pseudopeople Documentation

Release 1.0.1.dev7+g6d51849

The pseudopeople developers

Apr 16, 2024

CONTENTS

pseudopeople is a Python package that generates realistic simulated data about a fictional United States population, designed for use in testing entity resolution (record linkage) methods or other data science algorithms at scale.

Simulated: These are made-up people! No need to worry about confidentiality.
Versatile: Generate multiple datasets about the same population: censuses, surveys, and administrative records.
✓ **Verifiable**: Ground-truth unique identifiers are present in every dataset for checking link correctness.
Customizable: Configure the levels of noise in each dataset.
Full-scale: Supports generating datasets at the size of the real-life US population.

The fictional US population was generated by stochastically simulating multiple decades of population dynamics such as fertility, mortality, migration and employment. pseudopeople builds on this fictional population data by simulating errors in the data collection process to create realistic, noisy datasets.

pseudopeople is currently in a public beta release. Things are still in flux! If you notice any issues, please let us know on GitHub.

The Simulation Science Team of the University of Washington's Institute for Health Metrics and Evaluation is excited to introduce pseudopeople, the Python package that simplifies entity resolution research and development. This package generates large-scale, simulated population data according to specifications by the user, to replicate a range of complexities found in real applications of probabilistic record linkage software. With sensitive data often required for entity resolution, accessing and testing new methods and software has been a challenge — until now. Our innovative approach creates realistic, simulated data including name, address, and date of birth, without compromising privacy.

Our work builds on the success of previous data synthesis projects, such as FEBRL, GeCo, and SOG, and generates a simulated population using real, publicly accessible data about the US population by leveraging the power of our simulation platform Vivarium.

Want to know more about pseudopeople? Please visit the pseudopeople project website, where you can find out more about the principles and processes underlying this work.

CHAPTER

ONE

QUICKSTART

pseudopeople requires a version of Python between 3.8 and 3.11 (inclusive) to be installed. Once Python is installed, you can install pseudopeople with pip by running the command:

```
$ pip install pseudopeople
```

Or, you can install from source on the pseudopeople GitHub repository.

Then, generate a small-scale simulated decennial census:

```
$ python
```

```
>>> import pseudopeople as psp
>>> census = psp.generate_decennial_census()
>>> census
      simulant_id household_id first_name middle_initial  last_name age date_of_birth  ..
↪. state zipcode housing_type relationship_to_reference_person     sex race_ethnicity ␣
↪year
0             0_2          0_7      Diana              P      Kelly  25    05/06/1994  ..
↪.    WA   00000    Household                Reference person  Female            NaN ␣
↪2020
1             0_3          0_7       Anna              A      Kelly  25    09/29/1994  ..
↪.    WA   00000    Household                  Other relative  Female          White ␣
↪2020
2           0_923       0_8033     Gerald              R      Allen  76    11/03/1943  ..
↪.    WA   00000    Household                Reference person    Male          Black ␣
↪2020
3          0_2641       0_1066    Loretta              T       Lowe  61    06/01/1958  ..
↪.    WA   00000    Household                Reference person  Female          White ␣
↪2020
4          0_2801       0_1138    Richard              R     Pinard  73    03/03/1947  ..
↪.    WA   00000    Household                Reference person    Male          White ␣
↪2020
...           ...          ...        ...            ...        ...  ..           ...  ..
↪.   ...     ...          ...                                ...     ...           ...  ␣
↪...
10215     0_18969       0_7630      Patty              E  Palmisano  87    01/11/1933  ..
↪.    WA   00000    Household             Opposite-sex spouse  Female          White ␣
↪2020
10216     0_19008       0_8361       John              V    Skeeter  58    12/29/1961  ..
↪.    WA   00000    Household                Reference person    Male          Black ␣
↪2020
```

(continues on next page)

(continued from previous page)

```
10217     0_20165       0_7999   Kimberly              K      Suitt  65    04/05/1955  ..
↪.    WA   00000    Household                Reference person  Female          White ␣
↪2020
10218     0_19020       0_8130   Virginia              G     Hoover  93    10/02/1926  ..
↪.    WA   00000    Household                Reference person  Female          White ␣
↪2020
10219     0_20163       0_7998   Victoria              R    Simmons  27    04/21/1992  ..
↪.    WA   00000    Household                Reference person  Female          White ␣
↪2020

[10220 rows x 18 columns]
```

And W-2 and 1099 tax forms from the same simulated population:

```
>>> taxes = psp.generate_taxes_w2_and_1099()
>>> taxes
     simulant_id household_id employer_id          ssn  wages  ... mailing_address_city␣
↪mailing_address_state mailing_address_zipcode tax_form tax_year
0            0_4          0_8          95  584-16-0130  10192  ...              Anytown ␣
↪                  WA                   00000       W2     2020
1            0_5          0_8          29  854-13-6295  28355  ...              Anytown ␣
↪                  WA                   00000       W2     2020
2            0_5          0_8          30  854-13-6295  18243  ...              Anytown ␣
↪                  WA                   00000       W2     2020
3         0_5621       0_2289          46  674-27-1745   7704  ...              Anytown ␣
↪                  WA                   00000       W2     2020
4         0_5623       0_2289          83  794-23-1522   3490  ...              Anytown ␣
↪                  WA                   00000       W2     2020
...          ...          ...         ...          ...    ...  ...                  ... ␣
↪                 ...                     ...      ...      ...
9911      0_18936       0_7621          23  006-92-7857   9585  ...              Anytown ␣
↪                  WA                   00000       W2     2020
9912      0_18936       0_7621          90  006-92-7857  57906  ...              Anytown ␣
↪                  WA                   00000       W2     2020
9913      0_18937       0_7621           1  182-82-5017  19609  ...              Anytown ␣
↪                  WA                      NaN     1099     2020
9914      0_18937       0_7621         105  182-82-5017   8061  ...              Anytown ␣
↪                  WA                   00000     1099     2020
9915      0_18939       0_7621           9  283-97-5940   4961  ...              Anytown ␣
↪                  WA                   00000       W2     2020

[9916 rows x 24 columns]
```

The simulated people in these datasets are called "simulants." Both datasets have a `simulant_id` column that uniquely identifies an individual. The unique `simulant_id` present in both datasets provides us with a truth deck, which we wouldn't have for a linkage task with real, sensitive data.

Note that in the small-scale simulated population that is available by default, these addresses all have their city/state/zip code set to the fictitious location of Anytown, WA 00000. To read more about obtaining large-scale data with more realistic city, state, and zip code data, please see *Simulated populations*.

```
>>> # To find how many matches there are between records about a given simulant,
>>> # we need to multiply the number of records about that simulant in the census by
```

(continues on next page)

(continued from previous page)

```
>>> # the number of records about that simulant in taxes
>>> true_matches = census.groupby("simulant_id").size().multiply(
...     taxes.groupby("simulant_id").size(), fill_value=0
... ).sum().astype(int)
>>> print(f"There are {true_matches:,} true matches to find between these datasets!")
There are 9,034 true matches to find between these datasets!
```

Now, see how many your record linkage method can find – without access to the truth deck, of course!

Not linking in Python? Just save your datasets as files, for example CSV files:

```
>>> census.to_csv('census.csv')
>>> taxes.to_csv('taxes.csv')
```

Now you can load these datasets in any environment that can read CSV.

CHAPTER

TWO

WHAT'S NEXT?

Now that you've generated a simulated dataset with pseudopeople, here are some next steps:

- To get started with customizing the noise in your datasets, try out the *tutorial on configuring noise*.
- To learn more about the kinds of simulated datasets that are available, check out our *Datasets page*.
- If you need larger datasets with millions instead of thousands of rows, take a look at the *Simulated populations page*.
- To dive deeper into noise, read the docs about *noise* and *noise configuration*.
- To stay informed and recieve updates about this software package join the mailing list here.

2.1 Tutorials

Here you'll find a set of tutorials that provide step-by-step instructions for common tasks using the `pseudopeople` package.

2.1.1 Configuring Noise

In this tutorial, we will walk through an example of how to customize the amount of noise in a simulated dataset generated by pseudopeople.

If you haven't already used pseudopeople to generate a dataset with the default settings, follow the *Quickstart* before continuing with this tutorial.

The problem of fake names

Sometimes when people respond to a survey, they don't want to share their personal information. If the survey (whether online, on paper, or in person) requires a response, they might just make something up.

pseudopeople has a noise type to simulate these sorts of responses for first and/or last names: *"Use a fake name."* With pseudopeople's default settings, this happens just 1% of the time. But let's say we're concerned it will happen more often in the future, and we want to see how robust our entity resolution methods are to this issue.

Generating a simulated Current Population Survey

Let's generate some 2025 Current Population Survey (CPS) data in a future scenario where 30% of people use fake last names on the survey. To do this, we will pass some **configuration** to the *pseudopeople.generate_current_population_survey()* function.

```
cps_2025 = psp.generate_current_population_survey(year=2025, config={
    'current_population_survey': {
        'column_noise': {
            'last_name': {
                'use_fake_name': {
                    'cell_probability': 0.3,
                },
            },
        },
    },
})
```

In English, our configuration says: in the **CPS** dataset, for the **last name** column, the **fake name** noise type's **cell probability** parameter should be 0.3 (30%). The full set of parameters for each noise type is documented at *Noise Type Details*. The `column_noise` key specifies that we are configuring column-based noise; the categories of noise are explained in more detail on the *noise page*.

Let's take a look at the names in our generated CPS dataset:

```
>>> cps_2025[['first_name', 'middle_initial', 'last_name']]
   first_name middle_initial          last_name
0     Gregory              J               Four
1   Bridgette              J            Kennedy
2     William              G           Phillips
3     Valerie              M               Male
4       Molly              A            Wheeler
5      Thomas              K             Eastep
6     Kenneth              C             Harper
7      Daniel              M             Harper
8       Susan              M              Adult
9     Dorothy              P             Gaytan
10      Daisy              R           Williams
11       Mark              T               Rock
12    Mohamed              C             Person
13    Giselle              L              Weber
14       Jean              F              Stull
15       Lila              L                  C
16      Carli              G            Mckamey
17     Justin              B                  E
18        Ana              S  Davidson Granados
19       Rose              K           Carrillo
20     Nayeli              A           Carrillo
21     Robert              O           Carrillo
22     Emilio              P           Carrillo
23      Mindy              K             Walton
24        Lee              J          Household
25  Janautica              K            Clapper
26     Tanner              M            Clapper
```

(continues on next page)

(continued from previous page)

```
27      Helen              K        Of The Home
28      Tonya              A                  Y
29      Chris              C             Frazier
30     Arnold              J              Friend
31   Patricia              C                Wife
32     Jessie              L              Madden
33      Laura              V          Fortenberry
34       Zaid              B          Fortenberry
35      Sammy              R            Mcfarlan
```

As expected, we see a number of strings in the last name column that are unlikely to be true last names. We can check exactly which ones are fake names by comparing to the same dataset without fake name noise in the last name column. For brevity, we do not show the steps to do this here, but we would find that there are eleven such strings, which is almost exactly 30% of our 36 respondents.

Increasing noise in first names

Imagine we also want to increase the probability of a fake first name from its default of 1%. We can do this by modifying the configuration dictionary. This time, we'll save the configuration dictionary to a variable before using it to generate the dataset:

```
config = {
    'current_population_survey': {
        'column_noise': {
            'last_name': {
                'use_fake_name': {
                    'cell_probability': 0.3,
                },
            },
            'first_name': {
                'use_fake_name': {
                    'cell_probability': 0.2,
                },
            },
        },
    },
}
cps_2025 = psp.generate_current_population_survey(year=2025, config=config)
```

By specifying multiple keys within `column_noise`, we are able to independently adjust noise settings for different columns. Here we have set the probability of a fake first name to 0.2 (20%) while retaining the 0.3 (30%) probability of a fake last name. Let's see how our CPS data look now:

```
>>> cps_2025[['first_name', 'middle_initial', 'last_name']]
       first_name middle_initial          last_name
0         Gregory              J               Four
1       Bridgette              J            Kennedy
2         William              G           Phillips
3         Valerie              M               Male
4           Molly              A            Wheeler
5          Thomas              K             Eastep
6      Man In The              C             Harper
```

(continues on next page)

(continued from previous page)

```
7             Man              M             Harper
8               R              M              Adult
9   Granddaughter              P             Gaytan
10              G              R           Williams
11           Mark              T               Rock
12           Girl              C             Person
13        Giselle              L              Weber
14         Friend              F              Stull
15           Lila              L                  C
16          Carli              G            Mckamey
17         Justin              B                  E
18            Ana              S  Davidson Granados
19           Rose              K           Carrillo
20         Son Of              A           Carrillo
21         Sister              O           Carrillo
22         Emilio              P           Carrillo
23          Mindy              K             Walton
24            Lee              J          Household
25      Janautica              K            Clapper
26         Tanner              M            Clapper
27          Helen              K        Of The Home
28          Tonya              A                  Y
29          Chris              C            Frazier
30         Arnold              J             Friend
31        Brother              C               Wife
32          House              L             Madden
33              T              V        Fortenberry
34           Zaid              B        Fortenberry
35              H              R          Mcfarlane
```

Here we see that 13 respondents have used fake first names. Why aren't there closer to $0.2 * 36 = 7.2$ respondents with fake first names? Remember that the parameter we set was called cell **probability** – there is randomness involved in whether or not each cell in the column actually receives noise.

An alternate format for configuration

It is also possible to specify configuration in a YAML file. The file equivalent to our final configuration above would be:

Listing 1: configuration_example.yaml

```yaml
current_population_survey:
  column_noise:
    last_name:
      use_fake_name:
        cell_probability: 0.3
    first_name:
      use_fake_name:
        cell_probability: 0.2
```

If `configuration_example.yaml` is in the current working directory, it can be used like so:

```
cps_2025 = psp.generate_current_population_survey(year=2025, config='configuration_
↪example.yaml')
```

For more on configuration, see the *Configuration page*.

2.2 Datasets

Here we cover the realistic simulated datasets, which are analogous to "real world" administrative records such as tax documents and routinely generated files of social security numbers, that users can generate using Pseudopeople for developing and testing Entity Resolution algorithms and software.

Each of the datasets that can be generated using pseudopeople has "noise" added to it, thereby realistically simulating how population data can be corrupted or distorted, which creates challenges in linking those records. To read more about the different kinds of noise that can be applied to the different datasets, please see the *Noise page*.

pseudopeople generates datasets about a single simulated US population, which is followed through time between January 1st, 2019 and May 1st, 2041. Most datasets are yearly and can be generated for any year between 2019 and 2041 (inclusive), though 2041 data will be partial.

There are two kinds of street addresses present in pseudopeople datasets: physical addresses and mailing addresses. A **physical address** represents the physical location where a simulant lives, which is where they are recorded in the Decennial Census and surveys. A **mailing address** represents the address a simulant uses to receive mail, which may be different – for example, a PO box. Mailing addresses, not physical addresses, are recorded in tax filings.

Note that in the small-scale simulated population that is available by default, these addresses all have their city/state/zip code set to the fictitious location of Anytown, WA 00000. This is to ensure that linking is not unrealistically easy with the sample population (i.e., using these attributes to eliminate clear non-matches is not possible, as they are all identical). To read more about obtaining large-scale data with more realistic city, state, and zip code data, please see *Simulated populations*.

Some fields are not applicable to every record in a simulated dataset, so some columns may contain "missing" values, even if no noise has been added to the data. For example, most addresses do not have a unit number, and some do not have a street number, so the `unit_number` and/or `street_number` fields will be "missing" for many rows in any dataset that contains addresses. Similarly, columns pertaining to spouse or dependents in the 1040 tax dataset are not applicable to every simulant, so these columns also contain missing values. Values that are missing because they are not applicable are represented by `numpy.nan`.

The datasets that can be generated are listed below.

- *US Decennial Census*
- *American Community Survey (ACS)*
- *Current Population Survey (CPS)*
- *Women, Infants, and Children (WIC)*
- *Social Security Administration*
- *Tax forms: W-2 & 1099*
- *Tax form: 1040*

2.2.1 US Decennial Census

The Decennial Census dataset is a simulated enumeration of the US Census Bureau's Decennial Census of Population and Housing. To find out more about the Decennial Census, please visit the Decennial Census homepage.

It is only possible to generate Decennial Census data for decennial years – 2020, 2030, and 2040.

Generate Decennial Census data with *pseudopeople.generate_decennial_census()*.

The following columns are included in this dataset:

Table 1: **Dataset columns**

Attribute Name	Column Name	Notes
Unique simulant ID	simulant	Not affected by noise; intended use is "ground truth" for testing and validation; consistent across all datasets.
Unique household ID	household	Not affected by noise; intended use is "ground truth" for testing and validation; consistent across all datasets.
First name	first_na	
Middle initial	middle_i	
Last name	last_nam	
Age	age	Rounded down to an integer.
Date of birth	date_of_	Formatted as MM/DD/YYYY.
Physical address street number	street_n	
Physical address street name	street_n	
Physical address unit number	unit_num	
Physical address city	city	Default simulated population always has value "Anytown"
Physical address state	state	Default simulated population always has value "WA"
Physical address ZIP code	zipcode	Default simulated population always has value "00000"

2.2.2 American Community Survey (ACS)

ACS is one of two household surveys that can currently be simulated using Pseudopeople. ACS is an ongoing household survey conducted by the US Census Bureau that gathers information on a rolling basis about American community populations. Information collected includes ancestry, citizenship, education, income, language proficienccy, migration, employment, disability, and housing characteristics. To find out more about ACS, please visit the ACS homepage.

pseudopeople can generate ACS data for a user-specified year, which will include records from simulated surveys conducted throughout that calendar year.

Generate ACS data with *pseudopeople.generate_american_community_survey()*.

The following columns are included in this dataset:

Table 2: **Dataset columns**

Attribute Name	Column Name	Notes
Unique simulant ID	simulant	Not affected by noise; intended use is "ground truth" for testing and validation; consistent across all datasets.
Unique household ID	household	Not affected by noise; intended use is "ground truth" for testing and validation; consistent across all datasets.
Survey date	survey_d	Date on which the survey was conducted; metadata that would not be collected directly; not affected by noise. Stored as a `pandas.Timestamp`, which displays in YYYY-MM-DD format by default.
First name	first_na	
Middle initial	middle_i	
Last name	last_nam	
Age	age	Rounded down to an integer.
Date of birth	date_of_	Formatted as MM/DD/YYYY.
Physical address street number	street_n	
Physical address street name	street_n	
Physical address unit number	unit_num	
Physical address city	city	Default simulated population always has value "Anytown"
Physical address state	state	Default simulated population always has value "WA"
Physical address zipcode	zipcode	Default simulated population always has value "00000"
Physical ad		

2.2.3 Current Population Survey (CPS)

CPS is another household survey that can be simulated using Pseudopeople. CPS is conducted jointly by the US Census Bureau and the US Bureau of Labor Statistics. CPS collects labor force data, such as annual work activity and income, veteran status, school enrollment, contingent employment, worker displacement, job tenure, and more. To find out more about CPS, please visit the CPS homepage.

pseudopeople can generate CPS data for a user-specified year, which will include records from simulated surveys conducted throughout that calendar year.

Generate CPS data with *pseudopeople.generate_current_population_survey()*.

The following columns are included in this dataset:

Table 3: **Dataset columns**

Attribute Name	Column Name	Notes
Unique simulant ID	`simula`	Not affected by noise; intended use is "ground truth" for testing and validation; consistent across all datasets.
Unique household ID	`househ`	Not affected by noise; intended use is "ground truth" for testing and validation; consistent across all datasets.
Survey date	`survey`	Date on which the survey was conducted; metadata that would not be collected directly; not affected by noise. Stored as a `pandas.Timestamp`, which displays in YYYY-MM-DD format by default.
First name	`first_`	
Middle initial	`middle`	
Last name	`last_n`	
Age	`age`	Rounded down to an integer.
Date of birth	`date_o`	Formatted as MM/DD/YYYY.
Physical address street number	`street`	
Physical address street name	`street`	
Physical address unit number	`unit_n`	
Physical address city	`city`	Default simulated population always has value "Anytown"
Physical address state	`state`	Default simulated population always has value "WA"
Physical address ZIP code	`zipcod`	Default simulated population always has value "00000"
Sex	`sex`	Binary; "male" or "female"
Race/ethnici	`race_e`	The categories for the single composite "race/ethnicity" field are as follows: "White"; "Black"; "Latino"; "American Indian and Alaskan Native (AIAN)"; "Asian"; "Native Hawaiian and Other Pacific Islander (NHOPI)"; and "Multiracial or Some Other Race".

2.2.4 Women, Infants, and Children (WIC)

The Special Supplemental Nutrition Program for Women, Infants, and Children (WIC) is a government benefits program designed to support mothers and young children. The main qualifications are income and the presence of young children in the home. To find out more about this service, please visit the WIC homepage.

pseudopeople can generate a simulated version of the administrative data that would be recorded by WIC. This is a yearly file of information about all simulants enrolled in the program as of the end of that year. For the final year available, 2041, the file includes those enrolled as of May 1st, because this is the end of our simulated timespan.

Generate WIC data with *pseudopeople.generate_women_infants_and_children()*.

The following columns are included in this dataset:

Table 4: **Dataset columns**

Attribute Name	Column Name	Notes
Unique simulant ID	simula	Not affected by noise; intended use is "ground truth" for testing and validation; consistent across all datasets.
Unique household ID	househ	Not affected by noise; intended use is "ground truth" for testing and validation; consistent across all datasets.
First name	first_	
Middle initial	middle	
Last name	last_n	
Date of birth	date_o	Formatted as MMDDYYYY.
Physical address street number	street	
Physical address street name	street	
Physical address unit number	unit_n	
Physical address city	city	Default simulated population always has value "Anytown"
Physical address state	state	Default simulated population always has value "WA"
Physical address ZIP code	zipcod	Default simulated population always has value "00000"
Sex	sex	Binary; "male" or "female"
Race/ethnici	race_e	The categories for the single composite "race/ethnicity" field are as follows: "White"; "Black"; "Latino"; "American Indian and Alaskan Native (AIAN)"; "Asian"; "Native Hawaiian and Other Pacific Islander (NHOPI)"; and "Multiracial or Some Other Race".
Year	year	Year in which benefits were received; metadata that would not be collected directly; not affected by noise.

2.2.5 Social Security Administration

The Social Security Administration (SSA) is the US federal government agency that administers Social Security, the social insurance program that consists of retirement, disability and survivor benefits. To find out more about this program, visit the SSA homepage.

pseudopeople can generate a simulated version of a subset of the administrative data that would be recorded by SSA. Currently, the simulated SSA data includes records of SSN creation and dates of death. This is a yearly data file that is **cumulative** – when you specify a year, you will recieve all records *up to the end of* that year.

The simulated SSA data files will not include records about simulants who died before 2019 (the start of our simulated timespan). Therefore, while SSA data files can be generated for years prior to 2019, they will only include records for SSN creation, and only for simulants who were still alive in 2019.

Generate SSA data with *pseudopeople.generate_social_security()*.

The following columns are included in this dataset:

Table 5: **Dataset columns**

Attribute Name	Column Name	Notes
Unique simulant ID	simulant_	Not affected by noise; intended use is "ground truth" for testing and validation; consistent across all datasets.
Social security number	ssn	By default, the SSN column in the SSA dataset has no *column-based noise*. However, it can be *configured* to have noise if desired.
First name	first_name	
Middle name	middle_nar	
Last name	last_name	
Date of birth	date_of_b	Formatted as YYYYMMDD.
Sex	sex	Binary; "male" or "female"
Type of event	event_type	Possible values are "Creation" and "Death".
Date of event	event_date	Formatted as YYYYMMDD.

2.2.6 Tax forms: W-2 & 1099

Administrative data reported in annual tax forms, such as W-2s and 1099s, can also be simulated by Pseudopeople. 1099 forms are used for independent contractors or self-employed individuals, while a W-2 form is submitted by an employer for their employee (as the employer withholds payroll taxes from employee earnings).

pseudopeople can generate a simulated version of the data collected by W-2 and 1099 forms. This is a yearly dataset, where the user-specified year is the **tax year** of the data. That is, the data for 2022 will be the result of tax forms filed in early 2023. Tax data can be generated for tax years 2019 through 2040 (inclusive).

Generate W-2 and 1099 data with *pseudopeople.generate_taxes_w2_and_1099()*.

The following columns are included in these datasets:

Table 6: **Dataset columns**

Attribute Name	Column Name	Notes
Unique simulant ID	simulant_id	Not affected by noise; intended use is "ground truth" for testing and validation; consistent across all datasets.
Unique household ID	household_id	Not affected by noise; intended use is "ground truth" for testing and validation; consistent across all datasets.
Employer ID	employer_id	
Social security number	ssn	
Wages	wages	
Employer Name	employer_name	
Employer street number	employer_street_num	
Employer street name	employer_street_nam	
Employer unit number	employer_unit_numbe	
Employer city	employer_city	Default simulated population always has value "Anytown"
Employer state	employer_state	Default simulated population always has value "WA"
Employer ZIP code	employer_zipcode	Default simulated population always has value "00000"
First name	first_name	
Middle initial	middle_initial	
Last name	last_name	
Mailing address street number	mailing_address_str	
Mailing address street name	mailing_address_str	
Mailing address unit number	mailing_address_uni	
Mailing address PO Box	mailing_address_po_	
Mailing address city	mailing_address_cit	Default simulated population always has value "Anytown"
Mailing address state	mailing_address_sta	Default simulated population always has value "WA"
Mailing address ZIP code	mailing_address_zip	Default simulated population always has value "00000"
Type of tax form	tax_form	Possible values are "W2" or "1099".
Tax year	tax_year	Year for which tax data were collected; metadata that would not be collected directly; not affected by noise.

2.2.7 Tax form: 1040

As with data collected from W-2 and 1099 forms, pseudopeople enables the simulation of administrative records from 1040 forms, which are also reported to the IRS on an annual basis. While W-2 forms are submitted by an employer to the IRS, 1040 forms are submitted by the employee. To find out more about the 1040 tax form, visit the IRS information page.

A single row in a pseudopeople-generated 1040 dataset may contain information about several simulants: the primary filer, the primary filer's joint filer (spouse) if they are married filing jointly, and up to four claimed dependents. When not applicable, all relevant fields are `numpy.nan`; for example, a row representing a 1040 filed by only one simulant, without a joint filer, would have missingness in all joint filer columns.

If a simulant claims fewer than four dependents, they will be filled in starting with `dependent_1`. For example, a simulant claiming three dependents would have missingness in all `dependent_4` columns. A simulant may claim more than four dependents, but only four will appear in the dataset; the rest are omitted.

All columns not otherwise labeled are about the primary filer; for example, the `first_name` column is the first name of the primary filer. The `simulant_id` and `household_id` columns represent the "ground truth" of which simulant is the primary filer, and which household *that primary filer* lives in. It is not guaranteed that all simulants described in a 1040 row live in the same household; for example, college students may be claimed as dependents while living elsewhere.

A single simulant can appear in multiple rows in this dataset, for example if they filed a 1040 and were also claimed as a dependent on another simulant's 1040.

This is a yearly dataset, where the user-specified year is the **tax year** of the data. 1040 data can be generated for tax years 2019 through 2040 (inclusive).

Generate 1040 data with *`pseudopeople.generate_taxes_1040()`*.

The following columns are included in this dataset:

Table 7: **Dataset columns**

Attribute Name	Column Name	Notes
Unique simulant ID	`simulant_id`	Not affected by noise; intended use is "ground
Unique household ID	`household_id`	Not affected by noise; intended use is "ground
First name	`first_name`	
Middle initial	`middle_initial`	
Last name	`last_name`	
Social Security Number (SSN)	`ssn`	Individual Taxpayer Identification Number (IT
Mailing address street number	`mailing_address_street_number`	
Mailing address street name	`mailing_address_street_name`	
Mailing address unit number	`mailing_address_unit_number`	
Mailing address PO box	`mailing_address_po_box`	
Mailing address city	`mailing_address_city`	Default simulated population always has value
Mailing address state	`mailing_address_state`	Default simulated population always has value
Mailing address ZIP code	`mailing_address_zipcode`	Default simulated population always has value
Joint filer first name	`spouse_first_name`	
Joint filer middle initial	`spouse_middle_initial`	
Joint filer last name	`spouse_last_name`	
Joint filer social security number	`spouse_ssn`	Individual Taxpayer Identification Number (IT
Dependent 1 first name	`dependent_1_first_name`	
Dependent 1 last name	`dependent_1_last_name`	
Dependent 1 Social Security Number (SSN)	`dependent_1_ssn`	Individual Taxpayer Identification Number (IT
Dependent 2 first name	`dependent_2_first_name`	

Table 7 – continued from previous page

Attribute Name	Column Name	Notes
Dependent 2 last name	`dependent_2_last_name`	
Dependent 2 social security number	`dependent_2_ssn`	Individual Taxpayer Identification Number (IT
Dependent 3 first name	`dependent_3_first_name`	
Dependent 3 last name	`dependent_3_last_name`	
Dependent 3 social security number	`dependent_3_ssn`	Individual Taxpayer Identification Number (IT
Dependent 4 first name	`dependent_4_first_name`	
Dependent 4 last name	`dependent_4_last_name`	
Dependent 4 social security number	`dependent_4_ssn`	Individual Taxpayer Identification Number (IT
Tax year	`tax_year`	Year for which tax data were collected; metad

2.3 Simulated populations

pseudopeople generates multiple *datasets* about a simulated population which can be specified by the user when calling the *dataset generation functions*. There are currently three simulated populations available for generating datasets with pseudopeople:

- **Sample population** (a fictional population of ~10,000 simulants living in the fictional Anytown, WA, included with the pseudopeople package)
- **Rhode Island** (a fictional population of ~1,000,000 simulants living in a simulated state of Rhode Island)
- **United States** (a fictional population of ~330,000,000 simulants living throughout a simulated United States)

When generating a dataset, pseudopeople uses the included sample population by default unless an explicit path to another directory containing simulated population data is specified. See the sections below for more information about accessing and using the larger simulated populations.

Note: The simulated population data used by pseudopeople is the output of a Vivarium microsimulation and must be in a specific format for the dataset generation functions to work. Vivarium uses real, publicly available data to stochastically simulate multiple decades of population dynamics such as fertility, mortality, migration, and employment. Then pseudopeople takes the simulated population data output by Vivarium and simulates the data collection process with user-configurable *noise* added to the resulting datasets.

2.3.1 Accessing the large-scale simulated populations

To gain access to the larger-scale simulated populations (i.e., Rhode Island and United States), follow these steps:

1. Log in to GitHub (you must first create a GitHub account if you don't have one).
2. Open a new Data access request using the template under the Issues tab on pseudopeople's GitHub page.
3. Fill out the information on the access request form to tell us about your project. You can simply put "Data access request" in the title field.
4. We will get back to you after we receive your request!

2.3.2 Validating the simulated population data

Checksums can be used to validate that you've successfully downloaded the correct and uncorrupted zip file. The following table provides the SHA-256 checksum for the larger-scale simulated population zip files:

Table 8: SHA-256 checksums

Location	File	SHA-256 checksum
Rhode Island	pseudopeople_simulated_population_ri_2_0_1.zip	fadcbf40c87217f77f36f2c684a6a568460a1215696bc2f8a0c2069a00cdc78c
US	pseudopeople_simulated_population_usa_2_0_0.zip	0025978196c2a84c1df502e857bec35a84c25092fbfb6b143c0b8ff30dea5eed
Rhode Island	pseudopeople_simulated_population_ri_2_0_0.zip	bfec148c947096b44201a7961a1b38f63961cd820578f10a479f623d8d79f0d1
US	pseudopeople_simulated_population_usa_1_0_0.zip	9462cc60b333fb2a3d16554a9e59b5428a81a2b1d2c34ed383883d7b68d2f89f
Rhode Island	pseudopeople_simulated_population_ri_1_0_0.zip	d3f1ccdfbfca8b53254c4ceeb18afe17c3d3b3fe02f56cc20d1254f818c39435

If the SHA-256 checksum that you generate for the downloaded file matches the value provided above, you can be sure you downloaded the file successfully.

Possibly the simplest way to verify checksums is to generate the value using the terminal/cmd command below (be sure to replace *PATH/TO/ZIP* with the actual path to the zip you downloaded) and visually compare the result to the values provided above. Note that if even the first few and last few characters match then it is very likely the entire string matches.

Linux:

```
$ sha256sum PATH/TO/ZIP
```

Mac:

```
$ shasum -a 256 PATH/TO/ZIP
```

Windows:

```
$ CertUtil -hashfile PATH/TO/ZIP SHA256
```

Note: Generating the checksum can take a long time for larger files, e.g. several minutes for the Rhode Island dataset and ~1 hour for the United States dataset.

If the generated checksum does not match the one provided in the table above, please try re-downloading the dataset.

If after downloading the file a second time the checksums still do not match, please open a Bug report using the template under the Issues tab on pseudopeople's GitHub page.

2.3.3 Using the simulated population data

Once you've downloaded the large-scale simulated population (either Rhode Island or United States), unzip the contents to the desired location on your computer.

Important: Do not modify the contents of the directory containing the unzipped simulated population data! Modifications to the pseudopeople simulated population data may cause the dataset generation functions to fail.

Once you've unzipped the simulated population data, you can pass the directory path to the `source` parameter of the *dataset generation functions* to generate large-scale datasets!

2.4 Noise

In order to have a realistic challenge with entity resolution, it is essential to add noise to the simulated data. "Noise" refers to various types of errors introduced into the data and may also be called "corruption" or "distortion." By default, pseudopeople applies noise to the simulated datasets using some reasonable settings. If desired, the user can change the noise settings through the configuration system—see the *Configuration section* for details.

2.4.1 Categories of noise

pseudopeople can add two broad categories of noise to the datasets it generates:

1. **Row-based noise:** Errors in the inclusion or exclusion of entire rows of data, such as duplication or omission
2. **Column-based noise:** Errors in data entry for an individual field within a row, such as miswriting or incorrectly selecting responses

Each type of row-based noise operates on the entire dataset (selecting rows to include or exclude), while each type of column-based noise operates on one column of data at a time (selecting cells within that column to noise). Currently, errors added in different columns are independent of each other.

2.4.2 Available noise types

These tables list all the available noise types, but not every type of noise will necessarily be applied to every dataset or every column. Noise types are applied in the order they are listed here. The "Config key" column shows the name of the noise type in the *configuration system*.

Table 9: Types of row-based noise (`row_noise`)

Noise type	Config key	Example cause
Duplicate with guardian	`duplicate_with_guardi`	Filling out a survey questionnaire for your child that lives in college housing, while they simultaneously fill out the same questionnaire for themselves
Do not respond	`do_not_respond`	Not returning the American Community Survey questionnaire that the US Census Bureau sent you
Omit a row	`omit_row`	Losing data because of an administrative error

Table 10: Types of column-based noise (`column_noise`)

Noise type	Config key	Example cause
Borrow a social security number	Not configurable	Using your housemate's SSN on a W-2 because you do not have one
Leave a field blank	`leave_blank`	Forgetting to write your name on the designated line
Choose the wrong option	`choose_wrong_option`	Marking the "Male" box when you meant "Female"
Copy from household member	`copy_from_household_m`	Accidentally writing your daughter's age in a box that asked about your son's age on a survey questionnaire
Use a nickname	`use_nickname`	Writing 'Alex' instead of legal name 'Alexander'
Use a fake name	`use_fake_name`	Using "Mr" rather than actual first name
Swap month and day	`swap_month_and_day`	Reporting 17/05/1976 when a survey asks for the date in MM/DD/YYYY format
Misreport age	`misreport_age`	Reporting that you are 28 years old when you are actually 27
Write the wrong digits	`write_wrong_digits`	Writing "732 Main St" as your street address instead of "932 Main St"
Write the wrong ZIP code digits	`write_wrong_zipcode_d`	Writing ZIP code 98118 when you actually live in 98112
Make phonetic errors	`make_phonetic_errors`	Mishearing a 't' for a 'd'
Make Optical Character Recognition (OCR) errors	`make_ocr_errors`	Misreading an 'S' instead of a '5'
Make typos	`make_typos`	Accidentally typing an "l" instead of a "k" because they are right next to each other on a QWERTY keyboard

2.4.3 Noise types for each column

Table 11: Types of noise for each column

Column name	Applicable datasets	Types of noise	Notes
First name	Decennial Census, ACS, CPS, WIC, W-2 and 1099, 1040, SSA	Leave a field blank, use a nickname, use a fake name, make phonetic errors, make OCR errors, make typos	In the 1040 form, the same noise types apply to the first name columns for the joint filer and dependents
Middle name	SSA	Leave a field blank, use a nickname, use a fake name, make phonetic errors, make OCR errors, make typos	Middle names use the same lists of nicknames and fake names used for first names
Middle initial	Decennial Census, ACS, CPS, WIC, W-2 and 1099, 1040	Leave a field blank, make phonetic errors, make OCR errors, make typos	In the 1040 form, the same noise types apply to the middle initial columns for the joint filer and dependents
Last name	Decennial Census, ACS, CPS, WIC, W-2 and 1099, 1040, SSA	Leave a field blank, use a fake name, make phonetic errors, make OCR errors, make typos	Last names use a different list of fake names than the list for first names. In the 1040 form, the same noise types apply to the last name columns for the joint filer and dependents
Age	Decennial Census, ACS, CPS	Leave a field blank, copy from household member, misreport age, make OCR errors, make typos	
Date of birth	Decennial Census, ACS, CPS, WIC, SSA	Leave a field blank, copy from household member, swap month and day, write the wrong digits, make OCR errors, make typos	
Street number for any address (physical, mailing, or employer)	Decennial Census, ACS, CPS, WIC, W-2 and 1099, 1040	Leave a field blank, write the wrong digits, make OCR errors, make typos	Noise for all types of addresses works in the same way
Street name for any address (physical, mailing, or employer)	Decennial Census, ACS, CPS, WIC, W-2 and 1099, 1040	Leave a field blank, make phonetic errors, make OCR errors, make typos	Noise for all types of addresses works in the same way
Unit number for any address (physical, mailing, or employer)	Decennial Census, ACS, CPS, WIC, W-2 and 1099, 1040	Leave a field blank, write the wrong digits, make OCR errors, make typos	Noise for all types of addresses works in the same way
PO Box for mailing address	W-2 and 1099, 1040	Leave a field blank, write the wrong digits, make OCR errors, make typos	
City name for any address (physical, mailing, or employer)	Decennial Census, ACS, CPS, WIC, W-2 and 1099, 1040	Leave a field blank, make phonetic errors, make OCR errors, make typos	Noise for all types of addresses works in the same way
State for any address (physical, mailing, or employer)	Decennial Census, ACS, CPS, WIC, W-2 and 1099, 1040	Leave a field blank, choose the wrong option	Noise for all types of addresses works in the same way
ZIP code for any address (physical, mailing, or employer)	Decennial Census, ACS, CPS, WIC, W-2 and 1099, 1040	Leave a field blank, write the wrong zipcode digits, make OCR errors, make typos	
Housing type	Decennial Census, ACS	Leave a field blank, choose the wrong option	

2.4.4 Noise type details

For more details on the different row-based and column-based noise types covered above, please follow the links below.

Row-based Noise

Row-based noise operates on one row of data at a time, for example by omitting or duplicating entire rows.

Types of row-based noise:

Duplicate with guardian

A known challenge in entity resolution is people being reported multiple times at different addresses. This can occur when family structures are complex and children spend time at multiple households. A related challenge occurs with college students, who are sometimes counted both at their university and at their guardian's home address.

To simulate such challenges, pseudopeople can apply this type of duplication to two mutually exclusive categories of simulants based on age and GQ status: Simulants younger than 18 and not in GQ and simulants under 24 and in college GQ.

For each of the two categories of simulants, a maximum duplication rate will be calculated based on those who have a guardian living at a different address. Most simulants in college GQ will have a guardian at a different address, but most simulants younger than 18 will not. If you as the user select a duplication rate that is higher than the calculated maximum rate, you will see a warning that the requested rate is greater than the maximum possible.

This noise type is called `duplicate_with_guardian` in the configuration. It takes two parameters:

Table 12: Parameters to the duplicate_with_guardian noise type

Parameter	Description	Default
`row_proba`	The probability that a simulant under 18 in a household is recorded twice.	0.02 (2%)
`row_proba`	The probability that a simulant under 24 in college GQ is recorded twice.	0.05 (5%)

Do not respond

Sometimes people don't respond to a census or survey questionnaire and are unable to be reached by other means such as telephone or personal visit. For the Decennial Census and household surveys such as the ACS and CPS, people are found to respond at different rates depending on demographics such as age, sex, and race or ethnicity.

For each demographic subgroup in pseudopeople, we assumed the nonresponse rate in the Decennial Census was equal to the *net* rate of undercount (ignoring duplication) estimated in [Census_PES]. Net undercount effects of age/sex were combined additively with the effects of race/ethnicity, and demographic subgroups with resulting net overcounts (negative nonresponse rates) were given a nonresponse rate of 0. We assumed nonresponse in the ACS was the same as in the Decennial Census, since these are conducted similarly. By contrast, the CPS uses only phone calls and personal

visits but not mail/online questionnaires. Thus we assumed that CPS had the same nonresponse pattern as the Decennial Census, but with a constant 27.6% added to the nonresponse rates due to this survey having fewer contact modes, as that was the nonresponse rate for CPS reported for July 2022 in [Response_Rates_BLS].

To simulate nonresponse bias in the Decennial Census and the ACS or CPS, the user can choose an overall rate of nonresponse, and pseudopeople will scale the nonresponse rates for different demographic subgroups so that the overall average rate approximately matches the target. The default overall rates were calculated from our simulated population after applying the nonresponse rates derived from [Census_PES] and [Response_Rates_BLS] to each demographic subgroup as described above.

This noise type is called `do_not_respond` in the configuration. It takes one parameter:

Table 13: Parameters to the do_not_respond noise type

Parameter	Description	Default
`row_proba`	The probability that a simulant does not respond to the census or survey.	• 0.0145 (1.45%) for the Decennial Census and ACS • 0.2905 (29.05%) for CPS

Omit a row

Sometimes an entire record may be missing from a dataset where one would normally expect to find it. For example, a WIC record could be missing by mistake because of an administrative error, or someone's tax record could be missing because they didn't file their taxes on time.

This noise type is called `omit_row` in the configuration. It takes one parameter:

Table 14: Parameters to the omit_row noise type

Parameter	Description	Default
`row_proba`	The probability that a row is missing from the dataset.	• 0.005 (0.5%) for WIC and tax forms W2 and 1099 • 0.0 (0%) for other datasets

When applying `omit_row` noise, each row of data is selected for omission independently with probability `row_probability`.

Column-based Noise

Column-based noise operates on one column of data at a time, introducing errors to individual cells within the column.

In pseudopeople, column-based noise is currently performed independently between columns. For example, a simulant making a typo in their first name on the decennial Census doesn't make them any more likely to make a typo in their last name as well, even though in real life we might expect these things to be related.

Column-based noise is also unrelated to attributes of the simulant or record. For example, a simulant who is 20 is just as likely to misreport their age as a simulant who is 75.

These are both areas where we may add more complexity in future versions of pseudopeople.

Types of column-based noise:

Borrow a social security number

The W-2 and 1099 tax forms require a Social Security Number (SSN). Many people who are employed in the US do not have an SSN, but they or their employer still file W-2 or 1099 forms, presumably using someone else's SSN or a made-up SSN.

As a simple way to replicate this behavior, when a simulant without an SSN has a W-2 or 1099 filed, pseudopeople uses an SSN borrowed from a randomly selected simulant in their household who does have one. If there is nobody in their household with an SSN, a totally random SSN is created and used on the form.

This type of noise cannot be configured. It is always present on all W-2 and 1099 forms about a simulant who does not have an SSN.

Leave a field blank

Often some of the data in certain columns of a dataset will be missing. This could be because the input for that field was left blank, an answer was refused, or the answer was illegible or unintelligible. To simulate this type of noise, pseudopeople will replace the value in the relevant cell with `numpy.nan` to indicate that the value is missing.

It is important to note, however, that some columns in the generated data may contain missing values, even if no noise has been added to the data, simply because the column is not applicable to every row. Columns that may have missing values regardless of noise include unit number, street number, and any columns pertaining to spouse or dependents in the 1040 tax dataset, for example. In these cases where fields are blank even without noise, missing values are also represented by `numpy.nan`.

This noise type is called `leave_blank` in the configuration. It takes one parameter:

Table 15: Parameters to the leave_blank noise type

Parameter	Description	Default
`cell_probabi`	The probability that a cell in the column being configured is blank.	0.01 (1%)

Choose the wrong option

If a question on a survey or administrative form provides a list of options, respondents may sometimes choose the wrong option, either intentionally or by mistake. pseudopeople simulates this type of noise by sometimes selecting an incorrect option for columns that would have a list of options. All wrong options are equally likely. The possible values to select from depend on the column: the *Datasets page* lists them for each applicable column in pseudopeople's datasets.

This noise type is called `choose_wrong_option` in the configuration. It takes one parameter:

Table 16: Parameters to the choose_wrong_option noise type

Parameter	Description	Default
`cell_probabi`	The probability that, for a cell in the column being configured, the wrong option is chosen.	0.01 (1%)

Copy from household member

When responding to a survey or filling out a form, someone might accidentally or purposely answer a question about one household member with information about a different household member. To capture this type of error, pseudopeople can fill in certain fields about a simulant with values from a different member of the simulant's household, chosen at random. This type of noise can be applied to ages, dates of birth, and social security numbers.

Note that simulants who live in group quarters or who live alone are not eligible for this type of noise, so for each dataset, there is some maximum fraction of rows to which "copy from household member" noise can be applied. If the user requests a cell probability that is larger than what's possible, pseudopeople will add noise to the maximum possible number of rows.

This noise type is called `copy_from_household_member` in the configuration. It takes one parameter:

Table 17: Parameters to the copy_from_household_member noise type

Parameter	Description	Default
`cell_probabi`	The probability that, for a cell in the column being configured, the cell's value is replaced by the corresponding value from a household member.	0.01 (1%)

Note: The default value of 0.01 applies to most datasets. However, the default value is 0.0 for the SSN column in the W2 & 1099 dataset since SSNs are already subject to "borrow a social security number" noise in that dataset, and is also 0.0 for the SSN column in the SSA dataset because that column has no noise by default.

Use a nickname

Many people, when filling out forms or survey answers, choose to use nicknames instead of their legal names. A common example is an Alexander who chooses to go by Alex.

The "Use a nickname" noise type in pseudopeople simulates these kinds of responses for first and middle names. In order to do this, we used a list of 1,080 names and their relevant nicknames, from a project by Old Dominion University's Web Science and Digital Libraries Research Group. You can read more about the list of nicknames in the group's GitHub repository.

Instead of the person's legal name, pseudopeople selects the subset of simulated individuals who are eligible for a nickname (i.e., those whose legal first or middle name is included in the nicknames list detailed above), then replaces each selected simulant's first name with any of the nicknames included in the csv file.

This noise type is called `use_nickname` in the configuration. It takes one parameter:

Table 18: Parameters to the use_nickname noise type

Parameter	Description	Default
`cell_probabi`	The probability that, for a cell in the `first_name` column, a nickname is recorded.	0.01 (1%)

Use a fake name

Sometimes when people respond to a survey or fill out a form, they don't want to share their personal information. If the survey or form (whether online, on paper, or in person) requires a response, they might just make something up.

The "Use a fake name" noise type in pseudopeople simulates these kinds of responses for first and last names. Instead of the person's real name, pseudopeople records a randomly selected value from the "List of First Names Considered Fake or Incomplete" (for first names) or the "List of Last Names Considered Fake or Incomplete" (for last names) found in the NORC assessment of the Census Bureau's Person Identification Validation System.

This noise type is called `use_fake_name` in the configuration. It takes one parameter:

Table 19: Parameters to the use_fake_name noise type

Parameter	Description	Default
`cell_probabi`	The probability that, for a cell in the column (either first or last name), a fake name is recorded.	0.01 (1%)

Swap month and day

Swap month and day is a noise type that only applies to dates. It occurs when someone swaps the month and day to be in the incorrect position (e.g., December 8, 2022 would be listed in MM/DD/YYYY format as 08/12/2022).

This noise type is called `swap_month_and_day` in the configuration. It takes one parameter:

Table 20: Parameters to the swap_month_and_day noise type

Parameter	Description	Default
`cell_probabi`	The probability of a cell date having its month and day swapped.	0.01 (1%)

Misreport age

When someone reports their age in years, or especially when someone reports the age of someone else such as a family member, they may not get the value exactly right. For this type of simulated noise, the reported age is off by some amount, for example a year or two older or younger than the person actually is.

This noise type is called `misreport_age` in the configuration. It takes two parameters:

Table 21: Parameters to the misreport_age noise type

Parameter	Description	Default
`cell_prob`	The probability of each age value being misreported.	0.01 (1%)
`possible_`	One of two options: • A list of possible differences to add to the true age to get the misreported age. A negative number means that the reported age is too young, while a positive number means it is too old. Each difference is equally likely. • A dictionary, where the keys are the possible differences and the values are the probabilities of those differences. This is like the list option, except that it allows some age differences to be more likely than others. The probabilities must add up to 1. Zero (no change) is not allowed as a possible difference.	{-2: 0.1, -1: 0.4, +1: 0.4, +2: 0.1}

We assume that age would never be incorrectly reported as a negative number. In rare cases where applying the configured difference value would result in a negative age, we reflect this age back to positive (e.g. -2 becomes 2). This means there is still a spread of errors (they don't "bunch up" at zero). If this reflection would cause the age to be correct, we instead make the reported age one year younger than the true age.

Write the wrong digits

Sometimes people may write the wrong number for numeric data such as a street number, date, or social security number. This could be intentional or accidental. pseudopeople simulates this type of noise in fields that include numbers by randomly replacing some digits with different digits selected uniformly at random.

This noise type is called `write_wrong_digits` in the configuration. It takes two parameters:

Table 22: Parameters to the write_wrong_digits noise type

Parameter	Description	Default
`cell_probabi`	The probability that any given cell in the column will be selected to be eligible for this type of noise.	0.01 (1%)
`token_probab`	The conditional probability, given that a numeric cell has been selected for noise eligibility, that any given digit in the true number will be replaced by a different digit.	0.1 (10%)

Write the wrong ZIP code digits

When reporting a ZIP code on a survey or form, people may misremember or misreport the digits. They are probably more likely to do this for the last few digits (which identify the small, specific area) than the first few (which will be the same over a larger area). The "Write the wrong ZIP code digits" noise type is just like "Write the wrong digits" except that it can capture this difference between digits in different positions. The ZIP code column uses this noise type instead of "Write the wrong digits" for this reason.

This noise type is called `write_wrong_zipcode_digits` in the configuration. It takes two parameters:

Table 23: Parameters to the write_wrong_zipcode_digits noise type

Parameter	Description	Default
`cell_prob`	The probability of a cell being *considered* to have this noise type. One way to think about this is the probability that a ZIP code is reported by someone who isn't sure of their ZIP code. Whether or not there are actually any errors depends on the next parameter.	0.01 (1%)
`digit_pro`	A list of five probabilities, one for each digit in a (5-digit) ZIP code. The first value in this list is the probability that the first digit of the ZIP code will be wrong **given that the cell is being considered for this noise type**. The second value in the list is the corresponding probability for the second digit, and so on.	[0.04, 0.04, 0.20, 0.36, 0.36]

Make phonetic errors

A phonetic error occurs when a character is misheard. For instance, this could happen with similar sounding letters when spoken (like 't' and 'd') or letters that make the same sounds within a word (like 'o' and 'ou').

pseudopeople defines the possible phonetic substitutions using this file, which was produced by the GeCO project.

This noise type is called `make_phonetic_errors` in the configuration. It takes two parameters:

Table 24: Parameters to the make_phonetic_errors noise type

Parameter	Description	Default
`cell_probabi`	The probability of a cell being *considered* to have this noise type. One way to think about this is the probability that a string is transcribed by an error-prone program or human transcriber. Whether or not there are actually any errors depends on the next parameter.	0.01 (1%)
`token_probat`	The probability of each corruption-eligible token being misheard **given that the cell is being considered for this noise type**. One way to think about this is the probability of a phonetic error on any given corruption-eligible token when the transcriber is error-prone.	0.1 (10%)

Make optical character recognition (OCR) errors

An optical character recognition (OCR) error is when a string is misread for another string that is visually similar. Some common examples are 'S' instead of '5' and 'm' instead of 'iii'.

pseudopeople defines the possible OCR substitutions using this CSV file, which was produced by the GeCO project. In the file, the first column is the real string (which we call a "token") and the second column is what it could be misread as (a "corruption"). The same token can be associated with multiple corruptions.

To implement this, we first select the rows to noise, as in other noise types. For those rows, each corruption-eligible token in the relevant string is selected to be corrupted or not, according to the token noise probability. Each token selected for corruption is replaced with its corruption according to the above CSV file (choosing uniformly at random in the case of multiple corruption options for a single token), **unless a token with any overlapping characters (in the original string) has already been corrupted**.

Note: Tokens are corrupted in the order of the location of their first character in the original string, from beginning to end, breaking ties (e.g. 'l' and 'l>' are both corruption-eligible tokens and may start on the same 'l') by corrupting longer tokens first. Note that in an example `abcd` where `ab`, `bc`, **and** `cd` have **all** been selected to be corrupted, the corruption of `ab` prevents the corruption of `bc` from occurring, which then allows `cd` to be corrupted even though it overlapped with `bc`.

This noise type is called `make_ocr_errors` in the configuration. It takes two parameters:

Table 25: Parameters to the make_ocr_errors noise type

Parameter	Description	Default
`cell_probabi`	The probability of a cell being *considered* to have this noise type. One way to think about this is the probability that a string is read by an inaccurate OCR program or human reader. Whether or not there are actually any errors depends on the next parameter.	0.01 (1%)
`token_probat`	The probability of each corruption-eligible token being misread **given that the cell is being considered for this noise type**. One way to think about this is the probability of an OCR error on any given corruption-eligible token when a string is being read inaccurately.	0.1 (10%)

Make typos

Typos occur in survey and administrative datasets when someone – a survey respondent, a canvasser, or someone entering their own information on a form – types a value incorrectly.

Currently, pseudopeople implements two kinds of typos: inserting extra characters directly preceding characters that are adjacent on a keyboard, or replacing a character with one that is adjacent. When pseudopeople introduces typos, 10% of them are inserted characters, while the other 90% are replaced characters. This is currently not configurable. In either kind of typo, all adjacent characters are equally likely to be chosen.

To define "adjacent", we use a grid version of a QWERTY keyboard layout (left-justified, which is not exactly accurate to most keyboards' half-key-offset layout) and accompanying number pad. This approach is inspired by the GeCO project, with some changes to include capital letters and have a complete numberpad. Two characters are considered adjacent if they are touching, either on a side or diagonally:

```
qwertyuiop
asdfghjkl
```

(continues on next page)

(continued from previous page)

```
zxcvbnm

QWERTYUIOP
ASDFGHJKL
ZXCVBNM

789
456
123
0
```

Note that there are empty lines above, which separate the parts. Therefore, a number is never replaced by a letter (or vice versa), and a capital letter is never replaced by a lowercase letter (or vice versa). There are currently no typos involving special characters.

This noise type is called `make_typos` in the configuration. It takes two parameters:

Table 26: Parameters to the make_typos noise type

Parameter	Description	Default
`cell_probabi`	The probability of a cell being *considered* to have this noise type. One way to think about this is the probability that a value is typed carelessly. Whether or not there are actually any errors depends on the next parameter.	0.01 (1%)
`token_probak`	The probability of each character (which we call a "token") having a typo **given that the cell is being considered for this noise type**. One way to think about this is the probability of a typo on any given character when the value is being typed carelessly.	0.1 (10%)

2.5 Configuration

You can customize the noise in the datsets the pseudopeople package generates. This allows you to explore different scenarios and see how sensitive entity resolution methods are to the types and levels of noise present in their input data.

2.5.1 Overriding defaults

Noise is configurable at a very fine-grained level. It can be customized separately for each **dataset** and **noise type**. *Column-based noise types* can additionally have different settings for each **column**.

Due to this fine-grained control, there are a very large number of settings. **It is not necessary to configure everything.** pseudopeople includes reasonable default noise settings and your configuration can override as few or as many of the default values as you like. You can also pass the special value `pseudopeople.NO_NOISE`, which prevents all configurable noise types from occurring at all.

To learn more about the default settings, see *Noise Type Details*. You can access the defaults from your Python code by calling the *`pseudopeople.get_config()`* function.

2.5.2 Configuration structure

Configuration can be supplied as a nested Python dictionary, or as a YAML file. In either case, the structure is the same:

- The top-level keys are the datasets.
- Within each of these are keys for the *categories of noise*: row-based and column-based.
- **For column-based noise-only**, the next layer of keys is for the columns in the dataset.
- Nested within these are keys for the individual *noise types*.
- Finally, each noise type has parameters.

As an example, say we wanted to change the cell probability parameter (which is the probability of a cell being wrong) of the *Choose the wrong option* noise type, for the sex column of the Decennial Census dataset. Here are the configurations to do this in Python and YAML, respectively:

```
config = {
    'decennial_census': { # Dataset
        'column_noise': { # "Choose the wrong option" is in the column-based noise␣
↪category
            'sex': { # Column
                'choose_wrong_option': { # Noise type
                    'cell_probability': 0.05, # Parameter (and value)
                },
            },
        },
    },
}
```

```
decennial_census: # Dataset
    column_noise: # "Choose the wrong option" is in the column-based noise category
        sex: # Column
            choose_wrong_option: # Noise type
                cell_probability: 0.05 # Parameter (and value)
```

Row-based noise is similar, except that there is no key to specify the column, since it is not column-specific. For example to change the probability of *nonresponse* in the Decennial Census, the configuration would be:

```
config = {
    'decennial_census': { # Dataset
        'row_noise': { # "Omit a row" is in the row-based noise category
            'do_not_respond': { # Noise type
                'row_probability': 0.05, # Parameter (and value)
            },
        },
    },
}
```

```
decennial_census: # Dataset
    row_noise: # "Omit a row" is in the row-based noise category
        do_not_respond: # Noise type
            row_probability: 0.05 # Parameter (and value)
```

2.5.3 How to pass configuration to pseudopeople

Each of pseudopeople's *dataset generation functions* takes a `config` argument. This argument can be passed either a Python dictionary, the path to a YAML file, or the special value `pseudopeople.NO_NOISE`, which prevents all configurable noise types from occurring at all.

2.5.4 Configurable parameters

The noise types that can be configured, and the parameters of each, are listed in the *Noise Type Details* section.

2.6 API Reference

This section of the docs describes the details of how to use each function included in pseudopeople. These functions make up pseudopeople's public Application Programming Interface (API). If you are a new user, you may want to start with the *Quickstart* guide instead.

2.6.1 Dataset Generation Functions

Each of the following functions generates one of the simulated datasets documented on the *Datasets page*. For example, *`pseudopeople.generate_decennial_census()`* generates the Decennial Census dataset.

All of the dataset generation functions have the same (optional) parameters. Notable parameters include:

- a *source* path to the root directory of *pseudopeople simulated population data* (defaults to using the sample population included with pseudopeople).
- a *config* path to a YAML file, a Python dictionary, or the special value `pseudopeople.NO_NOISE`, to *override the default configuration*.
- a *year* (defaults to 2020).

For applied examples of using these functions, see the *Quickstart* and *tutorials*.

`pseudopeople.`**`generate_american_community_survey`**(*source=None*, *seed=0*, *config=None*, *year=2020*, *state=None*, *verbose=False*)

Generates a pseudopeople ACS dataset which represents simulated responses to the ACS survey.

The American Community Survey (ACS) is an ongoing household survey conducted by the US Census Bureau that gathers information on a rolling basis about American community populations. Information collected includes ancestry, citizenship, education, income, language proficiency, migration, employment, disability, and housing characteristics.

Parameters

- **`source`** (*`Path | str`*) – The root directory containing pseudopeople simulated population data. Defaults to using the included sample population when source is *None*.
- **`seed`** (*`int`*) – An integer seed for randomness. Defaults to 0.
- **`config`** (*`Path | str | Dict[str, Dict]`*) – An optional override to the default configuration. Can be a path to a configuration YAML file, a configuration dictionary, or the sentinel value *pseudopeople.NO_NOISE*, which will generate a dataset without any configurable noise.

- **year** (*int | None*) – The year for which to generate simulated American Community Surveys of the simulated population (format YYYY, e.g., 2036); the simulated dataset will contain records for surveys conducted on any date in the specified year. Default is 2020. If *None* is passed instead, data for all available years are included in the returned dataset.
- **state** (*str | None*) – The US state for which to generate simulated American Community Surveys of the simulated population, or *None* (default) to generate data for all available US states. The returned dataset will contain survey data for simulants living in the specified state during the specified year. Can be a full state name or a state abbreviation (e.g., "Ohio" or "OH").
- **verbose** (*bool*) – Log with verbosity if *True*. Default is *False*.

Returns
A *pandas.DataFrame* of simulated ACS data.

Raises

- **ConfigurationError** – An invalid *config* is provided.
- **DataSourceError** – An invalid pseudopeople simulated population data source is provided.
- **ValueError** – The simulated population has no data for this dataset in the specified year or state.

Return type
DataFrame

pseudopeople.**generate_current_population_survey**(*source=None*, *seed=0*, *config=None*, *year=2020*, *state=None*, *verbose=False*)

Generates a pseudopeople CPS dataset which represents simulated responses to the CPS survey.

The Current Population Survey (CPS) is a household survey conducted by the US Census Bureau and the US Bureau of Labor Statistics. This survey is administered by Census Bureau field representatives across the country through both personal and telephone interviews. CPS collects labor force data, such as annual work activity and income, veteran status, school enrollment, contingent employment, worker displacement, job tenure, and more.

Parameters

- **source** (*Path | str*) – The root directory containing pseudopeople simulated population data. Defaults to using the included sample population when source is *None*.
- **seed** (*int*) – An integer seed for randomness. Defaults to 0.
- **config** (*Path | str | Dict[str, Dict]*) – An optional override to the default configuration. Can be a path to a configuration YAML file, a configuration dictionary, or the sentinel value *pseudopeople.NO_NOISE*, which will generate a dataset without any configurable noise.
- **year** (*int | None*) – The year for which to generate simulated Current Population Surveys of the simulated population (format YYYY, e.g., 2036); the simulated dataset will contain records for surveys conducted on any date in the specified year. Default is 2020. If *None* is passed instead, data for all available years are included in the returned dataset.
- **state** (*str | None*) – The US state for which to generate simulated Current Population Surveys of the simulated population, or *None* (default) to generate data for all available US states. The returned dataset will contain survey data for simulants living in the specified state during the specified year. Can be a full state name or a state abbreviation (e.g., "Ohio" or "OH").
- **verbose** (*bool*) – Log with verbosity if *True*. Default is *False*.

Returns
A *pandas.DataFrame* of simulated CPS data.

Raises

- **ConfigurationError** – An invalid *config* is provided.
- **DataSourceError** – An invalid pseudopeople simulated population data source is provided.
- **ValueError** – The simulated population has no data for this dataset in the specified year or state.

Return type
DataFrame

pseudopeople.**generate_decennial_census**(*source=None*, *seed=0*, *config=None*, *year=2020*, *state=None*, *verbose=False*)

Generates a pseudopeople decennial census dataset which represents simulated responses to the US Census Bureau's Census of Population and Housing.

Parameters

- **source** (*Path | str*) – The root directory containing pseudopeople simulated population data. Defaults to using the included sample population when source is *None*.
- **seed** (*int*) – An integer seed for randomness. Defaults to 0.
- **config** (*Path | str | Dict[str, Dict]*) – An optional override to the default configuration. Can be a path to a configuration YAML file, a configuration dictionary, or the sentinel value *pseudopeople.NO_NOISE*, which will generate a dataset without any configurable noise.
- **year** (*int | None*) – The year for which to generate a simulated decennial census of the simulated population (format YYYY, e.g., 2030). Must be a decennial year (e.g., 2020, 2030, 2040). Default is 2020. If *None* is passed instead, data for all available years are included in the returned dataset.
- **state** (*str | None*) – The US state for which to generate a simulated census of the simulated population, or *None* (default) to generate data for all available US states. The returned dataset will contain data for simulants living in the specified state on Census Day (April 1) of the specified year. Can be a full state name or a state abbreviation (e.g., "Ohio" or "OH").
- **verbose** (*bool*) – Log with verbosity if *True*. Default is *False*.

Returns
A *pandas.DataFrame* of simulated decennial census data.

Raises

- **ConfigurationError** – An invalid *config* is provided.
- **DataSourceError** – An invalid pseudopeople simulated population data source is provided.
- **ValueError** – The simulated population has no data for this dataset in the specified year or state.

Return type
DataFrame

pseudopeople.**generate_social_security**(*source=None*, *seed=0*, *config=None*, *year=2020*, *verbose=False*)

Generates a pseudopeople SSA dataset which represents simulated Social Security Administration (SSA) data.

Parameters

- **source** (*Path | str*) – The root directory containing pseudopeople simulated population data. Defaults to using the included sample population when source is *None*.
- **seed** (*int*) – An integer seed for randomness. Defaults to 0.
- **config** (*Path | str | Dict[str, Dict]*) – An optional override to the default configuration. Can be a path to a configuration YAML file, a configuration dictionary, or the sentinel value *pseudopeople.NO_NOISE*, which will generate a dataset without any configurable noise.
- **year** (*int | None*) – The final year of simulated social security records to include in the dataset (format YYYY, e.g., 2036); will also include records from all previous years. Default is 2020. If *None* is passed instead, data for all available years are included in the returned dataset.
- **verbose** (*bool*) – Log with verbosity if *True*. Default is *False*.

Returns
A *pandas.DataFrame* of simulated SSA data.

Raises

- **ConfigurationError** – An invalid *config* is provided.
- **DataSourceError** – An invalid pseudopeople simulated population data source is provided.
- **ValueError** – The simulated population has no data for this dataset in the specified year or any prior years.

Return type
DataFrame

pseudopeople.**generate_taxes_1040**(*source=None*, *seed=0*, *config=None*, *year=2020*, *state=None*, *verbose=False*)

Generates a pseudopeople 1040 tax dataset which represents simulated tax form data.

Parameters

- **source** (*Path | str*) – The root directory containing pseudopeople simulated population data. Defaults to using the included sample population when source is *None*.
- **seed** (*int*) – An integer seed for randomness. Defaults to 0.
- **config** (*Path | str | Dict[str, Dict]*) – An optional override to the default configuration. Can be a path to a configuration YAML file, a configuration dictionary, or the sentinel value *pseudopeople.NO_NOISE*, which will generate a dataset without any configurable noise.
- **year** (*int | None*) – The tax year for which to generate records (format YYYY, e.g., 2036); the simulated dataset will contain the 1040 tax forms filed by simulants for the specified year. Default is 2020. If *None* is passed instead, data for all available years are included in the returned dataset.
- **state** (*str | None*) – The US state for which to generate tax records from the simulated population, or *None* (default) to generate data for all available US states. The returned dataset will contain 1040 tax forms filed by simulants living in the specified state during the specified tax year. Can be a full state name or a state abbreviation (e.g., "Ohio" or "OH").
- **verbose** (*bool*) – Log with verbosity if *True*. Default is *False*.

Returns
A *pandas.DataFrame* of simulated 1040 tax data.

Raises

- **ConfigurationError** – An invalid *config* is provided.
- **DataSourceError** – An invalid pseudopeople simulated population data source is provided.
- **ValueError** – The simulated population has no data for this dataset in the specified year or state.

Return type
DataFrame

pseudopeople.**generate_taxes_w2_and_1099**(*source=None*, *seed=0*, *config=None*, *year=2020*, *state=None*, *verbose=False*)

Generates a pseudopeople W2 and 1099 tax dataset which represents simulated tax form data.

Parameters

- **source** (*Path | str*) – The root directory containing pseudopeople simulated population data. Defaults to using the included sample population when source is *None*.
- **seed** (*int*) – An integer seed for randomness. Defaults to 0.
- **config** (*Path | str | Dict[str, Dict]*) – An optional override to the default configuration. Can be a path to a configuration YAML file, a configuration dictionary, or the sentinel value *pseudopeople.NO_NOISE*, which will generate a dataset without any configurable noise.
- **year** (*int | None*) – The tax year for which to generate records (format YYYY, e.g., 2036); the simulated dataset will contain the W2 & 1099 tax forms filed by simulated employers for the specified year. Default is 2020. If *None* is passed instead, data for all available years are included in the returned dataset.
- **state** (*str | None*) – The US state for which to generate tax records from the simulated population, or *None* (default) to generate data for all available US states. The returned dataset will contain W2 & 1099 tax forms filed for simulants living in the specified state during the specified tax year. Can be a full state name or a state abbreviation (e.g., "Ohio" or "OH").
- **verbose** (*bool*) – Log with verbosity if *True*. Default is *False*.

Returns
A *pandas.DataFrame* of simulated W2 and 1099 tax data.

Raises

- **ConfigurationError** – An invalid *config* is provided.
- **DataSourceError** – An invalid pseudopeople simulated population data source is provided.
- **ValueError** – The simulated population has no data for this dataset in the specified year or state.

Return type
DataFrame

pseudopeople.**generate_women_infants_and_children**(*source=None*, *seed=0*, *config=None*, *year=2020*, *state=None*, *verbose=False*)

Generates a pseudopeople WIC dataset which represents a simulated version of the administrative data that would be recorded by WIC. This is a yearly file of information about all simulants enrolled in the program as of the end of that year.

The Special Supplemental Nutrition Program for Women, Infants, and Children (WIC) is a government benefits program designed to support mothers and young children. The main qualifications are income and the presence of young children in the home.

Parameters

- **source** (*Path | str*) – The root directory containing pseudopeople simulated population data. Defaults to using the included sample population when source is *None*.
- **seed** (*int*) – An integer seed for randomness. Defaults to 0.
- **config** (*Path | str | Dict[str, Dict]*) – An optional override to the default configuration. Can be a path to a configuration YAML file, a configuration dictionary, or the sentinel value *pseudopeople.NO_NOISE*, which will generate a dataset without any configurable noise.
- **year** (*int | None*) – The year for which to generate WIC administrative records (format YYYY, e.g., 2036); the simulated dataset will contain records for simulants enrolled in WIC at the end of the specified year (or on May 1, 2041 if *year=2041* since that is the end date of the simulation). Default is 2020. If *None* is passed instead, data for all available years are included in the returned dataset.
- **state** (*str | None*) – The US state for which to generate WIC administrative records from the simulated population, or *None* (default) to generate data for all available US states. The returned dataset will contain records for enrolled simulants living in the specified state at the end of the specified year (or on May 1, 2041 if *year=2041* since that is the end date of the simulation). Can be a full state name or a state abbreviation (e.g., "Ohio" or "OH").
- **verbose** (*bool*) – Log with verbosity if *True*. Default is *False*.

Returns

A *pandas.DataFrame* of simulated WIC data.

Raises

- **ConfigurationError** – An invalid *config* is provided.
- **DataSourceError** – An invalid pseudopeople simulated population data source is provided.
- **ValueError** – The simulated population has no data for this dataset in the specified year or state.

Return type

DataFrame

2.6.2 Working with the Configuration

The pseudopeople *configuration* contains a very large number of settings. It may be easier in some situations to interact with the configuration programmatically in Python, rather than by creating a configuration dictionary or YAML file by hand. For example, if you wanted to double the cell probability parameter of every noise type in every column, or set all noise types to zero except one to isolate the effect of a specific type of noise, it would be easier to access the defaults as a data structure in Python and modify them that way.

We currently have one function that facilitates this kind of use of the configuration, shown below. We may add more functionality in this area in a future release of pseudopeople.

pseudopeople.**get_config**(*overrides=None*)

Function that returns the pseudopeople configuration containing all default values. To get the default probability of nonresponse in the Decennial Census dataset:

```
>>> import pseudopeople as psp
>>> psp.get_config()['decennial_census']['row_noise']['do_not_respond']
{'row_probability': 0.0145}
```

To view that same part of the configuration after applying a user override:

```
>>> overrides = {'decennial_census': {'row_noise': {'do_not_respond': {'row_
↪probability': 0.1}}}}
>>> psp.get_config(overrides)['decenial_census']['row_noise']['do_not_respond']
{'row_probability': 0.1}
```

Parameters

overrides (*Path | str | Dict*) – An optional set of overrides to the default configuration. Can be a (nested) Python dictionary mapping noise type parameters to the desired override values, a path to a YAML file with the same nested structure (see the *configuration structure* section of the documentation for details), or the special sentinel value *pseudopeople.NO_NOISE*, which will return a configuration in which all configurable noise is set to zero. When passing a dictionary or YAML file, it is not necessary to provide a complete configuration; any configuration parameters not specified in *overrides* will be filled in with the default values.

Returns

A complete configuration dictionary.

Raises

ConfigurationError – An invalid configuration is passed with *overrides*.

Return type

Dict

2.7 Glossary

Configuration

Settings that allow you to customize the noise present in the datasets generated by pseudopeople. Noise is configurable at a very fine-grained level, with settings specific to the dataset, noise type, and column (where applicable).

Datasets

The types of data that can be simulated with pseudopeople, each of which is the simulated analog of a "real world" database from a census, survey, or administrative source. For example, pseudopeople's American Community Survey (ACS) dataset is analogous to the data that would be collected by that survey in real life.

Entity resolution (ER)

The task of identifying the unique entities associated with a set of records, where multiple records may refer to the same entity. Also called "record linkage," among other names.

Noise

Errors introduced to a pseudopeople dataset. These simulate data errors that would be found in real-life survey and administrative data.

Noise types

The types of error that can be introduced to a pseudopeople dataset. Each one simulates a specific type of mistake or inaccuracy that could occur in a real-life data collection or generation process. For example, one of the noise types in pseudopeople is a simulant choosing the wrong option from a list of choices on a form.

Probabilistic record linkage (PRL)

Entity resolution ("record linkage") methods that internally use probabilities of some kind to represent uncertainty about which records belong to which entities.

Record linkage

Another term for entity resolution.

Simulant
A simulated person represented in a pseudopeople-generated dataset.

BIBLIOGRAPHY

[Census_PES] Bureau, US Census. March 10, 2022. “Detailed Coverage Estimates for the 2020 Census Released Today.” Census.Gov. Accessed September 29, 2022. https://www.census.gov/library/stories/2022/03/who-was-undercounted-overcounted-in-2020-census.html.

[Response_Rates_BLS] “Household and Establishment Survey Response Rates: U.S. Bureau of Labor Statistics.” n.d. Accessed October 11, 2022. https://www.bls.gov/osmr/response-rates/home.htm.

PYTHON MODULE INDEX

INDEX