USER GUIDE FOR THE NNDSS SYNTHETIC DATA GENERATORS

Prepared for:
Centers for Disease Control and Prevention (CDC)
The Center for Surveillance, Epidemiology, and Laboratory Sciences (CSELS)
Division of Health Informatics and Surveillance (DHIS)
1600 Clifton Rd. NE
Atlanta, Georgia 30329-4027

Prepared by:
Information and Communications Laboratory
Georgia Tech Research Institute
Georgia Institute of Technology
Atlanta, Georgia 30332

Under:
GTRI-CDC Innovation Collaboration
Contract W31P4Q-18-D-0002
Task Order W31P4Q-19-F-0584

GEORGIA INSTITUTE OF TECHNOLOGY
A Unit of the University System of Georgia
Atlanta, Georgia 30332

Georgia Tech Research Institute

CREATING THE NEXT®

GTRI.gatech.edu

USER GUIDE FOR THE NNDSS SYNTHETIC DATA GENERATORS

25 August 2021

Prepared for:

CENTERS FOR DISEASE CONTROL AND PREVENTION
The Center for Surveillance, Epidemiology, and Laboratory Sciences
Division of Health Informatics and Surveillance
1600 Clifton Rd NE
Atlanta, Georgia 30329-4027

Prepared by:

Information and Communications Laboratory
Georgia Tech Research Institute
GEORGIA INSTITUTE OF TECHNOLOGY
Atlanta, Georgia 30332

Initial Release
25 February 2021

Authored By:
GEORGIA TECH RESEARCH CORPORATION
Atlanta, Georgia 30332-0415

DISCLAIMERS

1. DISTRIBUTION STATEMENT B: Distribution authorized to U.S. Government Agencies only: Software Documentation, 22 Feb 2021. Other requests for this document shall be referred to the Division of Health Informatics and Surveillance Program Manager (DHIS), Centers for Disease Control and Prevention, 1600 Clifton Rd. NE, Atlanta, GA 30329-4027. Telephone: toll-free (800) 232-4636.
2. REPRODUCTION NOTICE: Unless otherwise explicitly noted, the technical data herein was generated and/or developed exclusively with Government funds. Technical data generated and/or developed exclusively with Government funds are provided with unlimited rights to the Government as defined under the Rights in Technical Data Clause, DFARS 252.227-7013. Unlimited rights means rights to use, modify, reproduce, perform, display, release, or disclose technical data in whole or in part, in any manner, and for any purpose whatsoever, and to have or authorize others to do so.

Revision Record

Revision	Date	Revised by	Description
-	2021-02-25	R. Boyd / GTRI A. Himschoot / GTRI	Initial Release
1	2021-08-25	R. Boyd / GTRI A. Himschoot / GTRI C. Heilig / CDC	Updated for latest code changes. Added new section containing an overview of the code structure and random number generator usage.

Table of Contents

Section **Page No.**

Table of Tables

Table **Page No.**

Table of Figures

Figure **Page No.**

1 INTRODUCTION

This document contains an overview of the NETSS and HL7 synthetic data generation software. Instructions are provided for system preparation, installation of required dependencies, NETSS and HL7 data preprocessing, and configuration and use of the synthetic data generators.

The NETSS and HL7 generators create synthetic data files for a specified condition in a specified jurisdiction. For instance, a user can create synthetic data for Hepatitis A in Oregon, Chlamydia in California, Lyme Disease in Connecticut, etc. This software does not support the creation of synthetic data for multiple jurisdictions or regions. Any jurisdictional or regional merging of data must be done by other means.

The synthetic data generators require source data for the desired condition and jurisdiction as input. Some aspects of the source data are modified by the generators to create the synthetic data. Other synthetic features are created *de novo* based on statistical properties of the original data. If a jurisdiction has no case data for one or more reportable conditions, the generators will not be able to create synthetic data for those conditions in that jurisdiction.

2 SYSTEM PREPARATION

The synthetic data generation software has been implemented in the python programming language and therefore requires a python runtime environment. We strongly recommend the use of Anaconda environments (command name *conda*) for python development. Python has a standard package manager called *pip*, but conda is able to resolve dependencies and version conflicts much better than pip. Conda also provides a modern Anaconda-compatible replacement for pip.

There are two Anaconda distributions: *conda*, a full-featured numerical computing, machine learning, and statistical software stack; and *miniconda*, a "lite" version of the full conda installation. Only one of these distributions should be installed at a time, and they both use the *conda* command for configuration. The following instructions assume the use of miniconda on a Linux or Mac system.

The software requires python version 3.6 or greater. The python 3.6 release improved python's random number generation capabilities significantly. The generators require these improvements to mitigate the risk of identifying a specific individual from the synthetic results. The generators detect the python version at initialization and will exit with an error message for python versions older than 3.6.

2.1 Install the Miniconda Distribution

1. Download the latest miniconda package:
   `wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh`

2. Compute the SHA-256 hash of the downloaded file:
   `sha256sum Miniconda3-latest-Linux-x86_64.sh`

3. Check the hash against those on this page:
   `https://conda.io/en/latest/miniconda_hashes.html`

4. Install miniconda:
   `bash Miniconda3-latest-Linux-x86_64.sh`
   <Accept the default install location and answer yes to all questions.>

5. Activate the installer's modifications to your `.bash_profile` file by closing the terminal window and starting a new terminal window.

6. Test the installation by listing the installed conda packages:
   `conda list`

7. If your system cannot find the conda executable, then something went wrong with the modifications to your PATH environment variable. Either edit the path by hand or consult the Anaconda documentation for further instructions.

8. Update the installation:
   `conda update conda`

2.2 Create a Conda Environment

A dedicated conda "environment" will be created for running the software. Conda environments are isolated from each other, can be activated and deactivated at will, and can be configured and updated independently from other environments. The isolation helps prevent incompatible software upgrades and other problems caused by shared system library folders.

The environment will be called `dev` and must be explicitly activated to run the generators:

```
conda create --name dev
<accept defaults when prompted>
```

2.3 Install Required Packages

The next step is to activate the `dev` environment and install the required python libraries into it:

```
conda activate dev
conda install -c anaconda jupyter
conda install -c anaconda scipy
conda install -c anaconda statsmodels
conda install -c conda-forge matplotlib
```

The installed versions of these packages should be at least:

```
jupyter:     1.0.0
scipy:       1.6.2
statsmodels: 0.12.2
matplotlib:  3.4.2
```

2.4 Cleanup

```
conda deactivate
rm Miniconda3-latest-Linux-x86_64.sh
```

2.5 Run the Software

A data preprocessing step is required before running the NETSS or HL7 synthetic data generators. The preprocessor is in a separate notebook called `create_preprocessed_data.ipynb`. Instructions on how to run the preprocessor can be found in the next section of this document. The preprocessing step only needs to be done a single time for each source data update.

To run either the NETSS or HL7 synthetic data generation Jupyter notebooks, open a terminal window and change directories to that containing the notebooks. The NETSS generator notebook file has the name `SyntheticGenerator_NETSS.ipynb`; the HL7 generator is named `SyntheticGenerator_HL7.ipynb`.

Activate the `dev` environment in this terminal window:

```
conda activate dev
```

Start Jupyter with this command:

```
jupyter notebook
```

Jupyter may take a few seconds to start, especially on the initial launch. A web browser window will appear after it loads and starts.

Click on either the NETSS or HL7 notebooks. The notebooks should launch and appear on the screen.

To run the notebooks, provide values for the configuration variables in the first several sections of the notebook. Note that the `OUTPUT_DIR` variable must be set to a directory that already exists – the code will not create it.

From the `Kernel` menu, select `Restart & Clear Output`, then `Restart & Run All`.

When finished running the notebooks, click the `Logout` button near the upper right-hand corner of the Jupyter interface. That will close the active notebook. To completely log out of Jupyter, click the `Logout` button on the remaining Jupyter window, then press `CTRL-C` in the terminal window.

To deactivate the `dev` conda environment, run the command `conda deactivate` in the terminal window.

3 DATA PREPROCESSING

The NETSS and HL7 synthetic data generators require a data preprocessing stage prior to use. This preprocessing stage takes the raw NETSS and HL7 data, extracts all data for each condition-jurisdiction pairing, and builds a specially-formatted file tree containing the condition-jurisdiction data. The data is organized to facilitate lookup of the source data by the generators. This section explains how perform this preprocessing step. This step only needs to be done once for each new data update.

Note: GTRI received the raw NETSS and HL7 data as CSV files, so the preprocessor assumes this format for the raw data.

The `create_preprocessed_data.ipynb` notebook contains the code for the preprocessor. The file `jurisdiction_to_state_mapping.csv` (located in the GitHub repository) is required by the preprocessor. The path to this file must be changed to the local path in the 7th cell of the preprocessor notebook.

To edit the preprocessor notebook, open a terminal window, change directories to that containing the preprocessor notebook file, and run these commands:

```
conda activate dev
jupyter notebook
```

The jupyter window will open in a web browser and the preprocessor notebook will be visible in the directory tree that appears.

3.1 NETSS Preprocessing

The notebook loads the following NETSS files: `year2015a.csv`, `year2016a.csv`, `year2017pt1.csv`, `year2017pt2a.csv`, `year2018.csv`, `nndss19.csv`, and `nndss20_update.csv`. The notebook paths for these files are relative to the GTRI AWS instance, so they will need to be changed to an appropriate location for the end user. If a file is not available, comment out the line for the unavailable file and remove it from the list of files to concatenate in the NETSS cell block. If a new file needs to be added, replicate one of the file read lines and modify it to the name of the new file. Add the name for the new file to the list of other files to concatenate to include it in the preprocessing.

The preprocessor uses the following columns from the NETSS data: AGE, AGETYPE, CASSTAT, COUNT, COUNTY, EVENT, EVENTD, HISPANIC, RACE, SEX, and STATE. These header names must be used and in this format for all of the files in the preprocessor. A missing or lower-case name will cause the notebook to raise an exception.

Special care is taken for the STATE and COUNTY columns to ensure accurate codes. If the COUNTY code associated with a data record is between 0 and 999, exclusive, and its concatenation with the STATE code yields a valid FIPS code, then the COUNTY code is kept. Otherwise, the COUNTY code is replaced with 999.

Once the files have been read and preprocessed, the preprocessor output files will be located in the directory called `netss_update`. This directory will be created in the same location as the `create_preprocessed_data.ipynb` notebook. Inside this directory, there will a sub-directory for each jurisdiction in the data. Within these sub-directories are csv files for each condition found in that jurisdiction.

3.2 HL7 Preprocessing

The preprocessor loads these HL7 files: `genCore_2019_Export_transid.csv`, `genCore_2020_Export_transid.csv`, `genRace.csv`, and `genAddress.csv`. The relative paths for these files in the preprocessor notebook will also need to be changed to the end user's paths. Moreover, the assumption is that the field delimiter for these files is the special character '^'. If the files are delimited using another character an exception will be raised. Additional core files can be added or removed in the same way as the NETSS files. Copy or remove the syntax needed to read in the file and then add or delete from the concatenation step.

The columns extracted from the core files are: `age`, `age_units`, `birth_date_str`, `case_status_txt`, `condition_code`, `diag_dt`, `died_dt`, `earliest_cnty_dt`, `earliest_state_dt`, `ethnicity_txt`, `first_elec_submit_dt`, `hosp_admit_dt`, `illness_onset_dt`, `invest_start_dt`, `notif_result_status`, `phd_notif_dt`, `pregnant`, `report_county`, `report_dt`, `report_state`, `sex`, `current_record_flag`, and `msg_transaction_id`. The core files must contain these columns and have them appear in lower-case. For the race file, only the race column is needed, and the address file needs only `msg_transaction_id` and `Subj_County`. Potential exceptions can occur when the `make_condensed_race_table` function is called. These exceptions will happen if one of the values in the race column is not contained as a key in the `race_code_map` dictionary inside the `make_condensed_race_table` function. This error can be remedied by adding the value as a key to the dictionary with its appropriate value.

If all the files are available and in the correct format, then the preprocessor will create a directory called `hl7_update` in the same path as the notebook. This folder will have the same structure as the preprocessed NETSS data with subdirectories as jurisdictions and filenames as condition codes.

One special feature of the HL7 data is that the records are spread over multiple source files. The preprocessor does the equivalent of a database join on these records to find all data for a specific case, and it generates a single output file for the generator to use.

Similar to the NETSS preprocessing, invalid county codes are adapted. If a record's county code is a five digit string, the first two digits match the jurisdiction code, and the code is an officially valid FIPS, then it is kept. If the code holds the first two criteria, but not the third, then the jurisdiction code is appended with '999' and replaces the original value. If first criterion is met and not the second, then the first two digits of the county code is appended with '999'. When the first criterion fails, '99999' replaces the county code.

Another feature of the HL7 data is that a given subject can have multiple races. The acceptable values for the race variable can be found in the PHIN VADS value set for race. The preprocessor looks up the concept code for each stated race and concatenates them into a string using a semicolon character to separate the codes. For instance, if a subject's race is "white, Asian, and other", the race string generated by the preprocessor would be the concatenation of the three relevant concept codes: `2106-3;2028-9;2131-1`. These concatenated race strings also appear in the HL7 generator output files as the `race_mapped` field.

3.3 Preprocessor Use

To run the preprocessor, open a command terminal and enter these commands:

```
conda activate dev
jupyter notebook
```

Open the notebook called `create_preprocessed_data.ipynb`. From the Kernel menu, select `Restart & Clear Output`, then `Restart & Run All`. The notebook will define all the necessary functions, read the files, and create the directories. Warning: the previously made preprocessed directories will be deleted and new empty folders will be created. The folders will refill with files once the preprocessor runs all cells. The notebook will terminate after the elapsed time has been printed out in the last cell. Normal run times are on the scale of 2-3 hours on the GTRI-CDC Amazon Web Services server.

4 JUPYTER NOTEBOOK USER GUIDE

This section describes how to use the NETSS and HL7 Jupyter notebooks to generate synthetic data files. Some configuration variables must be set on each run (such as the desired condition and jurisdiction). Other configuration variables need to be set only once.

4.1 Set the Configuration Variables

4.1.1 Specify the Jurisdiction

The NNDSS jurisdiction must be specified for each run. The jurisdiction is either a US state or one of the entities in the bulleted list below. Enter the jurisdiction as a python string, which is a sequence of characters enclosed in single or double quotes. Full state names or the standard United States Postal Service (USPS) two-letter abbreviations can be used. For example, either `"Georgia"` or `"GA"` is acceptable for the jurisdiction of the state of Georgia. NNDSS includes these 10 non-state jurisdictions as well:

- “American Samoa” or “AS”
- “District of Columbia” or “DC”
- "Federated States of Micronesia" or "FM"
- “Guam” or “GU”
- "Marshall Islands" or "MH"
- “New York City” or “NYC”
- “Northern Mariana Islands” or “MP”
- "Palau" or "PW"
- “Puerto Rico” or “PR”
- “Virgin Islands” or “VI”

```
In [1]: # example: JURISDICTION = "District of Columbia"
        JURISDICTION = 'CA'
```

If the user enters an invalid jurisdiction string, an error will be reported to the user and the run will terminate.

4.1.2 Specify the Condition Code

The condition code must be specified for each run. Enter the condition code as a five-digit python integer, which consists only of digits without quotes.

```
In [2]: # example: CONDITION_CODE = 11080
        CONDITION_CODE = 10274
```

Data for all reportable conditions does not necessarily exist in all jurisdictions, especially for rare conditions. If the user attempts to do a run using a condition code for which no source data is available, the system reports this fact to the user and the run terminates. The run will also terminate if the user enters an invalid condition code.

4.1.3 Specify Condition Grouping

(Optional) The synthetic data generators will group data for some rare conditions for which data is scarce. For instance, data for all hemorrhagic fevers in a given jurisdiction will be combined and used as the source data for purposes of generating synthetic data for any one of them.

Data grouping can be disabled by setting the `DISABLE_GROUPING` variable to the python Boolean value `True`. The default value of this variable is `False`, meaning that condition grouping will be performed. Note that python Booleans are different from strings and are not surrounded by quotes.

```
In [3]: # Set DISABLE_GROUPING = True to DISABLE grouping of datasets.
        # Set DISABLE_GROUPING = False to ENABLE grouping of datasets.
        # The command-line program has DISABLE_GROUPING = False as the default.
        DISABLE_GROUPING = False
```

The next table contains all condition codes for which data will be grouped, along with the codes for the related conditions. Syphilis data can also be grouped but is handled as a special case.

Condition Codes Eligible for Data Grouping	
Code	**Codes Included in Group**
10049	10056
10051	10064
10052	10065
10053	10062
10054	10061
10056	10049
10057	10063
10058	10066
10059	10068
10061	10054
10062	10053
10063	10057
10064	10051
10065	10052
10066	10058
10068	10059
10078	10079
10079	10078
10081	10082
10082	10081
10257	10258
10258	10257
10548	10549,10550
10549	10548,10550
10550	10548,10549
10660	11630,11631,11632,11637,11638,11639,11640,11644,11648
10680	11704,11705
11000	50242,50265
11630	10660,11631,11632,11637,11638,11639,11640,11644,11648
11631	10660,11630,11632,11637,11638,11639,11640,11644,11648
11632	10660,11630,11631,11637,11638,11639,11640,11644,11648
11637	10660,11630,11631,11632,11638,11639,11640,11644,11648
11638	10660,11630,11631,11632,11637,11639,11640,11644,11648
11639	10660,11630,11631,11632,11637,11638,11640,11644,11648
11640	10660,11630,11631,11632,11637,11638,11639,11644,11648
11644	10660,11630,11631,11632,11637,11638,11639,11640,11648
11648	10660,11630,11631,11632,11637,11638,11639,11640,11644
11704	10680,11705
11705	10680,11704
11726	50221,50223
11736	50222,50224
50221	11726,50223
50222	11736,50224
50223	11726,50221
50224	11736,50222
50242	11000,50265
50265	11000,50242

Table 1: Condition Code Groups

4.1.3.1 Special Handling for Syphilis Grouping

(Optional) Syphilis requires special handling. Three of the Syphilis codes can be members of two different groups, unlike the previous codes which can belong to at most one group. The three Syphilis codes

- 10310: Syphilis, total primary and secondary
- 10311: Syphilis, primary
- 10312: Syphilis, secondary

can be members of either of these groups:

- Group 1: Syphilis primary and secondary (10310, 10311, 10312)
- Group 2: Syphilis total: (10310, 10311, 10312, 10313, 10314, 10316, 10319, 10320)

If the user desires to generate synthetic data for any of the codes 10310, 10311, or 10312, use the `SYPHILIS_TOTAL` Boolean variable to indicate whether the Syphilis total group (group 2 above) is desired or not:

```
In [4]: # Set SYPHILIS_TOTAL = False for Group 1
        # Set SYPHILIS_TOTAL = True for Group 2
        # The command-line program has SYPHILIS_TOTAL = False as the default.
        SYPHILIS_TOTAL = True
```

This Boolean variable is ignored if `ENABLE_GROUPING = False`, as well as for any codes other than 10310, 10311, or 10312. The values `True` and `False` are the only acceptable values for this variable.

4.1.4 Specify the Output Directory

This configuration step only needs to be done once. Create a folder into which the synthetic data files will be written. This folder must exist at the start of a run – the generators will not create it. If the user specifies a nonexistent directory the generator will report this fact and terminate the run. The notebooks are configured so that the synthetic data files will be written to default directories called `synthetic_data_netss` and `synthetic_data_hl7`.

On a Linux or Mac system, an output directory can be created with the user interface or by opening a command terminal window and entering these commands:

```
cd /path/to/containing/directory
mkdir output_directory_name
```

For example:

```
cd ~/data/nndss
mkdir synthetic_data_netss
```

4.1.5 Specify the Output File Name

(Optional) The output file name is a python string which should be surrounded with either single or double quotes.

If the output file name is omitted the generators will give the synthetic data file a default name and write it to the output directory. The default name has the form `synthetic_<code_list>_<jurisdiction>.csv`, where `code_list` is a list of one or more codes that were grouped, as described above. To use the default name, set the `OUTPUT_FILE_NAME` variable equal to the special python value `None` (no quotes).

For example: the default output file name for Hepatitis A (code 10106) in Oregon is `synthetic_10106_or.csv`. The default output file name for Syphilis group 1 (condition codes 10310, 10311, and 10312) is `synthetic_10310_10311_10312_ca.csv`. The default output file name for Syphilis group 2 in CA is `synthetic_syphilis_total_ca.csv`.

The generators support two output formats: CSV and JSON, with CSV being the default.

```
In [5]: # To use the default system-generated name:      OUTPUT_FILE_NAME = None
        # To override the system-generated name:        OUTPUT_FILE_NAME = "my_file_name.csv" or "my_file_name.json"
        # (If the name is anything other than the python special value None, quotes are required.)
        # The supported output formats are CSV and JSON. If the extension is omitted the format will default to CSV.
        OUTPUT_FILE_NAME = None
        #OUTPUT_FILE_NAME = 'myfile.json'
```

4.1.6 Specify the Source Directory

This configuration step must be done once. Set either the `NETSS_DIR` or `HL7_DIR` configuration variables to the path where the preprocessed data file tree can be found on your system. The generators need this path to locate the preprocessed source data files. This variable is a string surrounded by single or double quotes.

4.1.7 Specify the Number of Synthetic Samples to Generate

(Optional) The generators will typically create only as many synthetic samples as are required by the input data. This is the default behavior whenever the `NUM_SAMPLES` variable is set to the special python value `None`. Additional samples can be generated by setting `NUM_SAMPLES` to an integer value (digits only, no quotes).

4.1.8 Specify the Seed for the Random Number Generator

(Optional) The generators derive all random numbers from a single seed value. The use of different seeds causes the results to differ somewhat from run to run. Repeatability can be achieved by providing the random number generator with an integer seed value by use of the `RNG_SEED` configuration variable. The default value for this variable is the python special value `None`, which causes the generator to derive a new random seed with each run.

4.1.9 Specify Whether Debugging Output Should be Generated

(Optional) Set the `DEBUG` variable equal to the python Boolean value `True` to enable additional debug output. The default behavior of the generators is to suppress this debug output.

4.2 Run the Notebooks

With the configuration variables set (see previous section), run the NETSS or HL7 notebooks by clicking on the `Kernel` menu and selecting `Restart & Clear Output`. The numbers inside the brackets for each cell should disappear, along with any plots and data from the previous run. Then select `Restart & Run All` from the `Kernel` menu to start the new run.

When notebook execution starts, all cells will display a “*” character inside the cell brackets. As cells in the notebook execute, the “*” character gets replaced by integers, which record the sequential execution order of the notebook cells. Eventually the notebook should terminate.

If any errors occur, the cells that failed to execute will have empty brackets beside them. Scroll upwards through the notebook until you see the error message.

One error that will prove fatal to notebook execution is an original data set with either zero cases or only a single case report. The synthetic data generation algorithms require at least two sources cases, so the run will fail otherwise.

If the notebook execution is successful, the output file will be written to the `OUTPUT_DIR` (see above) and given the name `OUTPUT_FILE_NAME` as described above.

5 COMMAND-LINE PROGRAMS

Command-line synthetic data generation programs have also been provided, which may, in some instances, be more convenient to use than Jupyter notebooks. The command-line generators can be controlled via shell scripts and hence are useful for batch jobs, overnight runs, and other non-interactive scenarios. The command-line programs only write text to stdout – they do not produce any graphics.

5.1 How to Run the Command-Line Programs

The configuration variables for the command-line programs can be set with command-line arguments. These are "long" arguments in Unix parlance, each requiring two dashes. The complete set of command-line arguments supported by each program can be seen by running the programs with only the `--help` option, i.e.:

```
python ./<program name> --help
```

The command-line arguments are listed in the next table.

Command-Line Arguments		
Argument	**Required**	**Description**
`--help`	No	show help message and exit
`--hl7_dir, --netss_dir`	Yes	[quoted string] path to the root of the HL7 or NETSS preprocessed file trees
`--jurisdiction`	Yes	[quoted string] name or code of the jurisdiction
`--code`	Yes	[integer] condition code
`--outfile`	No	[quoted string] output filepath; if omitted a default name will be generated
`--num_samples`	No	[integer] number of samples to generate; if omitted the number required to simulate the full source dataset will be generated
`--rng_seed`	No	[integer] use the provided integer as the seed for the random number generator
`--disable_grouping`	No	[flag, no arguments] disable condition grouping; if this flag is omitted, condition grouping will be performed
`--syphilis_total`	No	[flag, no arguments] combine all syphilis data into one group, as described above
`--debug`	No	[flag, no arguments] print additional debugging information to stdout

Table 2: Command-Line Arguments

Suppose a user wants to generate synthetic data for Lyme disease in Connecticut. This user prefers to use the default system-generated name for the output file (`synthetic_11080_ct.csv`) and have it written to the default NETSS output directory (`synthetic_results_netss`).

First, the user opens a terminal window and browses to the location of the NETSS synthetic data generation command-line program (`gen_synthetic_data_netss.py`). Next the user creates the `synthetic_results_netss` folder inside the current working directory if it does not already exist.

The user generates the output file with this command:

```
python ./gen_synthetic_data_netss.py
      --netss-dir="/path/to/preprocessed/netss/files"
      --jurisdiction="CT"
      --code=11080
```

To write the results to a file named `lyme_disease_ct.json`, use this command:

```
python ./gen_synthetic_data_netss.py
      --netss-dir="/path/to/preprocessed/netss/files"
      --jurisdiction="CT"
      --code=11080
      --outfile="lyme_disease_ct.json"
```

To generate synthetic HL7 data for primary Syphilis (code 10311) in Michigan, to combine any other Syphilis data in Michigan together to form the source data, and to write the synthetic data to a file called `syphilis_grouped_mi.csv`, run this command:

```
python ./gen_synthetic_data_netss.py
      --hl7-dir="/path/to/preprocessed/hl7/files"
      --jurisdiction="MI"
      --code=10311
      --outfile="syphilis_grouped_mi.csv"
      --syphilis_total
```

To repeat the same run, to NOT group the data with other syphilis data, and to use a random number generator seed of 42, run this command:

```
python ./gen_synthetic_data_netss.py
      --hl7-dir="/path/to/preprocessed/hl7/files"
      --jurisdiction="MI"
      --code=10311
      --outfile="syphilis_grouped_mi.csv"
      --rng_seed=42
```

Be careful to not use spaces on either side of the equal sign with the command-line arguments.

6 OVERVIEW OF THE SYNTHETIC DATA GENERATION PROCESS

This section contains a high-level technical description of how the synthetic data is generated. The material in this section is provided purely for informational purposes and is not essential for use of the generators.

6.1 Preliminary Tasks

The first task for the generator software is to load all of the required python libraries. The code makes heavy use of NumPy, the leading package for scientific computing in Python, so it loads that module and all of its required dependencies. The synthetic data generation code has been structured to do most of its work in modules used both by the notebooks and the command-line programs, so all such modules are loaded next. These modules can be found in the `/src` folder.

After the library imports finish, the software enables debug output if the user requested it, then it constructs a list of condition codes. Normally each run requires a single condition code, so this list typically contains a single element. If the user wants to simulate a condition for which grouping is enabled, the software finds all grouped codes and adds them to the list.

The preprocessor associates each condition code with a source data file in a specially-configured directory tree. The synthetic data generators use each code and the jurisdiction to construct the fully-qualified path to each input file. Some jurisdictions may not have any source data for certain codes, so those filepaths are pruned from the list. The end result is a fully-qualified list of source files that the generators will need to load and merge for the run.

With the source file list available, the generators next construct the path to the output file. If the user did not specify either an output directory or an output file name, the system generates a default output directory and a default file name. As mentioned previously, this output directory must exist at the start of the run, since the generator will not create it.

The last setup task is to seed the random number generator, either with the user-supplied seed or a random value. Random seeds consist of 128 bits of entropy extracted from the system entropy pool. Numpy uses this value to initialize the random number generator in a cryptographically secure manner.

The generation and handling of of the random number sequence is important for hardening the synthetic results against attempts at re-identification. A detailed explanation of the generator's use of random numbers is provided in section 7.

6.2 Load Input Files and Remap Data

The first major activity of the generators is to load the input files. This is a tedious process that requires careful checking of each field in each record for conformance to the NETSS specification or PHIN VADS value sets. The file loaders make an attempt to understand non-conformant input by examining text matches on prefix strings. If the input is non-conformant and the code cannot determine what is meant, it maps the value to an appropriate unknown category.

If file merging is required, the process above is repeated for each input file. The data from each file consists of an array of timestamped records. The data from all files is placed into a map that builds a list of input records for each shared date. The data is read out in sequential order by date. The contents of each list (for a given date) are randomly shuffled, and the entire dataset is written to a temporary file. This temporary file thus contains all data from all input files and is used as the source file for the run.

With the data loaded and merged, the next activity is to map the specification-conformant field values to numeric values suitable for computation. As an example, the NETSS age variable spans the range from 0-120 years. The unknown age value is 999. The generators remap this unknown value to -1 so that all age variables form a contiguous set. The remapping process is performed on the values of each variable.

At this point the data consists of numeric values suitable for computation. The generators compute the Kendall tau correlation matrix for the source data, as well as the marginal distributions for each variable. These

distributions are used to compute the cumulative distribution functions and their inverses, which are needed for variable transformations later in the pipeline.

6.3 Signal Processing

The signal processing code runs next. The purpose of the signal processing operations is to compute the number of synthetic case reports for each day of the simulation. The number of case reports vs. time in the source data forms a "signal" from which the synthetic case report counts can be derived.

The signal processing for NETSS begins with an imperfect Fourier reconstruction of the source signal. A Fast Fourier Transform (FFT) of the source signal is the first step of this process. Some random Gaussian phase noise is added to a percentage of the high-frequency Fourier components. An inverse FFT is then performed on this noise-modified signal. The effect of the noise is to alter the signal from its original form, so that the reconstructed signal differs somewhat from the original. The amount of noise added and the percentage of the Fourier components modified were determined empirically through trial-and-error experimentation.

The noisy Fourier reconstruction process is not performed for the HL7 data. The reason is that the HL7 data often features long initial spans of zero or very few case reports, followed by a sudden increase in the case counts thereafter. The noisy reconstruction was found to add an unacceptable amount of noise in the initial "quiescent" region. Parameter tuning did not remove the noise, but instead simply altered its appearance. It was felt that the best overall results were obtained by omitting this imperfect Fourier reconstruction as the initial signal processing step.

For the HL7 data, the case count signal is determined by a different technique. Each HL7 data record contains multiple dates, such as the illness onset date, the date of diagnosis, etc. Some of these dates may be present in a given source data record, others may not. The code examines the available dates in each record and finds the earliest one. This date becomes the key to a map that associates a list of data records that all share a given earliest date. Following the construction of the map, the dates are read out in sequential order, and the number of data records sharing each date becomes the HL7 daily case count signal.

The code also has the capability to use the median date instead of the (default) earliest date. To use the median date a source code change is required. This change must be made prior to starting a run. The HL7 notebook and command-line program import a file named `src/model_data_hl7.py`. A variable in this file must be changed to enable use of the median date. To make this change, open the file in a text editor and go to line 81 where the variable _TUPLE_ANCHOR is initialized. Assign the value `_TUPLE_ANCHOR_MEDIAN` to this variable to cause the code to use the median date. Assign the value `_TUPLE_ANCHOR_EARLIEST` to this variable to use the earliest date instead.

An adaptive noise generation process is the next transform to be performed on the signal. This stage proceeds by applying a median filter to the signal to remove large spikes, which generally appear above a much smaller level of background cases. The presence of large spikes as input to the adaptive noise generation process tends to cause the added noise to be much too large.

After median filtering, the code scans a seven-day window over the signal, samples the local signal standard deviation within the window, and adds an amount of zero-mean Gaussian noise proportional to the local standard deviation. The zero-mean character of the added noise is important, since it can both add and subtract from the original case counts. Additional amplitude scaling in an amount proportional to the overall number of cases is also performed. The net effect of this process is to generate additive noise with reduced amplitude in the quieter regions of the signal and greater amplitude in the noisier portions. The additive noise then gets added to the original signal (not the median filtered version) to modify it.

The final signal processing operation applies only to sparse regions of the signal. The code segments the signal into 60-day windows and computes the number of zero values in this window. If the fraction of zero values in the window exceeds a threshold (0.9 is the default), the region is deemed to be sparse and is modified as follows. The nonzero values in the window are averaged and the average value clipped to the interval [1.0, 3.0] (empirically determined). Then each nonzero value is selected and has a 1-in-3 chance of being modified by either adding or subtracting the clipped average value. Then the zero locations within the window are randomly

shuffled and replaced with modified nonzero values. Further checks are applied to guarantee that least two nonzero values appear in the resulting signal, a condition required by subsequent stages of the pipeline.

6.4 Copula Model

The synthetic data generators use a copula model to generate correlated categorical variables. The basic idea is to compute the Kendall tau rank correlation matrix for the source data, derive a covariance matrix from it, and use that covariance matrix in a multivariate normal random number generator. The multivariate normal random variables that emerge ideally have the same Kendall tau rank correlation matrix as the source data, since each component of the normally-distributed n-tuples corresponds to one of the variables in the source data.

After generating the multivariate normal samples, each component of each sample is independently transformed to a uniform distribution on [0,1], then transformed again using the inverse empirical cumulative distribution for each variable. The net result is the generation of correlated categorical variables that conform to the marginal distributions for each variable. The two transformations employed are monotonic and increasing, thereby preserving the rank correlations inherent in the data. In this manner the synthetic generators create correlated synthetic variables that preserve both the correlations and the marginal distributions for the varibles.

6.4.1 Derivation of the Covariance Matrix

A random number generator for a multivariate normal distribution requires a covariance matrix as input. The synthetic generators derive this matrix from the Kendall tau rank correlation matrix. An analytic relationship between the two matrices exists in two dimensions only. If τ is the Kendall's tau rank correlation coefficient and ρ is the Pearson correlation coefficient, the two are related by: $\rho = \sin\frac{\pi}{2}\tau$. A 2D covariance matrix Σ would be constructed as $\Sigma = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}$. The generators use the relationship between ρ and τ for each component in higher dimensions. Typically the number of dimensions is six for NETSS and seven for HL7.

Sometimes this approximation creates a matrix with negative eigenvalues, which causes the multivariate Gaussian random number generation process to fail. The technique of simply setting the negative eigenvalues to zero and reconstructing the matrix produces a result that frequently fails in practice.

Following some ideas of Higham and collaborators, the generators implement a matrix repair procedure that attempts to remove the negative eigenvalues and iteratively restore the matrix to a positive semidefinite state. The condition of positive semidifiniteness can be tested by attempting a Cholesky factorization of the matrix. If the matrix repair procedure ultimately fails, the software defaults to using an identity matrix for the multivariate normal covariance matrix. The software generates diagnostic output that clearly shows whether the original covariance matrix was problematic and if the repair procedure worked. Testing to date has yet to uncover a situation in which the matrix repair procedure fails.

6.4.2 Correlated HL7 Synthetic Dates and Pseudopersons

The HL7 generator uses the concept of "pseudoperson" to generate synthetic correlated dates. The HL7 data features many different date variables, such as the diagnosis date, the illness onset date, the date the case was first reported to the CDC, and several others. These dates are obviously correlated and hence constitute another form of data correlation to be preserved in the synthetic results.

The HL7 source data is extremely sparse prior to 2015-01-01. By agreement with the CDC, any of these dates that occur prior to 2015-01-01 are treated by the HL7 generator as missing values and are replaced with empty strings.

Each HL7 data record includes the sex of the patient (male, female, unknown), as well as the case status (probable, suspected, confirmed, or unknown). Each independent combination of these values denotes a "pseudoperson" type. There are 3x4 = 12 possible pseudoperson types. These types can be enumerated by using a two letter scheme, in which the first letter corresponds to a sex value (M, F, U) and the second letter corresponds to a case status value: (P, S, C, U). The twelve pseudoperson types are thus: (MP, MS, MC, MU, FP, FS, FC, FU, UP, US, UC, UU).

In practice all twelve possible pseudoperson types are not needed, and the software actually uses only eight types. These types were determined by looking at the aggregate pseudoperson data for the entire HL7 dataset. The relevant types can be determined from a pseudoperson heatmap as seen in the next figure:

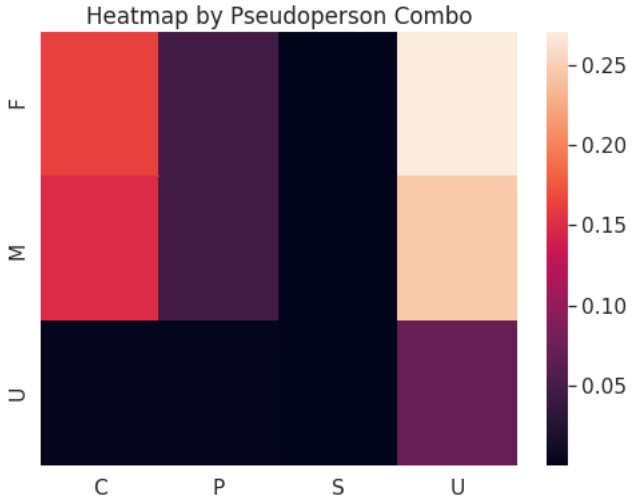

Figure 1: Pseudoperson Heatmap

This heatmap shows a color-coded depiction of the relative pseudoperson populations in the HL7 data. Sex values appear along the left edge and case status values along the bottom edge. Lighter colors have more data and darker colors have less, as indicated by the scale at the right edge of the image. Note that the pseudoperson categories UC, UP, US, MS, and FS are all darkly shaded and have relatively few records overall.

The HL7 input module loads the HL7 data and partitions the records into disjoint pseudoperson sets. The HL7 synthetic generator combines the UC, UP, US, MS, and FS categories into a single OTHER category, thereby creating a total of eight disjoint pseudoperson datasets: MC, MP, FC, FP, UU, MU, FU, and OTHER.

For each pseudoperson dataset, the software extracts the various dates of interest and computes the earliest-occurring date for each. The offset in days from this earliest date is computed for each such date, and a “date tuple” of these day offsets is created. The date tuples for each pseudoperson category are accumulated and indexed, the index distribution computed, as well as the inverse cumulative distribution function for the tuple indices. The output stage of the pipeline uses this inverse CDF to transform a uniform random number on [0,1] to a tuple index and thus to a date tuple for each synthetic data record.

Sometimes a synthetic pseudoperson is created that does not appear in the source data. The reason for this is that the sex and case status variables are generated independently. Even though the synthetic sex and case status values conform to the marginal distribution for each, the simultaneous combination of those values may represent a pseudoperson type not present in the source data. To prevent this occurrence, the synthetic data is scanned for invalid pseudoperson types. If any are found, another batch of synthetic categorical variables is generated and any records containing invalid pseudoperson types are replaced with valid records. Typically only a small fraction of the synthetic records need to be replaced.

6.5 Output File Generation

The generators create output files in either CSV or JSON format, with CSV being the default. The file writers iterate through each day of the simulation and extract the number of synthetic cases for that day from the signal processing results. For each of those cases, a synthetic categorical record is obtained from a precomputed array of such records. For each record, the pseudoperson type is obtained and a random date tuple is drawn from the date tuple distribution for that pseudoperson type. The current day of the simulation is taken to be the earliest date for the tuple. Since the date tuple records the offset in days for each relevant date of interest, those offsets are added to the current day of the simulation to produce actual synthetic dates for the current record. If any synthetic dates occur more 30 days past the end date of the simulation, those dates are truncated

at the 30-day limit. With the categorical values and correlated dates now available, the file writers create a new synthetic record for the current day and write it into the output file. This process continues until all days of the simulation have been processed.

The NETSS output file contains the fields in the next table. The CSV-formatted files contain a separate column for each variable. The JSON-formatted files only include a field if the data for that field is present.

NETSS Synthetic Data File Fields	
Field Name	**Interpretation**
EVENTD	Event date in YYYY-MM-DD format
COUNT	Number of cases in the current record for the current EVENTD, either 0 or 1
AGE	Age of the subject
AGETYPE	Units for the AGE variable
SEX	Sex of the subject
RACE	Race of the subject
HISPANIC	Whether the subject's ethnicity is Hispanic or not
CASSTAT	Case status
COUNTY	County index

Table 3: NETSS Synthetic Data File Fields

The HL7 output file contains the fields in the following table, with the same caveats for the presence or absence of a given field in the JSON output file.

Dates (actually datetimes) are formatted as YYYY-MM-SS HH:MM:SS. All synthetic dates have HH:MM:SS set to 00:00:00.

The `report_dt_new` field is a CDC-modification of the HL7 `report_dt` field. The value of this field is computed according to the following logic:

Definition of `report_dt_new`:

if `report_dt` is present use that
else if `phd_notif_dt` is present use that
else if `earliest_state_dt` is present use that
else if `earliest_cnty_dt` is present use that
else leave field blank

The HL7 generator maintains correlations among the following date variables: `diag_dt`, `died_dt`, `first_elec_submit_dt`, `report_dt_new`, `hosp_admit_dt`, `illness_onset_dt`, and `invest_start_dt`.

HL7 Synthetic Data File Fields	
Field Name	**Interpretation**
report_dt_new	CDC-modified report date in YYYY-MM-DD 00:00:00 format
count	Number of cases in the current record, either 0 or 1
age	Age of the subject
age_units	Units for the `age` variable (the HL7 generator converts all ages to years)

sex	Sex of the subject
ethnicity_txt	Whether the subject's ethnicity is Hispanic or not
race_mapped	Semicolon-concatenated string of all subject races (see section 3.2 for more on this)
case_status_txt	Case status
birth_date_str	Birth date of the subject. The synthetic generators use the `age` variable instead, so this field is always blank in the synthetic output.
notif_result_status	All data records with nonzero counts marked as final
pregnant	Whether the subject was pregnant during the illness
report_county	County reporting the notification.
first_elec_submit_dt	Date the case notification was first sent to the CDC.
subj_county	County of residence of the subject.
diag_dt	Earliest date of diagnosis for this condition.
died_dt	Date of death, if the subject died from this condition.
earliest_cnty_dt	Earliest date reported to county public health system. This field is incorporated into `report_dt_new` and is always blank in the synthetic output.
earliest_state_dt	Earliest date reported to state public health system. This field is incorporated into `report_dt_new` and is always blank in the synthetic output.
hosp_admit_dt	Subject's most recent admission to the hospital, if any, for this condition.
illness_onset_dt	Date of onset of symptoms for this condition.
invest_start_dt	Date the case investigation was initiated.
phd_notif_dt	Date the report was first sent to the public health dept (local, county, or state). This field is incorporated into `report_dt_new` and is always blank in the synthetic output.

Table 4: HL7 Synthetic Data File Fields

6.6 Use of the Synthetic Results as Input

The synthetic output files can actually be loaded by the generators and used as input. The generator notebooks do this as a means of checking the output and for creating correlation matrix element plots.

One use case for this technique is the substitution of synthetic data files for the preprocessor-generated files, all of which require HIPAA data protections. For systems that are not HIPAA-certified, a tree of synthetically-generated files can be installed and used instead.

7 CODE STRUCTURE AND RANDOM NUMBER GENERATOR USAGE

This section reviews the Python code base for the NNDSS synthetic data generators. This overview should help other users understand the content and broad functionality and interdependencies of the code as provided. In addition, since the generators are based on properties of real, sensitive data, this overview details the uses of pseudorandom number generators with a view toward ensuring that the code and the synthetic data do not present any disclosure risks stemming from the use of real data.

As explained in the preceding sections of this document, the synthetic data generator accommodates source data in the legacy NETSS format and in an adapted version of the modern HL7 format. This section addresses which components cover both NETSS- and HL7-format source data and differentiates components specific to each format.

7.1 Python Code Structure

The Python code base is organized in the source Git repository in 2 directories: `generator` and `generator/src`.

The directory `generator` contains 4 Jupyter notebooks:

- `create_preprocessed_data.ipynb` – Notebook for preprocessing source data (11 CSV files) into a file structure and format suitable for further processing (1 file for each unique combination of NETSS- or HL7-format jurisdiction and condition code). These source files contain sensitive data.
- `Date_Sequences_and_Missingness.ipynb` – Notebook for analyzing the quality of and interrelationships between date fields in HL7-format source data. (NETSS has only 1 date field.)
- `SyntheticGenerator_NETSS.ipynb` – Notebook for generating synthetic data from NETSS-format source data, given a jurisdiction and condition code.
- `SyntheticGenerator_HL7.ipynb` – Notebook for generating synthetic data from HL7-format source data, given a jurisdiction and condition code.

The directory `generator` contains 4 command-line modules:

- `gen_synthetic_data_netss.py` – Command-line module for generating synthetic data from NETSS-format source data, given a jurisdiction and condition code.
- `gen_synthetic_data_hl7.py` – Command-line module for generating synthetic data from HL7-format source data, given a jurisdiction and condition code.
- `tester_netss.py` – Command-line module for generating synthetic data for every available condition from NETSS-format source data, given a single jurisdiction or ‘all’ for all jurisdictions.
- `tester_hl7.py` – Command-line module for generating synthetic data for every available condition from HL7-format source data, given a single jurisdiction or ‘all’ for all jurisdictions.

The subdirectory `generator/src` contains 13 additional modules.

The subdirectory `generator/src` contains 6 modules that perform operations for both NETSS and HL7 sources:

- `jurisdictions.py` – Module for creating mappings between jurisdiction codes and jurisdiction names.

- `timeseries.py` – Module for synthesizing daily notification counts using Fourier and adaptive noise methods.
- `ecdf.py` – Module for implementing empirical cumulative distribution functions and their inverses. Used in synthesizing person-level demographic characteristics.
- `kernel_density_estimation.py` – Module for interpolating mass in synthesizing person-level demographic characteristics. Depends on modules `ecdf` and `netss` (not sure that the latter dependency is essential).
- `correlation_matrix.py` – Module for computing Kendall's tau-b correlation matrix. Used in synthesizing person-level demographic characteristics.
- `synthetic_data_model.py` – Module for normalizing input values (checking value sets, mathematical conditions, and the like) and to set up for output values. Depends on modules `jurisdictions`, `correlation_matrix`, and `timeseries`.

The subdirectory `generator/src` contains 3 modules that perform operations for NETSS sources:

- `netss.py` – Module for setting up NETSS-specific structural attributes, such as input and output variable names.
- `plots.py` – Module (called by NETSS notebook) for graphing marginal distributions of person-level characteristics, pairwise correlations, and time series. Depends on module `netss`.
- `model_data_netss.py` – Module for managing input data, setting up objects for analysis and synthetic data, generating synthetic data, and writing generated data to CSV or JSON format. Depends on modules `netss`, `correlation_matrix`, and `plots`.

The subdirectory `generator/src` contains 4 modules that perform operations for HL7 sources:

- `hl7.py` – Module for setting up HL7-specific structural attributes, such as input and output variable names.
- `plots_hl7.py` – Module (called by HL7 notebook) for graphing marginal distributions of person-level characteristics, pairwise correlations, time series, and additional date-related information. Depends on module `hl7`.
- `model_data_hl7.py` – Module for managing input data, setting up objects for analysis and synthetic data, generating synthetic data, and writing generated data to CSV or JSON format. Depends on modules `hl7`, `correlation_matrix`, and `plots_hl7`.
- `pseudoperson.py` – Module to support synthesizing person-level event dates based on a concept of a "pseudoperson". Depends on modules `hl7`, `synthetic_data_model`, and `model_data_hl7`.

7.2 Python Code Sequence, Emphasizing Random Number Generation

This section reviews the how the HL7-format synthetic data generator uses pseudorandom number generators (PNRGs). The NETSS-related process contains a strict subset of the same methods and will not be detailed here.

7.2.1 Code Sequence

Broadly, the HL7-based synthetic data generator uses randomness for 3 main purposes: (1) to generate a time series of daily notification counts that mimics properties of the source data, (2) to generate 7[i] person-level characteristics with a distribution that mimics those characteristics in the source data, and (3) to sample person-level event date information based on individual-level sex and case status[ii]. In addition, the synthetic data generator uses randomness to shuffle records.

The notebook `SyntheticGenerator_HL7.ipynb` and the command-line module `gen_synthetic_data_hl7.py` coordinate essentially the same tasks based on HL7-format source data, except that the notebook also creates illustrative and diagnostic graphics. Both the notebook and the command-line interface allow the user to specify an integer-valued seed for an underlying PRNG. The generator will produce the same synthetic data given the same input data sources and the same seed. If the user does not specify a seed, then the code sets a random seed.[iii]

Assume that HL7-format source data have been preprocessed in notebook `create_preprocessed_data.ipynb` and that the user is running either the notebook `SyntheticGenerator_HL7.ipynb` or the command-line module `gen_synthetic_data_hl7` with a specified jurisdiction and condition code, no seed for the pseudorandom number generator, and a request to write the resulting synthetic data to a CSV file with the default name and location. The following sequence of events occurs, keyed to the subheadings in this guide, section 6. This description is keyed to the line numbers in `gen_synthetic_data_hl7.py` and is followed by comprehensive module-by-module and line-by-line references.

Preliminary tasks. The user specifies, or the system retrieves, an integer value for the seed. The user-supplied arguments are checked for validity and file names and locations are set up. A call to `synthetic_data_model.init_rng` instantiates a pseudorandom number generator. By default, object `rng_seed` takes value `None` (line 161). If the user specifies a seed value, then that value is assigned as an integer to the object `rng_seed` (lines 135-138, 166-167). The PRNG is initialized (line 268) by passing `rng_seed` to function `init_rng` in module `synthetic_data_model`. If `init_rng` is not passed a user-specified seed, then it causes NumPy to generate a 128-bit random seed with bits drawn from the system entropy pool. Function `init_rng` returns a PRNG `Generator` instance `rng` created by the `default_rng` constructor in the NumPy module `random`. As of NumPy version 1.17 and later, the first call to the `default_rng` constructor initializes a PRNG, which defaults to a `Generator` instance of O'Neill's permutation congruential generator PCG-64. Subsequent calls to `Generator` methods applied to the `rng` instance generate pseudorandom values.

Load input files and remap data. A call to `model_data_hl7.init_model_data` loads and combines all input files for the selected jurisdiction and condition code(s), builds all data structures, and initializes the overall process, which includes computing data summaries for person-level characteristics (used in the copula model). The file preparation uses the previously instantiated PRNG.

Signal processing. The time series of case notifications per day is computed from the source data. Then calls to `timeseries.gen_synthetic_fourier` and `timeseries.modify_sparse_segments` generate a synthetic timeseries using Fourier methods with added adaptive noise and, if the series is sparse, modifies some nonzero values. These actions use the previously instantiated PRNG.

Person-level characteristics (copula model). A call to `synthetic_data_model.copula_n_variable_model` generates random variates that mimic source-data properties. Since the synthetic values of sex and case status will be used to generate synthetic person-level event dates, a call to `pseudoperson.correct_invalid_pseudopersons` ensures that this will be a valid process. If the previous step generated combinations of sex and case status that do not appear in the soure data, then this step calls back to `copula_n_variable_model` to generate additional synthetic persons until the correct number of valid synthetic persons is obtained. These actions use the previously instantiated PRNG.

Person-level event dates. A call to `model_data_hl7.generate_date_tuples` samples pseudopersons at random for assigning person-level event dates. These calls use the previously instantiated PRNG.

7.2.2 Code details for PRNG usage, by module and line number

Module `gen_synthetic_data_hl7`, as a command-line interface

- 268: calls function `init_rng` in module `synthetic_data_model`
- 295: calls function `init_model_data` in module `model_data_hl7`
- 356: calls function `gen_synthetic_fourier` in module `timeseries`
- 372: calls function `modify_sparse_segments` in module `timeseries`
- 417: calls function `copula_n_variable_model` in module `synthetic_data_model`
- 429: calls function `correct_invalid_pseudopersons` in module `pseudoperson`
- 451: calls function `generate_date_tuples` in module `model_data_hl7`

Module `src.synthetic_data_model`

- 134-149: defines function `init_rng`
  - 144: calls `numpy.random.default_rng` to construct `rng` with initial seed `rng_seed`
- 429-532: defines function `copula_n_variable_model`
  - 494, 499: `scipy.stats.multivariate_normal` creates a "frozen" multivariate normal object that generates random variates through the method `rvs` at line 504[iv]. The `random_state` parameter is initialized with the PRNG created in the call to `init_rng`.

Module `src.model_data_hl7`

- 546-671: defines function `_merge_files`, which calls `rng.shuffle` at line 648 to scramble record order for a given date when combining input files
- 1791-1844: defines function `init_model_data`, which calls `_merge_files` at line 1815
- 2006-2085: defines function `_select_date_tuple`; given sex and case status, selects dates
  - 2039: calls `rng.choice`
  - 2052: calls `rng.uniform`
- 2089-2226: defines function `generate_date_tuples`, which calls `_select_date_tuple` at line 2167

Module `src.timeseries`

- 102-219: defines function `modify_sparse_segments`
  - 167: calls `rng.uniform` to determine whether to modify values in sparse segments
  - 170: calls `rng.uniform` to determine whether to modify values in sparse segments
  - 182: calls `rng.shuffle` to shuffle the zero locations within a given segment
  - 210: calls `rng.shuffle` to shuffle the nonzero locations in the synthetic timeseries
  - 216: calls `rng.shuffle` to ensure at least 2 nonzero values, as needed for kernel density estimation bandwidth calculation
- 223-260: defines function `_adaptive_noise`, which calls `rng.normal` at line 256 to add noise to series
- 264-362: defines function `_add_noise_to_fft`
  - 295: calls `rng.normal` to change the phase of a given Fourier component
  - 359: calls `_adaptive_noise` to generate the adaptive noise and add to the signal
- 366-403: defines function `gen_synthetic_fourier`, which calls `_add_noise_to_fft` at line 391

Module `src.pseudoperson`

- 25-141: defines function `correct_invalid_pseudopersons`, which calls function `copula_n_variable_model` in module `synthetic_data_model` at line 97. The PRNG initialized in the call to `init_rng` is passed in as the `rng` parameter.

[i] The synthetic generator based on NETSS-format data constructs 6 person-level characteristics—the same characteristics as in the HL7-based generator except for pregnancy status.
[ii] Since the NETSS-format data include only 1 date field per notification record, the synthetic generator based on NETSS-format data does not include a component for subsampling pseudoperson-specific date sequences.
[iii] The seeding process should, with high probability, generate a complicated, high-value integer seed ($>2^{32}$). The built-in Python module `secrets` (or `urandom` in module `os`) uses a sophisticated notion of system entropy and ensures that the generated seed value has cryptographically robust properties. The generators use this module to create a random 128-bit seed drawn from the system entropy pool.
[iv] `scipy.stats.multivariate_normal.rvs` is essentially the same as `numpy.random.Generator.multivariate_normal`.