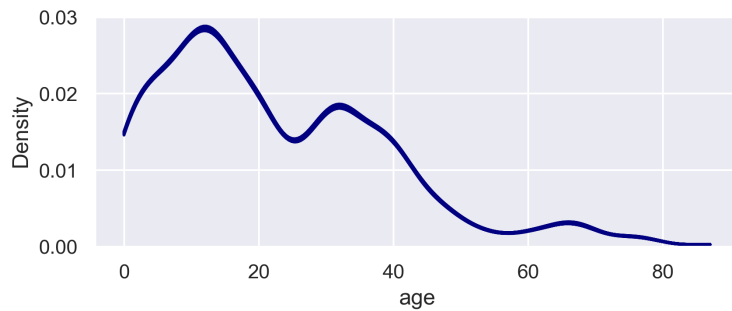
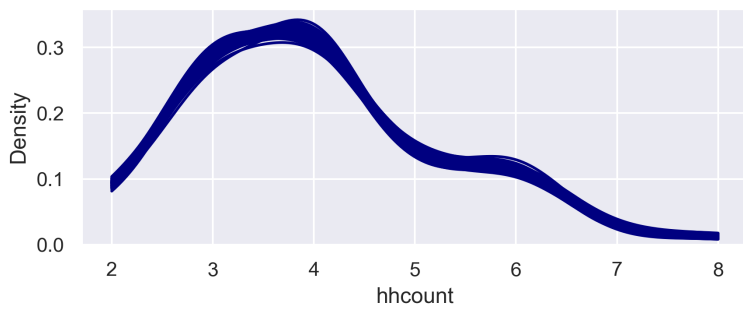
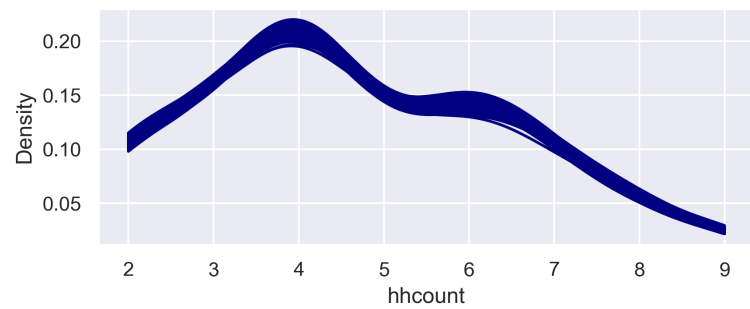


Figure 3.9: Age distributions for 100 generated populations at each site.

hhcount chikwawa - KDE for 100 generated populations.



hhcount chileka - KDE for 100 generated populations.



hhcount ndirande - KDE for 100 generated populations.

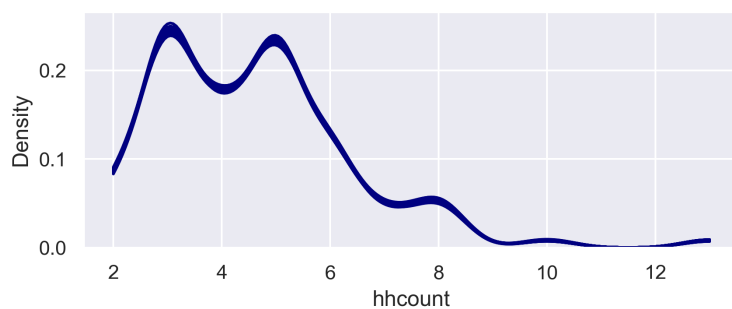
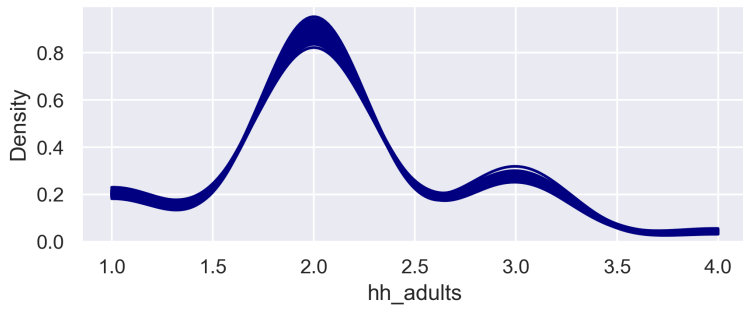
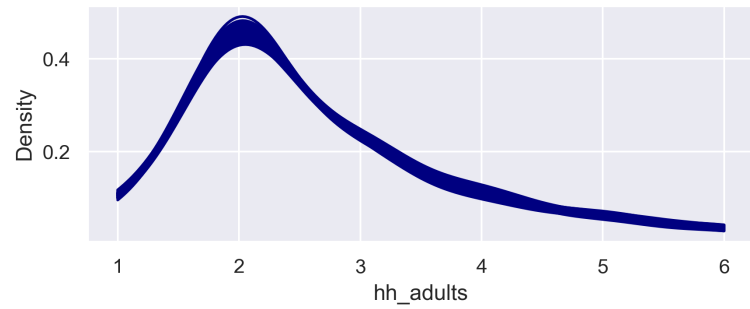


Figure 3.10: Household size distributions for 100 generated populations at each site.

hh_adults chikwawa - KDE for 100 generated populations.



hh_adults chileka - KDE for 100 generated populations.



hh_adults ndirande - KDE for 100 generated populations.

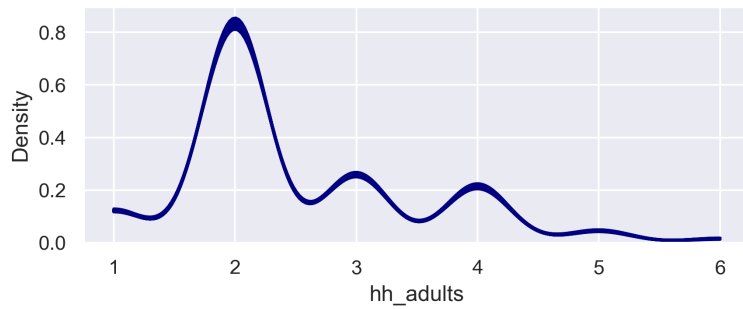
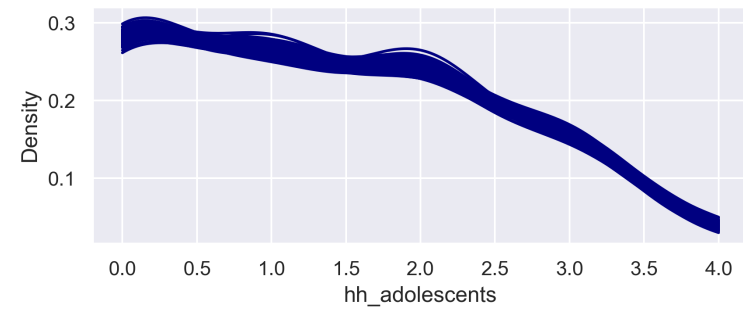
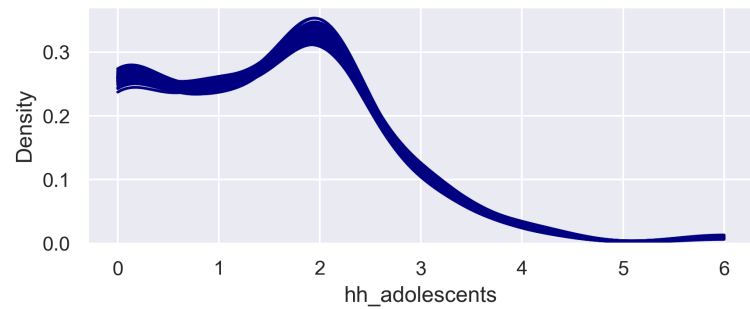


Figure 3.11: Adults per household - distributions for 100 generated populations at each site.

hh_adolescents chikwawa - KDE for 100 generated populations.



hh_adolescents chileka - KDE for 100 generated populations.



hh_adolescents ndirande - KDE for 100 generated populations.

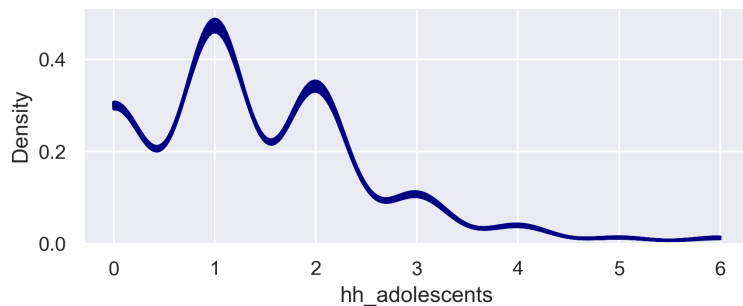
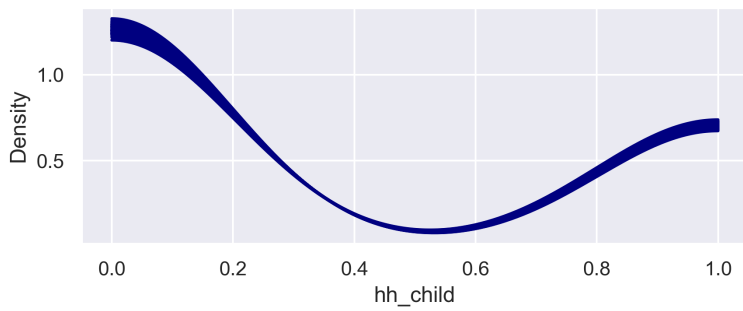
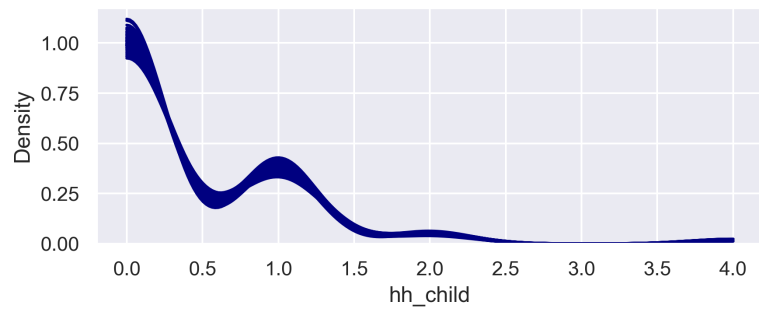


Figure 3.12: Adolescents per household - distributions for 100 generated populations at each site.

hh_child chikwawa - KDE for 100 generated populations.



hh_child chileka - KDE for 100 generated populations.



hh_child ndirande - KDE for 100 generated populations.

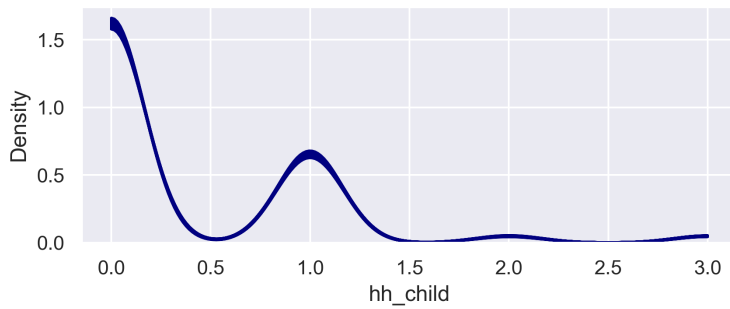
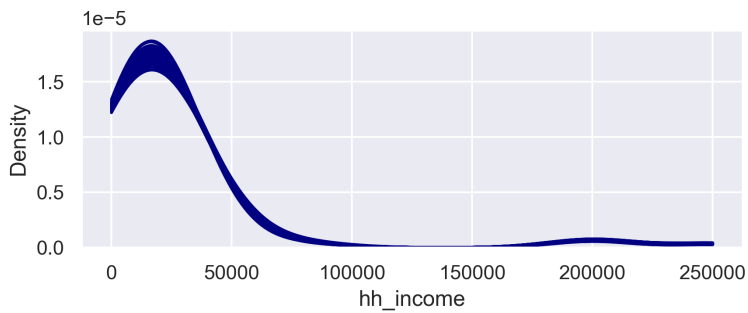
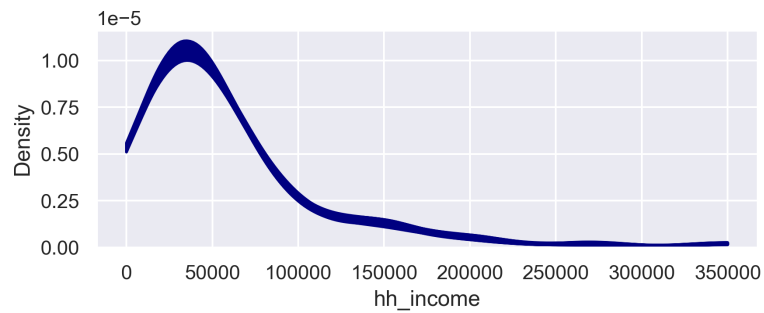


Figure 3.13: Children per household - distributions for 100 generated populations at each site.

hh_income chikwawa - KDE for 100 generated populations.



hh_income chileka - KDE for 100 generated populations.



hh_income ndirande - KDE for 100 generated populations.

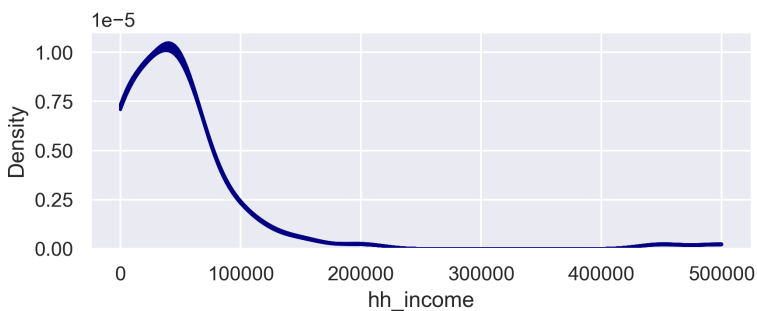


Figure 3.14: Income per household - distributions for 100 generated populations at each site.

5 Issues/Review

Things to review when updating the methodology:

- Not all original OSM buildings are populated in Chileka. **Suggests under-estimation of population size by WorldPop, and/or an insufficiently filtered OSM dataset. If this is the more affluent area, there may be more polygons per household (e.g. garages etc.)?**
- Random assignment assumes no demographic/economic spatial patterns between sites. STRATAA data suggests this may be the case for Ndirande (apart from a possible pattern relating to university education), but difficult to tell in other sites.

PART 4

Networks

1 Summary

An agent-based synthetic population is first generated based on the csv files generated in the synthpop module. To capture as much information as possible, individual agents are used for each person in the population and households in the area. Future development will involve adding objects to represent destinations in the area.

This module then builds contact networks based on the synthetic population.

2 Class Structure

The class structure for the synthetic population is presented in the UML diagram in Figure 4.1.

3 Config File

The networks module requires a config file with the following parameters defined:

- *chikwawa* [Boolean]: Flag identifying whether to build the networks for the Chikwawa study site.
- *chileka* [Boolean]: Flag identifying whether to build the networks for the Chileka study site.
- *ndirande* [Boolean]: Flag identifying whether to build the networks for the Ndirande study site.
- *child_age* [Integer]: Individuals below this age are defined as Child objects in the model.
- *max_dist* [Integer]: The maximum distance between individuals to be captured in the proximity network.
- *num_chunks_proximity* [Integer]: The number of chunks to break the proximity matrix into when parallelising calculations.
- *num_chunks_contact* [Integer]: The number of chunks to break the contact matrices into when parallelising calculations.

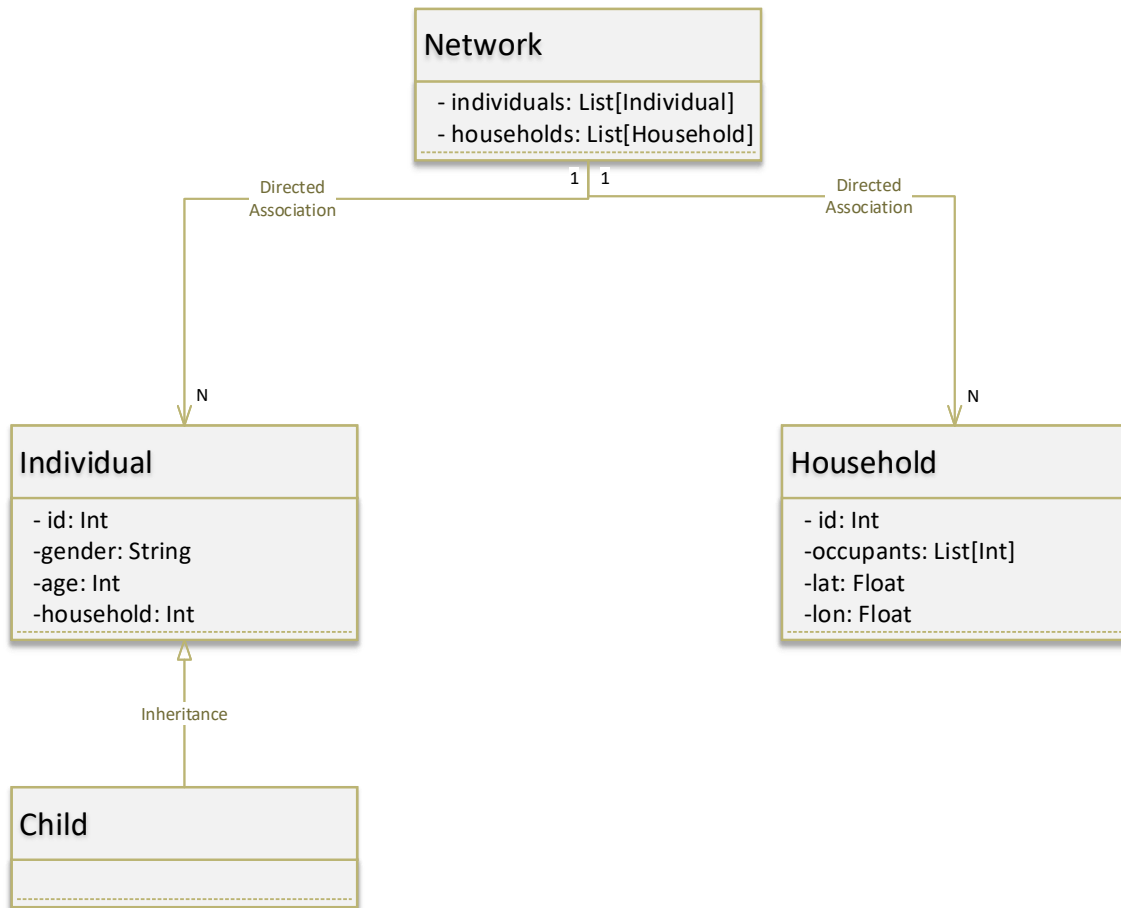


Figure 4.1: The class structure for synthetic population module.

4 Network Generation

The generation of networks is carried out in the following stages:

- **Generate households:** A list of households is created using the attributes in the `synthetic_[site]_households.csv` file generated as part of the `synthpop` module.
- **Generate base population:** A base population is created using the age/sex/-household ID parameters in the `synthetic_[site]_people.csv` file generated as part of the `synthpop` module. Any individual aged under `child_age` is a 'Child', and over this age is an 'Individual'.
- **Assign households:** Individuals are then linked with the households, by adding their ID to a household's list of occupants. The household is selected based on a matching ID to the household attribute the individual has been assigned.
- **Create networks:** Sparse matrices are used to capture the social networks that are formed based on the users of different destinations:
 - Households.
 - *Additional destinations will be added in future development of the model.*

In addition to the destination networks, a proximity network is generated which provides the distance (where the distance is less than `max_dist` metres) between individuals in the population. These networks are saved as `.npz` files.

- **Save to h5 file:** A hdf5 file is created, which stores the definition of population demographics and households.

PART 5

Model

1 Summary

The model module sets up the DRUM model. Key functions that occur in this module are:

- Update and merge networks to reflect different activities that may take place each day (This is currently placeholder functionality).
- Create a schedule that defines which networks will be active at each time step of the model.

Once these functions have all been completed, the model instance can be passed to the initialisation of a simulator object.

2 Config File

The model module requires a config file with the following parameters defined:

- *chikwawa* [Boolean]: Flag identifying whether to build the model for the Chikwawa study site.
- *chileka* [Boolean]: Flag identifying whether to build the model for the Chileka study site.
- *ndirande* [Boolean]: Flag identifying whether to build the model for the Ndirande study site.
- *num_days* [Integer]: The number of days for simulation.
- *ticks_per_day* [Integer]: The number of timesteps simulated per day.
- *model_dir* [Integer]: The sub-directory name to save the model outputs to. The full folder path will be data/models/[site]/[model_dir]/.

3 Merge networks

To model the agents' connections at different times of day, some of the networks may need to be merged, however since the current version only captures household contacts no merging is carried out.

All of the networks generated by this function are saved as .npz files in the `model_dir` directory.

4 Schedule

The final stage of the model set-up is generation of a schedule. A dictionary is created that defines which network should be modelled for each tick in a simulation. The following assumptions are used:

- Once every day, the proximity network is used to calculate new infections.
- Every tick of the simulation, irrespective of time of day or day of week, the household network is used to model contacts.

N.B. As the complexity of the model increases, this schedule will capture more detail. The schedule is saved in 'pickle' format in the `model_dir` directory. Finally, an instance of the model is also stored in the `model_dir` directory,

PART 6

Simulator

1 Summary

The simulator object runs the transmission models for the schedule and networks that have been constructed previously, using an array approach.

2 Config File

The model module requires a config file with the following parameters defined:

- *chikwawa* [Boolean]: Flag identifying whether to run the simulation for the Chikwawa study site.
- *chileka* [Boolean]: Flag identifying whether to run the simulation for the Chileka study site.
- *ndirande* [Boolean]: Flag identifying whether to run the simulation for the Ndirande study site.
- *model_dir* [Integer]: The sub-directory name to read the model inputs from, and save the simulation outputs to. The full folder paths will be `data/model/[site]/[model_dir]/` and `data/simulation/[site]/[model_dir]/`.
- *num_infected* [Integer]: Number of infections to seed the simulation with.
- *num_runs* [Integer]: The number of repeats to run the simulation for.
- *latency_mean* [Integer]: The mean number of days that an individual could be in latent state.
- *duration_mean* [Integer]: The mean number of days that an individual could spend in the infected state.
- *net_rate* [Float]: The network transmission rate.
- *forcing_rate* [Float]: The 'community forcing' transmission rate.
- *forcing_delta* [Float]: The 'delta' factor to apply to the proximity network, allowing us to calculate a community forcing network.

3 Initialise

On initialisation, the following functions are performed:

- The networks are all loaded, and a ‘community forcing’ network is calculated using the following decay function: $\delta / (\delta^2 + d_{ij}^2)^{1.5} * r$, where δ is the forcing_delta parameter, d is the proximity network, and r is the community forcing transmission rate. Networks are stored in a list of networks.
- The schedule is loaded.
- The simulation arrays are set up:
 - State: a named tuple that contains arrays, of length number of individuals, for each state in the transmission model (susceptible, latent, infectious, recovered, and the network id that resulted in an infection). For each array, 0 is used to represent individuals that are not in that state, and 1 is used to represent individuals that are. At this point, initial infections are applied.
 - Event Cache: a named tuple that contains arrays, of length number of individuals, which define the duration of latency and recovery periods. These are set to be random numbers within the provided ranges.
 - Network Select: An array that represents the index of the network that is active at each tick of the simulation.
 - Start of Day: An array that identifies whether each tick in the simulation represents the start of a day.

4 Iterate

The iterate function is repeatedly called for the number of runs defined on initialise.

4.1 Seed infected population

A random selection of the population is selected to be the initial infections for each individual run of the simulation, based on the number of infections defined on initialisation. Before returning the IDs of the infected agents, the function checks that the selected agents have more than two others in their household.

Individuals are then assigned a latency and recovery period (drawn from a geometric distribution using the means provided on initialisation). Any individuals that have been selected as initially infected are given a recovery period between 1 and 7, to take into account the fact that they have an unknown infection time.

Logic

The iterate function loops through each tick in the schedule, implementing the following logic:

- Select the active network, N , based on the current tick.
- Calculate new latency states (see Equation 6.1).

$$L_i = \text{Ber}(1 - e^{-R_i}) \quad (6.1)$$

Where L_i is the latent state of individual i , Ber is the Bernoulli distribution function, and R_i is the infection rate of individual i , as calculated in Equation 6.2.

$$R_i = b + r \sum_{j=0}^n N_{ij} I_j \quad (6.2)$$

Where R_i is the infection rate of individual i , b is the background infection rate, r is the network infection rate, N is the active network, and I is the array of infection states (0 for not infectious, 1 for infectious).

At the start of each day, the following additional calculations are performed:

- Summarize the state of each person into a single array, with 0: susceptible, 1: latent, 2: infectious, 3: recovered.
- Record summary data.
- Update the latency and infectious periods.
- Update infection states.

5 Output

Outputs are saved to a hdf5 file, with `inf_data`, and `net_infect` groups.

5.1 Infection Data

In the `inf_data` group in the output file, a separate table is created per run. Each table is of size number of individuals by number of days, representing the infection states through the simulation:

- 0: susceptible,
- 1: latent,
- 2: infectious,
- 3: recovered.

5.2 Network Information

In the `net_infect` group in the output file, a separate table is created per run. Each table is of size number of individuals by 1, with each value representing the network that caused an infection:

- 0: no infection,
- 1: household network,
- 2: proximity network.

PART 7

Analysis

1 Summary

The analysis module processes the results from simulation and visualises the output.

2 Config File

The analysis module requires a config file with the following parameters defined:

- *chikwawa* [Boolean]: Flag identifying whether to carry out analysis for the Chikwawa study site.
- *chileka* [Boolean]: Flag identifying whether to carry out analysis for the Chileka study site.
- *ndirande* [Boolean]: Flag identifying whether to carry out analysis for the Ndirande study site.
- *model_dir* [Integer]: The sub-directory name to read the simulation outputs from, and save the analysis outputs to. The full folder paths will be `data/simulation/[site]/[model_dir]/` and `data/analysis/[site]/[model_dir]/`.
- *num_days* [Integer]: Number of days the simulation was run for.
- *run_no* [Integer]: The individual run to analyse.
- *figsize_a4* [Tuple[Float, Float]]: The dimensions for plots.
- *dpi* [Integer]: The DPI resolution to be used for plotting.
- *prev_res* [Integer]: The resolution to be used for kriging calculations (in metres).
- *jitter* [Integer]: Controls the level of plot jitter (in metres).

3 Analysis A

The first stage of analysis calculates prevalence estimations for the site, through the simulation, using ‘ordinary kriging’. Depending on the resolution defined in the config file, this can be a lengthy process, so is completed prior to any visualisation. Results from this stage are saved in a ‘temp_prev’ directory.

4 Analysis B

The second stage of analysis involves plotting the prevalence estimations calculated previously. A plot is generated for each day of the simulation, before combining into a single animation.

Additional plots will be included in this module during future development.

Appendix A

‘Run’ Script

1 drum.py

1.1 Config File

The DRUM model requires a master config file with the following parameters defined:

- *run_synth_pop* [Boolean]: Flag identifying whether to run the synthetic population module.
- *synth_pop_config* [String]: Filename for the config file for the synthetic population module.
- *run_network* [Boolean]: Flag identifying whether to run the networks module.
- *network_config* [String]: Filename for the config file for the networks module.
- *run_model* [Boolean]: Flag identifying whether to run the model module.
- *model_config* [String]: Filename for the config file for the model module.
- *run_sim* [Boolean]: Flag identifying whether to run the simulation module.
- *sim_config* [String]: Filename for the config file for the simulation module.
- *run_analysis* [Boolean]: Flag identifying whether to run the analysis module.
- *analysis_config* [String]: Filename for the config file for the analysis module.

1.2 drum.py Code

```
1 #!/usr/bin/env python
2
3 """DRUM Simulation"""
4
5 # Condition to allow running multiprocessing in sub-modules in Windows
6 # without error.
7 # See, for example:
8 # https://stackoverflow.com/questions/20222534/python-multiprocessing-on
9 # -windows-if-name-main
10
11 if __name__ == "__main__":
```

```

11 import os
12
13 from utils.parse import parse_config
14 from synthpop.generate_synthetic import create_synth_pop
15 from networks.generate_networks import create_networks
16 from model.generate_model import create_model
17 from sim.generate_sim import run_simulation
18 from analysis.analyse import run_analysis
19
20
21 def run():
22     """Execute sub-modules based on choices in config-master.ini"""
23     modules = parse_config('config-master.ini')
24
25     # Run synth pop generation
26     if modules['run_synth_pop'] == 'True':
27         print('-----')
28         if os.path.isfile(modules['synth_pop_config']):
29             print(f"Generating synthetic population using config: {
modules['synth_pop_config']}")
30             create_synth_pop(modules['synth_pop_config'])
31         else:
32             print("Can't generate synth pop - given config file
doesn't exist.")
33
34     # Run network generation
35     if modules['run_network'] == 'True':
36         print('-----')
37         if os.path.isfile(modules['network_config']):
38             print(f"Generating networks using config: {modules['
network_config']}")
39
40             create_networks(modules['network_config'])
41
42         else:
43             print("Can't generate network - given config file doesn'
t exist.")
44
45     # Run model generation
46     if modules['run_model'] == 'True':
47         print('-----')
48         if os.path.isfile(modules['model_config']):
49             print(f"Building model using config: {modules['
model_config']}")
50
51             create_model(modules['model_config'])
52
53         else:
54             print("Can't build model - given config file doesn't
exist.")
55
56     # Run simulation
57     if modules['run_sim'] == 'True':
58         print('-----')
59         if os.path.isfile(modules['sim_config']):
60             print(f"Running simulation using config: {modules['
sim_config']}")
61
62             run_simulation(modules['sim_config'])
63

```

```
64         else:
65             print("Can't run simulation - given config file doesn't
66 exist.")
67
68     # Run analysis
69     if modules['run_analysis'] == 'True':
70         print('-----')
71         if os.path.isfile(modules['analysis_config']):
72             print(f"Running analysis using config: {modules['
73 analysis_config']}")
74             run_analysis(modules['analysis_config'])
75
76         else:
77             print("Can't run analysis - given config file doesn't
78 exist.")
79     run()
```