Supplement to "Comparing the Linkage Performance of fastLink, Splink, and Match*Pro at the Florida Cancer Registry Using Simulated pseudopeople Data"

Anders Alexandersson

06/30/2024

This supplement was not included in the main text (Alexandersson 2024) because of space and pedagogical constraints. The supplement consists of four parts. The first part describes how to create test data using the Python package `pseudopeople`. The remaining three parts describe the `fastLink`, `Splink`, and `Match*Pro` results on the test data. There are 670,214 linkable records, with 660,227 actual matches (true positives, TP) and 9,987 actual non-matches (true negatives, TN).

S1 Creating Test Data Using the Python Package `pseudopeople`

pseudopeople is a Python package that generates realistic simulated data about a fictional United States population, designed for use in testing record linkage (entity resolution) methods. `pseudopeople` is currently in a public beta release. There are currently three collections of `pseudopeople` input data:

- Sample data (a fictional population of ~10,000 simulants living in Anytown, US, 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)

The University of Washington (UW) IHME Simulation Science Team led by Abraham Flaxman is developing `pseudopeople`. The UW funding for the project comes from the Census Bureau as a Cooperative Agreement. The Florida Cancer Data System (FCDS) first received the Rhode Island input data, version 1.00, as a 23.9 GB zip file on July 7, 2023. See the data access request. The FCDS received version 2.0.0 on September 11, 2023 and the current version 2.0.1 on November 15, 2023 as a 61.3 GB zip file. Two large improvements in version 2.x are being compatible with `pandas` 2.x, and having `date_of_birth` in the correct dataset (it moved from `taxes_w2_and_1099` to `decennial_census`). The current version of `pseudopeople` is 1.0, which has duplicates. We used the previous version 0.8.3.

Python Installation

`pseudopeople` 0.8.3 requires Python 3.8-3.11 to be installed. Most users of the `pseudopeople` Rhode Island data probably need to install Python if Python is not already installed. We installed Python 3.11.0 on Windows. We used the RStudio instructions for installing Python on a Windows desktop. Specifically, we disabled App execution aliases, and we used pyenv-win. `pyenv-win` is a simple Python version management tool for Windows. The main alternative to `pyenv-win` is the standard Python installer from www.python.org/downloads/, which includes the Python launcher for Windows but not version management.

Overall, it is debatable whether `pyenv-win` or the standard installer is better for installing Python on Windows. Users who want to create test data using `pseudopeople` are likely more advanced than users who merely want to use such created test data. For these advanced users, Python installation typically should not be an issue. Software for automated reporting such as Jupyter or Quarto is optional. We used `Quarto` 1.3.433. "Add a guide on how to setup Python on Windows to work well with Quarto" is Quarto issue 4737 to be resolved in Quarto 1.5.

Tip to possibly avoid the Python Installation

You do not need to install Python to use the `pseudopeople` sample data; you can use Google Colab for that. However, it is tedious to upload large datasets to Google Colab. The easiest solution for uploading large datasets to Google Colab seems to be to first sign up for a Google One plan, for example Google One Basic for $1.99 per month which provides 100 GB of storage. Then, you can access the large datasets in Google Colab by mounting Google Drive locally.

Reproducibility

Python environments are infamously difficult to manage. For a funny illustration, see XKCD comic figure 1987. The reason is that a Python virtual environment is a directory on top of an existing Python installation, which leaves open implementation issues. There are several virtual environment tools in Python such as: venv, virtualenv, pipenv, poetry, flit, PDM, and hatch. In the long term, a Python Enhancement Proposal (PEP) could resolve the issue. A Draft PEP for a Python Packaging Authority (PyPA) is being discussed.

We used `venv` for the Python virtual environment tool, because `venv` requires no installation. `venv` is a *manual* virtual environment tool, and therefore it is best used for smaller projects. Quarto suggests the directory name `env`. However, the `venv` directory is commonly named `.venv` (notice the prepended dot) to indicate that it is a special directory for holding a virtual environment. Here, reproducibility is viewed as a five-step process:

1. Install `venv` using the command python -m venv .venv
2. Activate `venv` on Windows using the command . .venv/Scripts/activate
3. Install packages using pip.
4. Save the environment using pip freeze > requirements.txt. To update the environment, use `pip install -r requirements.txt --upgrade`.
5. Render the report using Quarto

Tip to create a `requirements.txt` file

A `requirements.txt` file generated with `pip freeze` will include both used and unused libraries, which is inefficient. A seemingly better way to create a `requirements.txt` file is with the package pipreqs. However, to review this was outside the scope of this supplement considering that `pipreqs` (v0.4.13) is much less seldom updated than `pip` (v23.3.1).

Creating the `pseudopeople` Rhode Island dataset

Imports should usually be on separate lines, according to PEP 8. Certain modules are conventionally imported with abbreviated names, for example `pseudopeple as psp`. The primary `pandas` (or `pd`) data structure is a DataFrame, usually abbreviated `df`. The `DataFrame` is a dataset with index (rows, starts at 0 by default) and columns. The first code below imports the basic required packages, and then it checks the version of `pseudopeople`.

```
import os                                                        ①
import warnings
import random

import numpy as np                                               ②
import pandas as pd
import pseudopeople as psp
# !date

# --- !pip install pseudopeople
```

```
warnings.filterwarnings('ignore')                                         ③
psp.__version__                                                           ④
```

① Import the Python Standard Library
② Import modules
③ Ignore warnings (otherwise displayed in PDF)
④ The package `pseudopeople` uses current version 0.8.3 from January 9, 2024.

```
'0.8.3'
```

Find the path of the input data, and get the current working directory:

```
# os.chdir('../')
# os.listdir('V:/Testing/Monograph_2024/pseudopeople/pseudopeople_simulated_population_ri_2_0
ospath = os.path.join("V:", os.sep, "Testing", "Monograph_2024",          ①
    "pseudopeople", "pseudopeople_simulated_population_ri_2_0_1",
    "pseudopeople_simulated_population_rhode_island_2_0_0")
os.getcwd()                                                               ②
```

① The path of the input data
② Get the current working directory

```
'V:\\Testing\\Monograph_2024\\pseudopeople'
```

The `pseudopeople` package can generate seven input datasets: US Decennial Census, American Community Survey (ACS), Current Population Survey (CPS), Women, Infants, and Children (WIC), Social Security Administration (SSA), Tax Forms: W-2 & 1099, and Tax Form: 1040.

The FCDS 2023 monograph used datasets which were generated before `pseudopeople` became public beta. In the developers' hyper-focus on Census Bureau style data, they no longer have a single dataset with the variables that the FCDS needs. Since those fields all appear *somewhere*, the simplest way to proceed is to merge the variables we need from two simulated datasets.

Below, we created the two datasets Tax Forms: W-2 & 1099 (`taxes_w2_and_1099`) and US Decennial Census (`decennial_census`), and then merged the variables we need.

By design, all `pseudopeople` variables have 1% "missingness", and it is defined as "Config key" followed by "parameter" and "value", for example `{'leave_blank': {'cell_probability': 0.01}}`. The documentation currently is confusing because some variables such as 'unit_number' defaults to *additional* missingness as opposed to missingness. The documentation is especially confusing when making composite address variables. In the future, the documentation likely will include a tutorial about making composite address variables.

Below, we changed to 20% missingness for `ssn` (see note 1), read the datasets `taxes_w2_and_1099` (note 2) and `decennial_census` (note 3), and we created a new variable `address` (note 4) as the concatenation of variables 'street_number' and 'street_name' and with 5% missingness (note 5). The Project US@ Technical Specification (on pages 13-14) states that address instead SHOULD be parsed into separate fields (variables). See the `pseudopeople` Configuration documentation for `pseudopeople` configuration details.

```python
# sample dataset:
# df1a = psp.generate_taxes_w2_and_1099()

# Rhode Island dataset:
# print ("Current working dir : %s" % os.getcwd())
# df1a = psp.generate_taxes_w2_and_1099(source = os.getcwd())

config = {
    'taxes_w2_and_1099': {
        'column_noise': {
            'ssn': {'leave_blank': {'cell_probability': 0.20}}              ①
        }
    },
    'decennial_census': {
        'column_noise': {
            'street_number': {'leave_blank': {'cell_probability': 0.0}},
            'street_name': {'leave_blank': {'cell_probability': 0.0}},
            'unit_number': {'leave_blank': {'cell_probability': 0.0}}
        }
    }
}
df1a = psp.generate_taxes_w2_and_1099(source = ospath,                      ②
    config=config)
print('Tax data contains', len(df1a.columns), 'columns (expect 24)')
df1a = df1a.filter(
  ['simulant_id', 'first_name', 'last_name', 'ssn'])

# there are multiple rows for simulants who had more than one employer
df1a = df1a.groupby('simulant_id').first().reset_index()
# df1a

# def my_concat_address_fields(s):
#  s = s.filter(['street_number', 'street_name', 'unit_number'])
#  return ' '.join(s.dropna())

# df1b = psp.generate_decennial_census()
```

```
df1b = psp.generate_decennial_census(source = ospath,                    ③
    config=config)
df1b['address'] = (df1b.street_number.fillna('')                          ④
    + ' '
    + df1b.street_name.fillna('')
)

percentage_blank = 5                                                      ⑤
num_blank = int(len(df1b) * percentage_blank / 100)
blank_indices = np.random.choice(df1b.index, num_blank, replace=False)
df1b.loc[blank_indices, 'address'] = np.nan

# df1b = df1b.replace(np.nan, '')   # new: replace nan with ''
# df1b['address'] = df1b.apply(my_concat_address_fields, axis=1)
# df1b = df1b.replace(r'^\s*$', np.nan, regex=True)  # new: put nan back
# object mm/dd/yyyy -> datetime yyyy-mm-dd -> string yyyymmdd

df1b = df1b.filter(
    ['simulant_id', 'sex', 'address', 'city', 'state', 'zipcode',
    'street_number', 'street_name', 'unit_number', 'date_of_birth'])
# df1b
df1 = pd.merge(df1a, df1b)
```

① Custom configuration of 20% missing `ssn` and 0% missing "`address`"
② Create the dataframe `taxes_w2_and_1099` (not saved) from Rhode Island data
③ Create the dataframe `decennial_census` (not saved) from Rhode Island data
④ Combine all address columns into a composite column called `address`
⑤ Make 5% of cells in the new `address` column blank

Note the configured 20% missingness noise in `ssn` and address. The function `get_config()` can be used to verify the configurations, for example as below:

```
psp.get_config(config)['taxes_w2_and_1099']['column_noise']['ssn']
```

```
{'leave_blank': {'cell_probability': 0.2},
 'copy_from_household_member': {'cell_probability': 0.0},
 'write_wrong_digits': {'cell_probability': 0.01, 'token_probability': 0.1},
 'make_ocr_errors': {'cell_probability': 0.01, 'token_probability': 0.1},
 'make_typos': {'cell_probability': 0.01, 'token_probability': 0.1}}
```

```
psp.get_config(config)['decennial_census']['column_noise']['street_name']
```

```
{'leave_blank': {'cell_probability': 0.0},
 'make_phonetic_errors': {'cell_probability': 0.01, 'token_probability': 0.1},
 'make_ocr_errors': {'cell_probability': 0.01, 'token_probability': 0.1},
 'make_typos': {'cell_probability': 0.01, 'token_probability': 0.1}}
```

We can now change variables as needed to make them similar to the required FCDS linkage variables. Whereas NAACCR data items are in mixed case, the FCDS linkage variables use uppercase. The variable `date_of_birth` has out-of-range errors in month that would not be expected in the FCDS database such as month "00" or "14". Therefore, `date_of_birth` is converted to cleaned variable `DOB`:

```
df1['ssn'] = df1['ssn'].str.replace("-", "")                                ①
df1['first_name'] = df1['first_name'].str.upper()                           ②
df1['last_name'] = df1['last_name'].str.upper()                             ③
df1['address'] = df1['address'].str.upper()                                 ④
df1['city'] = df1['city'].str.upper()                                       ⑤
df1['sex'] = np.where(df1['sex'] == "Male", 1, 2)                           ⑥

df1['DOB'] = pd.to_datetime(df1['date_of_birth'], errors='coerce')
df1['DOB'] = df1['DOB'].astype('string').str.replace('-', '')               ⑦
# print(df1['date_of_birth'])  # object         (in format mm/dd/yyyy)
# print(df1['DOB'])            # string         (in format yyyymmdd)
```

① SSN without dashes as in NACCR data item #2320: Social Security Number
② First name in uppercase for NACCR data item #2240: First Name
③ Last name in uppercase for NACCR data item #2232: Last Name
④ Address in uppercase for NACCR data item #2350: Street (Number & Name)
⑤ City in uppercase for NAACCR data item #1810: City
⑥ Sex coded as in NAACCR data item #220: Sex: 1=Male, 2=Female
⑦ DOB, converted from `date_of_birth`, as in NAACCR data item #240: City

For SSN, a common issue is transposition, that is, two adjacent digits being swapped. Because `pseudopeople` has no noise for it, we create a new function `swap_two()` and provide an example. It is easier to do this code change with SSN without dashes.:

```
def swap_two(x: str) -> str:
    # Convert the string to a list of characters
    char_list = list(x)

```

```python
    # Ensure that the string has at least two characters to swap
    if len(char_list) < 2:
        return x

    # Pick a random position within the valid range for swapping
    # (positions 0 to len(x) - 2)
    pos1 = random.choice(range(len(x) - 1))

    # Calculate pos2 while ensuring it stays within the valid range
    pos2 = min(pos1 + 1, len(x) - 1)

    # Make the swap
    char_list[pos1], char_list[pos2] = char_list[pos2], char_list[pos1]

    # Join the characters back into a string and return
    return ''.join(char_list)

result = swap_two('123457890')
print(result)
```

```
123547890
```

Before modifying `ssn`, we create a backup variable `ssn_before_swap`. We also set the seeds for reproducibility, 1% transposition noise for `ssn`, and we list the first and last rows with transposed `ssn`.

! Important – Reproducibility issue for `sex` and `address` to be fixed in version 1.0.0

`pseudopeople` 0.8.3 from January 9, 2024, has a randomness bug which is fixed in version 1.0.0 from February 12 (see pseudopeople/pull/383 for details). We re-ran the Quarto file to test this. Due to the randomness bug, variables `sex` and `address` in `df1` and `sex` in `df2` were not reproduced. We did not use `pseudopeople` 1.0.0 because it introduces new "row" (duplication) noise and it was released too late for this report. A future update should use the latest version.

```python
df1['ssn_before_swap'] = df1['ssn']
np.random.seed(0)  # for np.random.choice()
random.seed(1)     # for random.choice()
pct_to_swap = 1                                                          (1)
rows_to_swap = np.random.choice(df1.index, size=int(len(df1)*pct_to_swap/100))

```

```python
df1.loc[rows_to_swap, 'ssn'] = (
    df1.loc[rows_to_swap, 'ssn']
    .apply(lambda x: swap_two(x) if pd.notnull(x) else x)
)
# This simpler code fails with "TypeError: 'NoneType' object is not iterable":
# df1.loc[rows_to_swap, 'ssn'] = df1.loc[rows_to_swap, 'ssn'].apply(swap_two)

df1_ssn = df1.filter(like='ssn').dropna()
df1_ssn.loc[(df1['ssn'] != df1['ssn_before_swap'])]                    ②
```

① Swap 1% for `ssn`
② .loc prevents warning "Boolean Series key will be reindexed to match DataFrame index"

	ssn	ssn_before_swap
106	350816314	358016314
111	466611848	466611884
227	726905151	726901551
228	386148586	381648586
739	142245459	142244559
...	...	...
669901	610844020	601844020
669926	261718192	216718192
670128	395487820	394587820
670159	326956686	329656686
670169	493782382	493783282

Below, we do some basic error checking before saving the merged dataset, as CSV and as Parquet. As specified, the missingness is about 15% for `ssn`, 5% for `address`, and 1% for the other linkage variables (except the 95% for 'unit_number', which is discussed above):

```python
print('Merged df1 contains', df1.simulant_id.nunique(), 'unique ids, out of', len(df1)) ①
df1.shape                                                                   ②
df1.to_csv('df1.csv', index=False)                                          ③
df1.to_parquet('df1.parquet', engine = 'pyarrow', compression = 'gzip')     ④
```

① Confirm that these simulants have unique simulant ids
② Dimensions of the dataset `df1` (with `seed=0`, default)
③ Save to CSV (~73.6 MB).
④ Save to Parquet (~21.6 MB)

```
Merged df1 contains 670251 unique ids, out of 670251
```

In the dataset generation functions, the parameter seed for randomness defaults to 0 (`seed=0`). For the second artificial dataset to be created, `df2`, we arbitrarily instead set the seed to 1 (`seed=1`).

```
config2 = {
    'taxes_w2_and_1099': {
        'column_noise': {
            'ssn': {'leave_blank': {'cell_probability': 0.05}}          ①
        }
    },
    'decennial_census': {
        'column_noise': {
            'street_number': {'leave_blank': {'cell_probability': 0.00}},
            'street_name': {'leave_blank': {'cell_probability': 0.00}},
            'unit_number': {'leave_blank': {'cell_probability': 0.00}}
        }
    }
}
df2a = psp.generate_taxes_w2_and_1099(source = ospath,                  ②
    config=config2, seed=1)

print('Tax data contains', len(df2a.columns), 'columns (expect 24)')
df2a = df2a.filter(
  ['simulant_id', 'first_name', 'last_name', 'ssn'])

# there are multiple rows for simulants who had more than one employer
df2a = df2a.groupby('simulant_id').first().reset_index()
# df2a

# def my_concat_address_fields2(s):
#   s = s.filter(['street_number', 'street_name', 'unit_number'])
#   return ' '.join(s.dropna(how='all'))

df2b = psp.generate_decennial_census(source = ospath,                   ③
    config=config2, seed=1)
df2b['address'] = (df2b.street_number.fillna('')                        ④
    + ' '
    + df2b.street_name.fillna('')
)
percentage_blank = 1                                                    ⑤
num_blank = int(len(df1b) * percentage_blank / 100)
blank_indices = np.random.choice(df2b.index, num_blank, replace=False)
df2b.loc[blank_indices, 'address'] = np.nan
```

```

# df2b = df2b.replace(np.nan, '')   # new: replace nan with ''
# df2b['address'] = df2b.apply(my_concat_address_fields2, axis=1)
# df2b = df2b.replace(r'^\s*$', np.nan, regex=True)  # new: put nan back
df2b = df2b.filter(
    ['simulant_id', 'sex', 'address', 'city', 'state', 'zipcode',
        'street_number', 'street_name', 'unit_number', 'date_of_birth'])
# df2b
df2 = pd.merge(df2a, df2b)
```

① Custom configuration of 5% missing `ssn` and 0% missing "`address`"
② Create the dataframe `taxes_w2_and_1099` (not saved) from Rhode Island data
③ Create the dataframe `decennial_census` (not saved) from Rhode Island data
④ Combine all address columns into a composite column called `address`
⑤ Make 1% of cells in the new `address` column blank

As for `df1`, we can now for `df2` change variables as needed to make them similar to the required FCDS linkage variables. Unlike `df1`, we do not add transposition errors in `ssn`:

```
df2['ssn'] = df2['ssn'].str.replace("-", "")                                  ①
df2['first_name'] = df2['first_name'].str.upper()                             ②
df2['last_name'] = df2['last_name'].str.upper()                               ③
df2['address'] = df2['address'].str.upper()                                   ④
df2['city'] = df2['city'].str.upper()                                         ⑤
df2['sex'] = np.where(df2['sex'] == "Male", 1, 2)                             ⑥

df2['DOB'] = pd.to_datetime(df2['date_of_birth'], errors='coerce')
df2['DOB'] = df2['DOB'].astype('string').str.replace('-', '')                 ⑦
# print(df2['date_of_birth'])  # object         (in format mm/dd/yyyy)
# print(df2['DOB'])            # datetime64[ns] (cleaned date in format yyyy-mm-dd)
# print(df2['DOB'])            # string         (in format yyyymmdd)
```

① SSN without dashes as in NACCR data item #2320: Social Security Number
② First name in uppercase for NACCR data item #2240: First Name
③ Last name in uppercase for NACCR data item #2232: Last Name
④ Address in uppercase for NACCR data item #2350: Street (Number & Name)
⑤ City in uppercase for NAACCR data item #1810: City
⑥ Sex coded as in NAACCR data item #220: Sex: 1=Male, 2=Female
⑦ DOB, converted from `date_of_birth`, as in NAACCR data item #240: City

Below, we do basic error checking before saving the merged dataset, as CSV and as Parquet. As specified, the missingness is about 5% for `ssn`, 1% for `address`, and 1% for the other linkage variables (except the 94% missingness for 'unit_number', which is discussed on page 4):

```
print('Merged df2 contains', df2.simulant_id.nunique(), 'unique ids, out of', len(df2)) ①
df2.shape                                                                       ②
df2.to_csv('df2.csv', index=False)                                              ③
df2.to_parquet('df2.parquet', engine = 'pyarrow', compression = 'gzip')         ④
```

① Confirm that these simulants have unique simulant ids
② Dimensions of the dataset `df2` (with `seed=1`)
③ Save to CSV (~74.9 MB).
④ Save to Parquet (~22.0 MB)

```
Merged df2 contains 670214 unique ids, out of 670214
```

The created datasets (dataframes) `df1` and `df2` are representative of the Requestor ("finder file") and FCDS linkage files, respectively. Note that, typically, the Requestor file has a significant amount of missing SSN such as the specified 20% whereas for the FCDS the missingness of SSN is usually around the specified 5%.

Analysis of the Created DataFrames `df1` and `df2`

Missingness of `df1` (%, sorted), by variable:

```
df1.isnull().mean().round(6).mul(100).sort_values(
    ascending=False).to_frame('Missing (%)')
```

	Missing (%)
unit_number	94.7271
ssn	15.2697
ssn_before_swap	15.2697
address	4.9978
street_number	4.1593
DOB	2.6702
date_of_birth	1.0084
city	1.0032
zipcode	0.9949
state	0.9778
last_name	0.7133
first_name	0.7126
simulant_id	0.0000
sex	0.0000

	Missing (%)
street_name	0.0000

List first 5 rows and 5 variables of `df1`:

```python
df1[["simulant_id", "first_name", "last_name", "DOB", "ssn"]].head()
```

	simulant_id	first_name	last_name	DOB	ssn
0	1007_1001150	CHARLENE	MCMURRAY BATISTA	19791231	449632668
1	1007_1001333	SAVANNAH	REED	19991207	349511166
2	1007_1006629	ROSALIE	MIRANDA	19880209	023141049
3	1007_1006679	ERNEST	WISNESKI	19730524	553530413
4	1007_1009212	SAMUEL	WEIS	19900407	048133388

List last 5 variables of `df1`:

```python
df1[["sex", "address", "city", "state", "zipcode"]].head()
```

	sex	address	city	state	zipcode
0	2	32583 RAINVILLE AVENUE	W GREENWICH	RI	02861
1	2	1124 VILLA COURT NORTH	CHARLESTOWN	RI	02906
2	2	373 EUGENE LN	SCITUATE	RI	02908
3	1	11091 35TH AVENUE NORTHEAST	CUMBERLAND	RI	02893
4	1	3493 GOULD STR	WARWICK	RI	03920

Missingness of `df2` (%, sorted), by variable:

```python
df2.isnull().mean().round(6).mul(100).sort_values(
    ascending=False).to_frame('Missing (%)')
```

	Missing (%)
unit_number	94.7285
street_number	4.1596
ssn	3.6166
DOB	2.6508

	Missing (%)
zipcode	1.0148
address	1.0058
city	1.0045
state	0.9930
date_of_birth	0.9860
first_name	0.7502
last_name	0.7114
simulant_id	0.0000
sex	0.0000
street_name	0.0000

List first 5 rows and 5 variables of `df2`. Python defaults to `None` for missing (`null`):

```
df2[["simulant_id", "first_name", "last_name", "DOB", "ssn"]].head()
```

	simulant_id	first_name	last_name	DOB	ssn
0	1007_1001150	CHARLENE	MCMURRAY BATISTA	19791231	449632668
1	1007_1001333	SAVANNAH	REED	19991207	349518166
2	1007_1006629	ROSALIE	MIRANDA	19880209	None
3	1007_1006679	ERNEST	WISNESKI	19730524	553530413
4	1007_1009196	ANTHONY	MORAN	19790122	718020331

List last 5 variables of `df2`:

```
df2[["sex", "address", "city", "state", "zipcode"]].head()
```

	sex	address	city	state	zipcode
0	2	32583 RAINVILLE AVENUE	W GREENWICH	RI	02861
1	2	1124 VILLA COURT NORTH	CHARLESTOWN	RI	02906
2	2	373 EUGENE LN	SCITUATE	RI	02908
3	1	11091 35TH AVENUE NORTHEAST	CUMBERLAND	RI	02893
4	1	14420 VICTORY BOUL	PAWTUCKET	RI	02859

Creating a Dataset of Matches and Non-matches

There are 670,251 records in `df1`:

```python
df1.shape
```

```
(670251, 15)
```

There are 670,214 records in `df2`. Since `df2` is the smaller dataset, this is the maximum number of records that can be matched:

```python
df2.shape
```

```
(670214, 14)
```

The merged dataset is named `psp`. It has 680,238 records in total. The merge status variable is named `_merge`, as does `Stata`. Here `_merge` is also the true (actual) match status variable `actual`:

```python
psp = pd.merge(df1, df2, on='simulant_id', how='outer', indicator=True)
psp['actual'] = psp['_merge'].apply(lambda x: 'Match' if x == "both" else 'Non-match')
psp.shape
```

```
(680238, 30)
```

We save the dataframe `psp`. There are 670,214 linkable records, with 660,227 TP and 9,987 TN. There are also 10,024 TN in `df1` only but those TN records are not relevant since they come from the larger dataframe, and therefore are not used in the linkage.

```python
psp.to_csv('psp.csv', index=False)                                   ①
pd.crosstab(psp._merge, psp.actual, margins=True, margins_name='Total') ②
```

① Save dataframe `psp` to CSV
② Two-way frequency table to double-check that the variable `actual` is correct

actual	Match	Non-match	Total
_merge			
left_only	0	10024	10024
right_only	0	9987	9987
both	660227	0	660227
Total	660227	20011	680238

For easier analysis, we drop the non-linkable 10,024 records, keep only 3 variables (`simulant_id`, `_merge`, and `actual`), and save the dataframe as `psp_actual`. From now on, the variable `_merge` with values `both` and `right_only` is merely a placeholder. It should be replaced with variable `predicted` with values `Match` and `Non-match`, or with something similar such as variable `linked` with values `Link` and `Non-link`:

```
psp_actual = psp[['simulant_id', '_merge', 'actual']]                           ①
psp_actual = psp_actual.loc[psp_actual["_merge"] != 'left_only']                ②
psp_actual.to_csv('psp_actual.csv', index=False)                                ③
pd.crosstab(psp_actual['_merge'], psp_actual.actual, margins=True,
    margins_name="Total")                                                       ④
```

① Only keep these three variables
② Only keep these 670,214 matchable observations
③ Save dataframe `psp_actual` to CSV for easier analysis
④ Two-way frequency table of matches and non-matches. To be updated with predicted (linked) results.

actual _merge	Match	Non-match	Total
right_only	0	9987	9987
both	660227	0	660227
Total	660227	9987	670214

The remaining supplements will provide the predicted (linked) results for `fastLink`, `Splink`, and `Match*Pro`.

S2 fastLink Results on the pseudopeople Test Data

The first part of the supplement, S1, described how the FCDS created the test data using the Python package pseudopeople. This second part of the supplement, S2, describes the fastLink results on the pseudopeople test data. The structure of S2 is the same as the FCDS record linkage template using fastLink (unpublished 2022 FCDS Monograph) but with four important improvements:

1. The importance of Social Security Number (SSN) is now tested. fastLink has been updated from version 0.6 to 0.6.1. The new version has the Damerau-Levenshtein edit distance measure (`stringdist.method = "dl"`) which allows for adjacent transpositions (swap two numbers next to each other). It is especially useful for identifying a partial match on SSN because 2 digits off is usually an error unless it is an adjacent transposition.
2. The reporting tool R Markdown has been upgraded to Quarto. An example of a Quarto feature is code annotation.
3. The package renv for a **r**eproducible **env**ironment has been upgraded from version 0.15.2 to version 1.0.3. The 1.0.0 release recognized that renv is a mature product that has evolved through 30 releases in the last 4 years.
4. The pseudopeople datasets have been updated from a private development ("alpha") version by Abraham Flaxman to public test ("beta") version 0.8.3. It uses Python package pandas version 2.1.2.

Reproducibility

To create a new project in the RStudio IDE, use the Create Project command (available on the Projects menu and on the global toolbar). The `renv::init()` command converts a project to use renv. The R output is:

```
- Project 'V:/Testing/Monograph_2024/fastLink_v06' loaded. [renv 1.0.3].
```

The `renv::snapshot()` command saves any changes you make. Use `renv::restore()` to restore the project. The files to share with collaborators are `renv.lock`, `.Rprofile`, `renv/settings.json` and `renv/activate.R`; see renv.

Source Data

S1 created two datasets of simulated "Rhode Island" populations: df1 with 670,251 observations (using seed 0), and df2 with 670,214 observations (using seed 1). Both datasets exist as comma-separated values (CSV) files and Parquet files. The R script `fltest_0input.r` loads the CSV datasets and the code for loading the Parquet files are commented out:

```r
## Load the source data
# input files: df1, df2
# output files: dfA, dfB
source("fltest_0input.r")
```

Step 1: Attribute Alignment

We use the R script `fltest_1align.r` for cleaning the data:

```r
## Align the attributes (clean the source data)
# input files: dfA, dfB
# output files: (dfA2, dfB2) -- not saved
source("fltest_1align.r")
```

Table 1 shows the dimensions of the datasets. The data for Table 1 is in the file `fltest_table1.r`, and the code for displaying Table 1 is in the Quarto Markdown version of this report, that is, in `fltest.qmd`.

Table 1: The dimensions of the datasets

Dataset	Feature	Source	Cleaned
dfA	Number of observations	670251	670251
dfA	Number of variables	15	17
dfA	Percent joint missing-1	NA	9.75%
dfA	Percent joint missing-2	NA	18.63%
dfB	Number of observations	670214	670214
dfB	Number of variables	14	16
dfB	Percent joint missing-1	NA	6.01%
dfB	Percent joint missing-2	NA	7.54%

Note: "Joint missing-1" refers to the combination `first_name`, `last_name`, `sex`, `dob`, `zip` and `address`. "Joint missing-2" refers to the combination `first_name`, `last_name`, `sex`, `dob`, and `ssn`.

Markdown tables such as Table 1 are much less configurable than `kable` tables such as Table 2. For example, Table 2 has a formatted *Note* and striped rows whereas Table 1 does not. The Github Quarto issue is #6945.

Table 2 shows the missingness of the cleaned datasets, by variable.

Table 2: The missingness of the cleaned datasets, by variable

	dfA		dfB	
variable	# Missing	% Missing	# Missing	% Missing
req_pid	0	0.00	NA	NA
fcds_pid	NA	NA	0	0.00
ssn	102,345	15.27	24,239	3.62
address	33,498	5.00	6,741	1.01
zip	6,668	0.99	6,801	1.01
dob	17,897	2.67	17,766	2.65
sex	0	0.00	0	0.00
first_name	4,776	0.71	5,028	0.75
last_name	4,852	0.72	4,855	0.72
fname_dm	0	0.00	0	0.00
lname_dm	0	0.00	0	0.00

Note:
The dfA dataset (File 1) has 670,251 patient observations. The dfB dataset (File 2) has 670,214 patient observations. The data are simulated (artificial).

Step 2: Blocking

The blocking is done with exact matching `sex` (2 values) and k-means clustering on `first_name` (3 clusters), with a total of 6 blocks. For consistency with the FCDS template, we use the R script `fltest_2blocking.r` for blocking the data:

```
## Block the aligned data
# input files: dfA2, dfB2
# output files: (block_out object) -- not saved
source("fltest_2blocking.r")
```

Step 3: Record linkage

Links are declared matches, which are not necessarily true matches. The SSN variable `ssn` is missing 15.3% in the dataset `dfA` and 3.6% in the dataset `dfB`. The record linkage was done using the R script `fltest_3linkage.r`:

```r
## Create the patient matches (before manual review)
# warning messages are suppressed with `warning = FALSE`
# input files: block_out (object), dfA2, dfB2
# output files: (flobj_out object, matches) -- not saved
source("fltest_3linkage.r")
```

Table 3 provides frequencies of linkage pattern (variable `pattern`) by posterior matching probability (variable `posterior`):

Table 3: Frequencies of linkage pattern by posterior probability

Pattern	Posterior						
	0.95	0.96	0.97	0.98	0.99	1	Sum
0 2 2 2 2	0	0	0	0	0	5,305	5,305
0 NA 2 2 2	0	0	0	48	81	0	129
1 0 0 2 2	0	0	0	3	8	0	11
1 0 2 0 2	0	0	0	0	0	20	20
1 0 2 2 0	0	0	0	0	0	16	16
1 0 2 2 2	0	0	0	0	0	342	342
1 0 NA 2 2	0	0	0	0	0	23	23
1 2 0 0 2	0	0	0	0	15	0	15
1 2 0 2 2	0	0	0	0	0	439	439
1 2 2 0 0	0	0	0	0	0	18	18
1 2 2 0 2	0	0	0	0	0	405	405
1 2 2 2 0	0	0	0	0	0	628	628
1 2 2 2 2	0	0	0	0	0	12,303	12,303
1 2 NA 0 2	0	0	0	0	0	27	27
1 2 NA 2 0	0	0	0	0	9	18	27
1 2 NA 2 2	0	0	0	0	0	741	741
1 NA 0 2 2	0	0	0	0	0	6	6
1 NA 2 0 0	0	0	0	0	2	0	2
1 NA 2 0 2	0	0	0	0	0	12	12
1 NA 2 2 0	0	0	0	0	0	17	17
1 NA 2 2 2	0	0	0	0	0	271	271
1 NA NA 2 2	0	0	0	0	0	18	18
2 0 0 0 2	0	1	0	0	0	0	1
2 0 0 2 2	0	0	0	0	0	453	453
2 0 2 0 0	0	0	0	0	0	21	21
2 0 2 0 2	0	0	0	0	0	396	396

2 0 2 2 0	0	0	0	0	0	508	508
2 0 2 2 2	0	0	0	0	0	11,459	11,459
2 0 NA 0 2	0	0	0	0	0	18	18
2 0 NA 2 0	0	10	19	0	4	0	33
2 0 NA 2 2	0	0	0	0	0	654	654
2 2 0 0 2	0	0	0	0	0	507	507
2 2 0 2 0	0	0	0	0	0	663	663
2 2 0 2 2	0	0	0	0	0	14,345	14,345
2 2 2 0 0	0	0	0	0	0	630	630
2 2 2 0 2	0	0	0	0	0	13,154	13,154
2 2 2 2 0	0	0	0	0	0	19,131	19,131
2 2 2 2 2	0	0	0	0	0	389,545	389,545
2 2 NA 0 0	0	0	5	26	7	0	38
2 2 NA 0 2	0	0	0	0	0	752	752
2 2 NA 2 0	0	0	0	0	0	1,124	1,124
2 2 NA 2 2	0	0	0	0	0	22,350	22,350
2 NA 0 0 2	0	0	0	0	0	5	5
2 NA 0 2 0	0	0	3	8	2	0	13
2 NA 0 2 2	0	0	0	0	0	297	297
2 NA 2 0 0	0	0	0	0	0	10	10
2 NA 2 0 2	0	0	0	0	0	245	245
2 NA 2 2 0	0	0	0	0	0	392	392
2 NA 2 2 2	0	0	0	0	0	8,183	8,183
2 NA NA 0 2	0	0	0	0	0	17	17
2 NA NA 2 0	0	0	0	0	0	15	15
2 NA NA 2 2	0	0	0	0	0	495	495
NA 0 2 2 2	0	0	0	0	2,210	382	2,592
NA 2 2 0 2	0	0	0	0	1,497	1,536	3,033
NA 2 2 2 0	837	761	0	0	0	0	1,598
NA 2 2 2 2	0	0	0	0	0	89,306	89,306
NA NA 2 2 2	0	0	0	0	0	1,868	1,868
Sum	837	772	27	85	3,835	599,070	604,626

Note:
The `pattern` variable is a concatenation of the variables ssn, zip, dob, fname_dm, and lname_dm. The pattern values are 2 = Match, 1 = Partial Match, 0 = No Match, and NA = Missing.

Table 4 below, from the `confusion()` function, shows the linkage results *before* the clerical review as the confusion table. There are 604,626 links. The "Max Number of Obs to be Matched" (665,186) is less than the observations in the smaller dataset `dfB` (670,214) which is difficult to explain. The `confusion()` function does not have user documentation. For a developer comment, see fastLink issue #22. Another issue is that the duplicate row ids are removed before checking for them again in `dfB` which can result in duplicates in `dfB` despite deduplication, see issue #78.

Table 4: Confusion table from `fastLink`

	'True' Matches	'True' Non-Matches
Declared Matches	604,514.91	111.09
Declared Non-Matches	55.15	60,504.85

Note:
The 'True' Matches are only `fastLink` estimates

	results
Max Number of Obs to be Matched	665,186.00
Sensitivity (%)	99.99
Specificity (%)	99.82
Positive Predicted Value (%)	99.98
Negative Predicted Value (%)	99.91
False Positive Rate (%)	0.18
False Negative Rate (%)	0.01
Correctly Classified (%)	99.97
F1 Score (%)	99.99

Step 4: Canonicalization

The fourth and last record linkage step step is known as canonicalization. By default, `fastLink` deduplicates the matches into representative or "canonical" records. There are several *unsupervised* methods of canonicalization (Kaplan, Betancourt, and Steorts 2022). `fastLink` has two deduplication algorithms:

- a faster but less accurate "greedy" algorithm (default) which iteratively selects the maximum match probability for a given observation. The `fastLink` argument is `dedupe.matches`.
- a slower but more accurate "linear programming" algorithm (recommended but not default) which uses Winkler's solution. The `fastLink` argument is `linprog.dedupe`.

The R script `fltest_4canon.r` is available but not used for the canonicalization. For now, it is only placeholder code:

```
## Create the results data for the patient links (after manual review)
# input files: matches, dfA2, dfB2
# intermediate files: (possibles, reviewed, fltest_links) -- not used
# output files: fltest, fltest.csv (CSV version)
# source("fltest_4canon.r")
```

The largest advantage with using `pseudopeople` source data is that we know the *actual* true matches in the dataset `psp_actual`, as opposed to the *estimated* true matches or "true" matches or "links" in the dataset `matches`. It is a **true match** when the `simulant_id` in both linkage files are the same. It is a **true non-match** when the `simulant_id` in both linkage files are different. There are a few different ways to show this such as using `merge()`, `statar::join()` or `tidylog::full_join()` on the datasets `psp_actual` and `matches`. We used `statar::join()` (not shown) because it easily creates a new variable `merge`. From tabulating this new variable `merge`, we find that there are 10,024 non-matches only in `dfA`, 9,987 non-matches only in `dfB`, 660,227 matches and 670,214 rows in total. There are 41 FP, which are links with different `req_pid` and `fcds_pid` and where `simulant_id` are in both `dfA` and `dfB`. The count does not include possibly another 6 FP where `simulant_id` are in `dfA` only.

The most important linkage quality measure to the FCDS is FP. The linkage used the default `fastLink` method for de-duping matches. Table 5 below lists the 41 FP.

Table 5: List values of false positives (FP)

req_pid	fcds_pid	posterior	pattern	fp
1235_670643	1235_670644	1.00	2 0 0 2 2	1
1452_189659	1452_189662	1.00	2 2 2 0 2	1
1667_8269	1667_8267	1.00	2 2 0 0 2	1
1990_909799	1990_909801	1.00	2 2 0 0 2	1
2476_922610	2476_922611	1.00	2 2 0 0 2	1
2721_453341	2721_453339	1.00	NA 2 2 0 2	1
3298_428060	3298_428061	1.00	2 2 2 0 2	1
3481_381375	3481_381376	1.00	2 2 0 0 2	1
3481_615693	3481_615692	1.00	NA 2 2 0 2	1
3568_552537	3568_552538	1.00	2 2 2 0 2	1
40_28682	40_28686	1.00	2 0 NA 0 2	1
4400_278663	4400_278664	1.00	2 2 2 0 2	1
4637_233654	4637_233653	1.00	2 2 2 0 2	1
5072_872383	5072_872384	1.00	2 2 0 0 2	1
5475_550880	9247_36231	0.99	NA 0 2 2 2	1
5475_871168	5475_871171	0.99	NA 2 2 0 2	1
5619_28621	5619_28622	1.00	2 2 0 0 2	1

5670_764676	5670_764675	1.00	2 2 0 0 2	1
5812_445161	5812_445160	1.00	2 2 0 0 2	1
6487_3455	6487_3457	1.00	2 2 0 0 2	1
6545_493049	6545_493052	1.00	2 2 0 0 2	1
7264_109877	1935_698604	0.95	NA 2 2 2 0	1
7511_287109	7511_287111	1.00	2 2 0 0 2	1
7511_287111	7511_287109	1.00	2 2 0 0 2	1
7551_889857	7551_889856	1.00	2 2 0 0 2	1
7653_202950	7653_202948	1.00	2 2 0 0 2	1
781_175620	781_175619	0.99	NA 2 2 0 2	1
7985_384764	7985_384760	1.00	2 2 0 0 2	1
8129_172655	8129_172652	0.99	NA 2 2 0 2	1
8156_444517	8156_444518	1.00	2 2 2 0 2	1
8527_645266	8527_645264	1.00	2 2 0 0 2	1
8980_800863	8980_800860	1.00	2 2 0 0 2	1
9159_370494	9159_370492	1.00	2 2 0 0 2	1
9187_708655	9187_708653	1.00	2 2 2 0 2	1
9247_556126	9247_556127	1.00	2 2 0 0 2	1
9284_633649	9284_633647	1.00	2 2 0 0 2	1
9859_722895	9859_722894	1.00	2 2 0 0 2	1
9871_753609	9871_753608	1.00	NA 2 2 2 2	1
9871_924620	9871_924619	1.00	2 2 2 0 2	1
9888_254122	9888_254123	1.00	2 2 0 0 2	1
9888_254123	9888_254122	1.00	2 2 0 0 2	1
Total		40.91		41

Note:
A False Positive (FP) is defined as a mis-match of the `simulant_id` variables, here named `req_pid` for `dfA` and `fcds_pid` for `dfB`

The analysis dataset `mylinks` from the linkage has 604,626 links on these 4 variables: `req_pid`, `fcds_pid`, `posterior` and `pattern`. The analysis dataset `analysis` (not shown) has 670,214 observations (on the 4 variables in `mylinks` plus 4 more analysis variables).

```r
mylinks <- read_csv("fltest.csv", col_types = cols())
dim(mylinks)
```

```
[1] 604626      4
```

The merging of `psp_actual` (with the actual match status) and of `matches` from `fastLink` resulted in the components for the confusion table, see Table 6 below. Table 6 is hard coded, so it could be improved.

Table 6: Confusion Table using `fastLink`

	True Matches	**True Non-Matches**	**Total**
Links	604,579 TP	41 FP	604,620
Non-Links	55,648 FN	9,946 TN	65,594
Total	660,227	9,987	670,214

Note: There are 41 FP (possibly because of de-duplication). We excluded the 6 of 604,626 links where `simulant_id` are in `dfA` only. Therefore, the table only displays 604,620 links.

Sensitivity Analysis

Compared with Table 3, Table 7 below replaces variables `ssn`, `fname_dm`, and `lname_dm` with `address`, `first_name`, and `last_name`. Also, it replaces Damerau-Levenshtein partial match with exact match, and match threshold 0.95 with 0.98, and rounding precision 0.01 with 0.005. Table 7 displays 610,354 links in total, and 604,182 links having match probability 1, and 600,048 links with match probability of at least 0.995, when rounded to 0.005. For rounding off a 5, the standard is “go to the even digit”. For example, 0.998 is rounded to 1. The linkage run time was approximately 1 hour (60 minutes):

Table 7: Frequencies of linkage pattern by posterior probability (linkage 2)

Pattern	Posterior					
	0.98	0.985	0.99	0.995	1	Sum
0 2 2 0 2	0	0	0	1,053	0	1,053
0 2 2 2 2	0	0	0	0	22,939	22,939
0 NA 2 2 2	0	0	0	145	319	464
2 0 2 0 2	0	0	0	0	610	610
2 0 2 2 0	0	0	0	232	75	307
2 0 2 2 2	0	0	0	0	12,668	12,668
2 0 2 2 NA	0	0	0	84	90	174
2 0 NA 2 2	0	127	137	94	0	358
2 2 0 0 2	0	0	0	457	801	1,258
2 2 0 2 2	0	0	0	0	15,639	15,639

2 2 0 2 NA	0	0	102	45	0	147
2 2 2 0 0	0	0	0	0	981	981
2 2 2 0 2	0	0	0	0	20,853	20,853
2 2 2 0 NA	0	0	0	0	275	275
2 2 2 2 0	0	0	0	0	20,739	20,739
2 2 2 2 2	0	0	0	0	432,044	432,044
2 2 2 2 NA	0	0	0	0	6,575	6,575
2 2 NA 0 2	0	0	0	0	1,321	1,321
2 2 NA 2 0	0	0	439	177	0	616
2 2 NA 2 2	0	0	0	0	24,800	24,800
2 2 NA 2 NA	0	0	171	68	127	366
2 NA 0 2 2	0	0	0	0	317	317
2 NA 2 0 0	0	0	11	8	0	19
2 NA 2 0 2	0	0	0	0	403	403
2 NA 2 0 NA	0	0	0	0	7	7
2 NA 2 2 0	0	0	0	0	408	408
2 NA 2 2 2	0	0	0	0	9,154	9,154
2 NA 2 2 NA	0	0	0	0	164	164
2 NA NA 0 2	14	3	3	0	0	20
2 NA NA 2 2	0	0	0	0	559	559
NA 0 2 2 2	0	0	0	562	274	836
NA 2 2 0 2	0	0	0	0	1,415	1,415
NA 2 2 2 0	0	0	0	588	704	1,292
NA 2 2 2 2	0	0	0	0	28,904	28,904
NA 2 2 2 NA	0	0	0	0	436	436
NA 2 NA 2 2	0	505	526	602	0	1,633
NA NA 2 0 2	0	0	0	19	7	26
NA NA 2 2 2	0	0	0	0	574	574
Sum	14	635	1,389	4,134	604,182	610,354

Note:
The `pattern` variable is a concatenation of the variables address, zip, dob, first_name, and last_name. There is no variable ssn. The threshold is 0.98. The values are rounded to 0.005. The pattern values are 2 = Match, 1 = Partial Match, 0 = No Match, and NA = Missing.

Table 8 below provides the estimated confusion table for the new linkage results. There are 610,354 estimated links (that is, 610,304.9 TP + 49.1 FP).

Table 8: Confusion table from `fastLink`

	'True' Matches	'True' Non-Matches
Declared Matches	610,304.9	49.1
Declared Non-Matches	49.3	54,782.7

Note:
The 'True' Matches are only `fastLink` estimates

	results
Max Number of Obs to be Matched	665,186.00
Sensitivity (%)	99.99
Specificity (%)	99.91
Positive Predicted Value (%)	99.99
Negative Predicted Value (%)	99.91
False Positive Rate (%)	0.09
False Negative Rate (%)	0.01
Correctly Classified (%)	99.99
F1 Score (%)	99.99

Table 9 below shows the confusion table with 599 FP.

The change from 41 FP to 599 FP is most likely mostly due to replacing the person-identifier `ssn` with the household identifier `address`. To emphasize this, Table 9 adds the title text "without `ssn`":

Table 9: Confusion Table using `fastLink` without `ssn`

	True Matches	**True Non-Matches**	**Total**
Links	609,594 TP	599 FP	610,193
Non-Links	50,633 FN	9,388 TN	60,021
Total	660,227	9,987	670,214

Note: There are 599 FP (possibly because of de-duplication). We excluded the 161 of 610,354 links where `simulant_id` are in `dfA` only. Therefore, the table only displays 610,193 links.

Linkage Performance

Record linkage is performed with two different objectives: matching of entire databases ("matchability") and identification of individuals ("identifiability") (Ansolabehere and Hersh

2017). Usually, the record linkage process is the same for both objectives. In practice, matchability focuses on increasing the total number of declared matches (links) whereas identifiability focuses on reducing the wrong matches (FP). The focus at the FCDS is on having 0 FP.

The `fastLink` record linkage was a gradual process. These were the results in terms of links and FP:

- Estimated 111 FP if using `ssn` (linkage 1 with 604,626 links), see Table 4.
- Actual **41 FP** if using `ssn` (linkage 1 with 604,620 links), see Tables 5 and 6.
- Estimated 49 FP if not using `ssn` (linkage 2 with 610,354 links), see Table 8.
- Actual **599 FP** if not using `ssn` (linkage 2 with 610,193 links), see Table 9.

It is debatable whether to count links that are in `dfA` only as FP because the actual match status is “Missing” rather than “Non-match”. To give `fastLink` the benefit of doubt, we do not count those 6 links in linkage 1 and those 161 links in linkage 2 in the confusion tables with the actual match status; we refer to “matched links” instead of to only “links”. Therefore, the best result of `fastLink` in terms of FP is 41 FP (not 47 FP) from linkage 1 which used `ssn`.

The run time of the first linkage was approximately 4 hours (240 minutes). The run time of the second linkage was approximately 1 hour (60 minutes). The difference in run time is primarily because of the change from partial matching (of `ssn`) to exact matching (on `address`). Note that these are the first run times of each linkage. Subsequent run times for report editing are much faster thanks to cached results.

The results so far in terms of FP have counted all links. The results are worse when “Uncertain” or “Possible” record pairs are considered. These are links with match probability 0.95-0.94 in the first linkage, and links with match probability 0.98-0.997 in the second linkage.

Linkage 1 had 41 FP. The test results with linkage patterns in Table 3 for linkage 1 (with rounding to 0.01) indicate that “many” links with match probability 0.95-0.97 (that is, <0.98) are likely FP. Linkage 2 had 599 FP. The equivalent Table 7 for linkage 2 (with rounding to 0.005) indicate that “many” links with match probability 0.98-0.995 (that is, <1) are likely FP. These test results are similar to the published results from the developers.

Unfortunately, FP can be difficult to resolve efficiently. A known problem is that `first_name` and `sex` correlate. A limited review of the 41 FP showed that most FP are because of no match on on both `first_name` and `sex`. We addressed this in Tables 10 and 11 below.

Table 10 shows an improved confusion table with 19 FP when `ssn` is available.

The change from 41 FP to 19 FP is because of requiring a match on `first_name` and `sex`:

Table 10: Confusion Table using `fastLink` with `ssn`. Match on `first_name` and `sex`.

	True Matches	**True Non-Matches**	**Total**
Links	571,568 TP	19 FP	571,587
Non-Links	88,659 FN	9,968 TN	98,627
Total	660,227	9,987	670,214

Note: There are 19 FP (possibly because of de-duplication). We excluded the 33,033 of 604,620 links where there is no match on `first_name`, `sex` and `dob`. Therefore, the table only displays 571,587 links.

Table 11 shows an improved confusion table with 109 FP when `ssn` is unavailable.

The change from 599 FP to 109 FP is because of requiring a match on `first_name`, `sex`, and `dob`. If we require a match on `first_name` and `sex` but not on `dob`, then there are 226 FP. We prefer to be conservative, to have 109 FP rather than 226 FP before the clerical review:

Table 11: Confusion Table using `fastLink` without `ssn`. Match on `first_name`, `sex` and `dob`.

	True Matches	**True Non-Matches**	**Total**
Links	532,650 TP	109 FP	532,759
Non-Links	127,577 FN	9,878 TN	137,455
Total	660,227	9,987	670,214

Note: There are 109 FP (possibly because of de-duplication). We excluded the 77,434 of 610,193 links where there is no match on `first_name`, `sex` and `dob`. Therefore, the table only displays 532,759 links.

Recommendation

Based on the `fastLink` test results, the recommendation is to use the match threshold 0.98, and the rounding 0.005 as in the second linkage – or maybe even the higher match threshold 0.99. This assumes that `fastLink` is used. Part three of the supplement, S3, will describe the Splink results on the `pseudopeople` test data. The last part 4, S4, will describe the Match*Pro results on the test data.

⚠ Warning – Need to Update `fastLink`

- A new major release of `fastLink` with improved blocking and Active Learning has been promised. The FCDS does not know when that will happen. There is a need to update `fastLink`.

S3 `Splink` Results on the `pseudopeople` Test Data

The first part of the supplement, S1, described how the FCDS created the test data using the Python package `pseudopeople`. The second part of the supplement, S2, described the `fastLink` results on the `pseudopeople` test data. The structure of S2 was the same as the FCDS record linkage template using `fastLink` (unpublished 2022 FCDS Monograph) but with three important improvements, as discussed in S2. This third part of the supplement, S3, describes the `Splink` results on the `pseudopeople` test data.

> Tip to Try `pseudopeople` 1.0.0
>
> The data are from `pseudopeople` 0.8.3, see S1. The current `pseudopeole` 1.0.0 data will have some duplicates.

Reproducibility

As described in S2, we used `venv` for the Python virtual environment tool, because `venv` requires no installation.[1]

Source Data

```
# Step 0: Input data
import os                                                                    ①
import warnings

import numpy as np                                                           ②
```

[1]The file `monograph_2024_supplement3.qmd` reproduces the report. The file `monograph_2024_supplement3_nossn.qmd` produced the data for Table 2, confusion table without `ssn`.

```python
import pandas as pd
import time                                  #
import altair as alt

import splink
splink.__version__                                                         ③

# import pseudopeople as psp
# !date

# --- !pip install pseudopeople
# psp.__version__                            #
warnings.filterwarnings('ignore')                                          ④
```

① Import the Python Standard Library
② Import modules
③ The current `Splink` version is 3.9.14.
④ Ignore warnings (otherwise displayed in PDF)

```
'3.9.14'
```

Despite the efforts of configuring `venv` for `pseudopeople` in S1, there were two more Quarto configuration issues: how to remove a Jupyter kernel and how to set a Windows environment variable for QUARTO_PYTHON. A non-Python user would not be able to resolve those configuration issues. `splink` requires basic Python user skills.

The most common Python package for data analysis is `pandas` (abbreviated `pd`). Here we read the three CSV datasets from `pseudopeople` into `pandas`:

```python
# sample dataset:
# df1a = psp.generate_taxes_w2_and_1099()

# Rhode Island dataset:
# print ("Current working dir : %s" % os.getcwd())
# df1a = psp.generate_taxes_w2_and_1099(source = os.getcwd())

df1 = pd.read_csv("../pseudopeople/df1.csv", dtype={'DOB': str, 'ssn': str})
df2 = pd.read_csv("../pseudopeople/df2.csv", dtype={'DOB': str, 'ssn': str})
df_labels = pd.read_csv("../pseudopeople/psp_actual.csv")
```

Step 1: Attribute Alignment

As an example of attribute alignment, we rename two variables and display the last 5 records. By default, Python only displays the last output. Here, it means that Python only displays the last 5 records from `df2` (not from `df1`). `NaN` ("Not a Number") is missing, similar to `None`:

```python
# Step 1: Attribute alignment
df1.rename(columns={"zipcode": "zip", "date_of_birth": "dob"}, inplace=True) ①
df2.rename(columns={"zipcode": "zip", "date_of_birth": "dob"}, inplace=True)
df_labels.loc[df_labels.actual=="Match", 'clerical_match_score'] =1.0
df_labels.loc[df_labels.actual=="Non-match", 'clerical_match_score'] =0.0
df_labels = df_labels.filter(['simulant_id', 'clerical_match_score'])
df_labels.head()
df_labels.tail()
df1.head()
df1.tail()
df2.head()
df2.tail()
```

	simulant_id	clerical_match_score
0	1007_1001150	1.0
1	1007_1001333	1.0
2	1007_1006629	1.0
3	1007_1006679	1.0
4	1007_1009196	0.0

	simulant_id	clerical_match_score
670209	99_978308	1.0
670210	99_979738	1.0
670211	99_981197	1.0
670212	99_987024	1.0
670213	99_988029	1.0

	simulant_id	first_name	last_name	ssn	sex	address
0	1007_1001150	CHARLENE	MCMURRAY BATISTA	449632668	2	32583 RAINVILLE AVENUE
1	1007_1001333	SAVANNAH	REED	349511166	2	1124 VILLA COURT NORTH
2	1007_1006629	ROSALIE	MIRANDA	023141049	2	373 EUGENE LN
3	1007_1006679	ERNEST	WISNESKI	553530413	1	11091 35TH AVENUE NORTHEAST

	simulant_id	first_name	last_name	ssn	sex	address
4	1007_1009212	SAMUEL	WEIS	048133388	1	3493 GOULD STR

	simulant_id	first_name	last_name	ssn	sex	address
670246	99_978308	JAVIER	GUYER	746654363	1	6517 PEACEPIPE T
670247	99_979738	KATHLEEN	TONN	483701187	2	1010 CAT HOLLOW
670248	99_981197	DEREK	MORENO-AVENDANO	NaN	1	10083 GARRISON L
670249	99_987024	JUSTIN	EVANS	308590487	1	18 CLEVELAND AV
670250	99_988029	CHRISTOPHER	CAGE	240266261	1	3327 N PALM AVE

	simulant_id	first_name	last_name	ssn	sex	address
0	1007_1001150	CHARLENE	MCMURRAY BATISTA	449632668	2	32583 RAINVILLE AVENU
1	1007_1001333	SAVANNAH	REED	349518166	2	1124 VILLA COURT NORT
2	1007_1006629	ROSALIE	MIRANDA	NaN	2	373 EUGENE LN
3	1007_1006679	ERNEST	WISNESKI	553530413	1	11091 35TH AVENUE NOR
4	1007_1009196	ANTHONY	MORAN	718020331	1	14420 VICTORY BOUL

	simulant_id	first_name	last_name	ssn	sex	address
670209	99_978308	JAVIER	GUYER	746654363	1	6517 PEACEPIPE T
670210	99_979738	KATHLEEN	TONN	483701187	2	1010 CAT HOLLOW
670211	99_981197	DEREK	MORENO-AVENDANO	032571364	1	10083 GARRISON L
670212	99_987024	JUSTIN	EVANS	308590487	1	18 CLEVELAND AV
670213	99_988029	CHRISTOPHER	CAGE	240Z66261	1	3327 N PALM AVE

All linker input frames must have the same set of columns. We can use `df.drop()` to drop the unique column `ssn_before_swap`. We also do not need the column `unit_number` and `dob` because we know from supplement 1 that `unit_number` has about 95% missing and that `DOB` is the cleaned variable for date of birth. However, we prefer to use `.loc[]` to select the columns we want. The code should focus on the columns we want to use, not on the columns we want to drop. We rename `DOB` to `dob` to be consistent with lowercase but ideally this should be done in Supplement 1:

```
df1 = df1.loc[:,['simulant_id', 'first_name', 'last_name', 'ssn',
    'sex', 'address', 'city', 'state', 'zip', 'street_number',
    'street_name', 'DOB']]
df2 = df2.loc[:,['simulant_id', 'first_name', 'last_name', 'ssn',
```

```python
    'sex', 'address', 'city', 'state', 'zip', 'street_number',
    'street_name', 'DOB']]
# df1 = df1.drop(['ssn_before_swap', 'unit_number', 'dob'], axis=1)
# df2 = df2.drop(['unit_number', 'dob'], axis=1)
df1.rename(columns={"DOB": "dob"}, inplace=True)
df2.rename(columns={"DOB": "dob"}, inplace=True)
```

A comprehensive book on Pandas 2.2.0 is "Effective Pandas 2" (Harrison 2024). A comprehensive tutorial book on `Splink` 3.9.5 is "Hands-On Entity Resolution" (Shearer 2024).

Step 2: Blocking

Step 2 in the linkage process is deterministic matching, known as blocking. `Splink` defines a linkage model with a settings dictionary. `Splink` 4 is expected to be backwards compatible. `Splink` has two types of blocking: for prediction, and for estimation. The blocking for prediction is specified in the settings dictionary.

```python
# Step 2: Blocking
start_time = time.time()                                                ①

from splink.duckdb.linker import DuckDBLinker                           ②
from splink.duckdb import comparison_library as cl
from splink.duckdb.blocking_rule_library import block_on   # new in v3.9.5

settings = {                                                            ③
    "unique_id_column_name": "simulant_id",                             ④
    "link_type": "link_only",                                           ⑤
    "blocking_rules_to_generate_predictions": [                         ⑥
        block_on(["last_name", "dob"]),
        block_on(["first_name", "dob"]),
        block_on(["zip", "sex", "last_name", "first_name"]),
    ],
    "comparisons": [                                                    ⑦
        cl.jaro_winkler_at_thresholds("first_name", [0.9, 0.7]),
        cl.jaro_winkler_at_thresholds("last_name", [0.9, 0.7]),
        cl.levenshtein_at_thresholds("dob", [1,2]),            #
        cl.levenshtein_at_thresholds("street_number", [1,2]),  #
        cl.levenshtein_at_thresholds("street_name", [1,2]),
        cl.damerau_levenshtein_at_thresholds("ssn", [1, 2]),
        cl.exact_match("zip"),
        cl.exact_match("sex"),
    ],
    "retain_matching_columns": True,                                  ⑧
    "retain_intermediate_calculation_columns": True,
    "em_convergence": 0.0001                                          ⑨
}
```

① start to time the code
② DuckDB-specific code, see Choosing a backend
③ Define a `Splink` model with a settings dictionary
④ unique_id_column_name is required if not named `unique_id` (default).
⑤ Link type: Linkage *without* deduplication.
⑥ Blocking for prediction: Ensure comparisons are generated for all true matches
⑦ Customise comparisons using the ComparisonLibrary and, if wanted, Term-Frequency adjustments
⑧ See guide to `Splink` settings for more details
⑨ Default EM convergence tolerance. A smaller value such as 0.01 tends to result in slightly more links but of low quality.

The `Splink` linker object holds the record linkage model. `Splink` is highly scalable by allowing for different database backends. DuckDB is the default and is used here:

```python
# Step 3: Record Linkage
linker = DuckDBLinker([df1, df2], settings,                           ①
     input_table_aliases=["df1_large", "df2_small"])
```

① The linker object

The blog for `Splink` 3.9.10 mentions the completeness chart. The missingness chart is still available. In practice, the newer and more concise completeness chart replaces the missingness chart, see Figure 1 below. The blocking for prediction works cumulatively, as shown in Figure 2 below:

```python
c_completeness = linker.completeness_chart(cols=['simulant_id',
    'first_name', 'last_name', 'ssn', 'sex', 'address', 'city',
    'state', 'zip', 'street_number', 'street_name', 'dob'])
c_completeness.save("c_completeness.png")                             ①
# Block 1 = 610,592 obs, block 2 = 685,396 obs, and block 3 = 577,574 obs
block_chart = linker.cumulative_num_comparisons_from_blocking_rules_chart()
block_chart.save("block.png", scale_factor=2, ppi=300)                ②
```

① Completeness chart
② Blocking chart

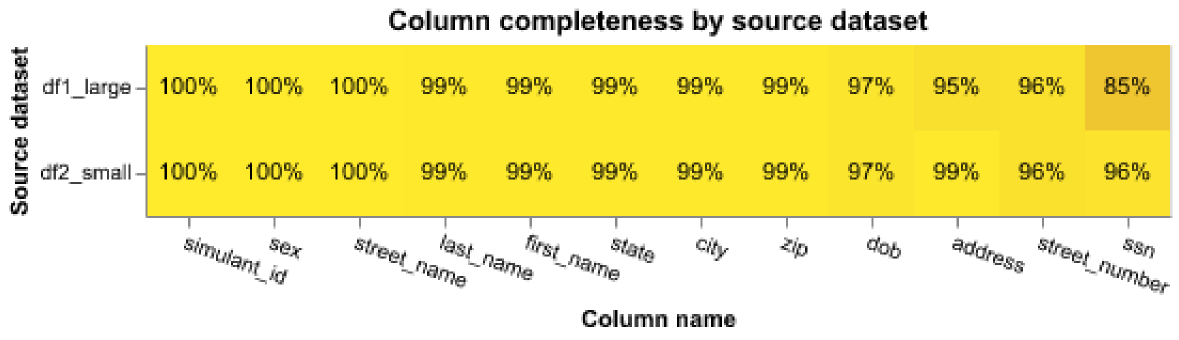


Figure 1: Completeness Chart

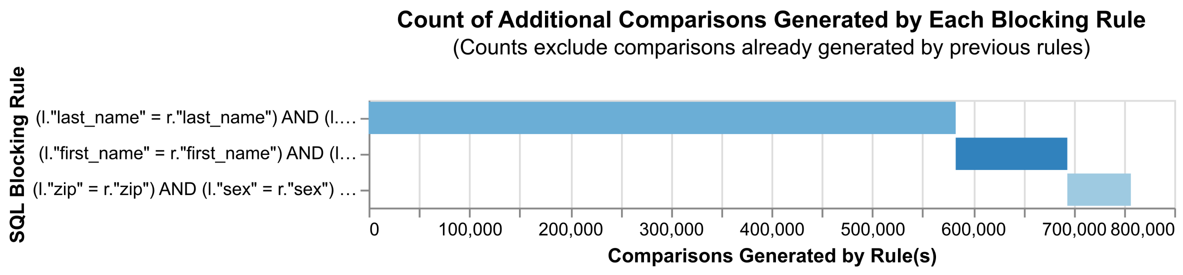


Figure 2: Blocking Chart

Step 3: Record linkage

Step 3 in the linkage process is the actual linkage. Blocking for model training may exclude true matches. Work in progress is auto blocking. Below, we manually create blocking rules for EM model training.

```
# https://moj-analytical-services.github.io/
#     splink/topic_guides/blocking/model_training.html
blocking_rule_4 = block_on(["first_name", "last_name", "street_number"])
count = linker.count_num_comparisons_from_blocking_rule(blocking_rule_4)
print(f"Comparisons generated by '{blocking_rule_4.blocking_rule_sql}': {count:,.0f}")
# -> 540,867

blocking_rule_5 = block_on(["sex", "zip", "street_name", "dob"])
count = linker.count_num_comparisons_from_blocking_rule(blocking_rule_5)
print(f"Comparisons generated by '{blocking_rule_5.blocking_rule_sql}': {count:,.0f}")
# -> 550,091

```

```python
blocking_rule_6 = block_on(["substr(ssn, 1,6)"])
count = linker.count_num_comparisons_from_blocking_rule(blocking_rule_6)
print(f"Comparisons generated by '{blocking_rule_6.blocking_rule_sql}': {count:,.0f}")
# -> 956,664
```

Three parameters need to be estimated for the record linkage: `lambda`, `u` and `m`. First, we estimate model parameter `lambda`, also known as the `prior`. A minor issue is that the output does not line wrap when rendering to PDF. Similarly, sometimes code does not line wrap when rendering to PDF, see Quarto issue 5343:

```python
deterministic_rules = [
    "l.ssn = r.ssn"
]
linker.estimate_probability_two_random_records_match(deterministic_rules, recall=0.85)
```

```
Probability two random records match is estimated to be  1.42e-06.
This means that amongst all possible pairwise record comparisons, one in 705,054.21 are
expected.
```

Estimate the probability `u` that wrong matches match. The `set seed` parameter is not always working, see issue 1743.

```python
linker.estimate_u_using_random_sampling(max_pairs=1e9, seed=123)                ①
```

① Linker function estimate_u_using_random_sampling()

It is the most time consuming part. Show run time in elapsed minutes:

```python
print("--- %s elapsed minutes ---" % round((time.time() - start_time) / 60))
```

```
--- 21 elapsed minutes ---
```

Estimate the probability `m` that true matches match:

```python
# (The text is truncated in PDF. Could be resolved by editing the intermediate md file.)
training_blocking_rule = block_on(["first_name", "last_name", "street_number"]) ①
training_session_names = linker.estimate_parameters_using_expectation_maximisation( ②
    training_blocking_rule, estimate_without_term_frequencies=True)

training_blocking_rule = block_on(["sex", "zip", "street_name", "dob"])     ③
training_session_names = linker.estimate_parameters_using_expectation_maximisation(
    training_blocking_rule)
training_session_names.match_weights_interactive_history_chart()

# training_blocking_rule = block_on(["substr(ssn, 1,6)"])                ④
# training_session_names = linker.estimate_parameters_using_expectation_maximisation(
#    training_blocking_rule)
# training_session_names.match_weights_interactive_history_chart()
```

① Blocking for estimation (training), first pass
② Estimate the probability `m` that true matches match
③ Blocking for estimation (training), second pass

The `Splink` match weight chart shows the results of a trained `Splink` model. The exact match comparison takes priority. Records that do not fall within a comparison level are allocated to the rest of the comparison levels. On these data, because of 1% transposition error of `ssn` in `df1`, the Damerau-Levenshtein comparison for `ssn` had a larger weight (18) than the second strongest linkage variable `dob` (14):

```python
c_matchweights = linker.match_weights_chart()                          ①
c_matchweights.save("c_matchweights.png")
```

① Match Weight Chart

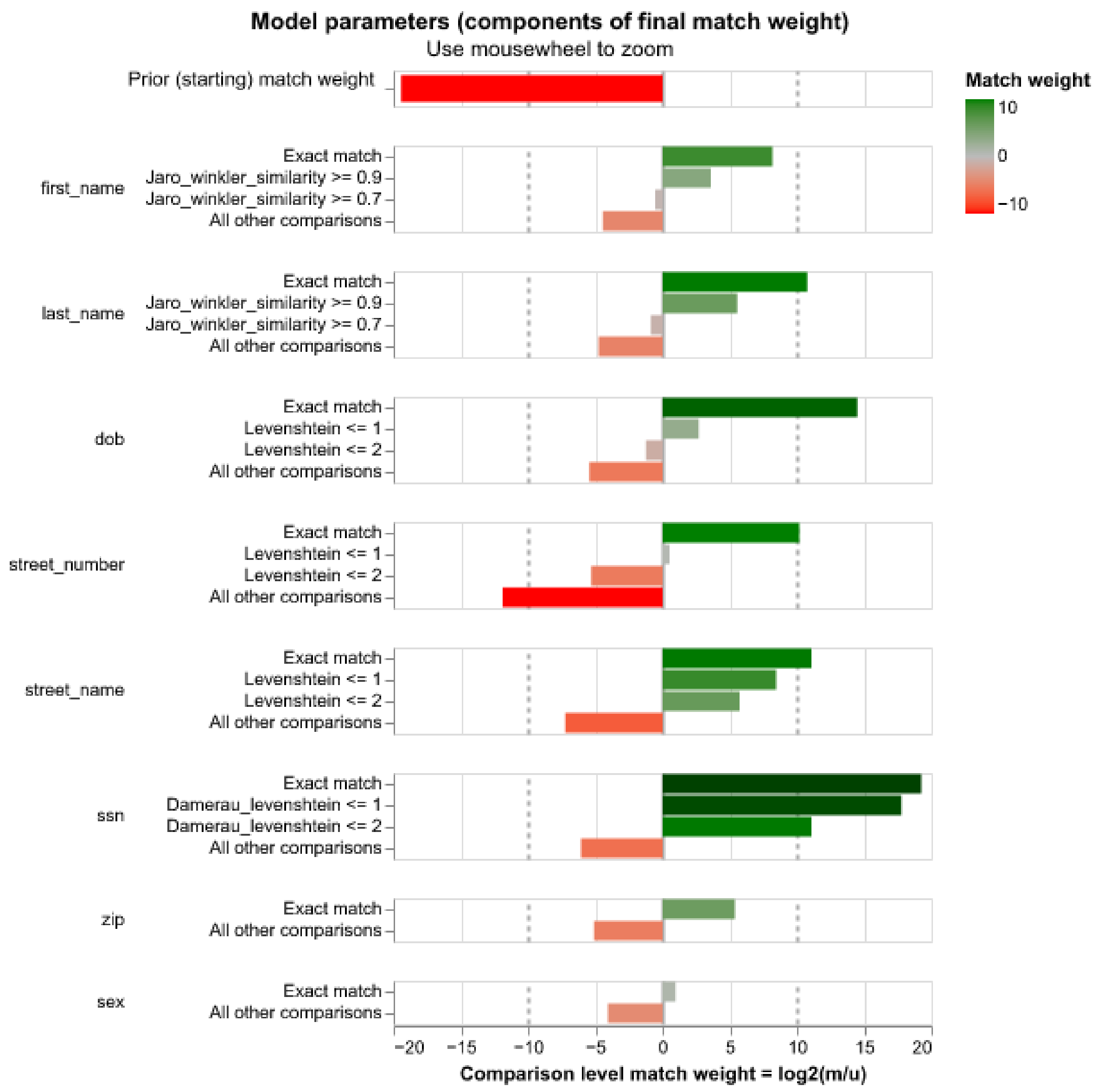


Figure 3: Match Weight Chart

Threshold selection is a key decision point within a linkage pipeline. One of the major benefits of probabilistic linkage versus a deterministic (i.e., rules-based) approach is the ability to choose the amount of evidence required for two records to be considered a match (i.e., a threshold).

When you have decided on the metrics that are important for your use case, you can use the Threshold Selection Tool to get a first estimate for what your threshold should be. It is a new feature in `Splink` 3.9.14. At minimum, it requires a single ground truth column as illustrated in an example titled Evaluation from ground truth column.

In principle, it would be a single column similar to `psp_actual`. However, in practical applications having fully labelled data is rare, which is probably why this simpler approach currently does not work; see issue 2059. Instead, we must use threshold_selection_tool_from_labels_table. There is no template code available, and therefore we opened discussion issue 2082. In the meanwhile, we have to create the ground truth manually. The first step is to create a dataframe of predictions without specifying the threshold:

```
# create dataframe of predictions
df_predict = linker.predict()    # SplinkDataFrame
df_predict.to_csv("splink_predictions.csv", overwrite=True) # 757,038
```

The next, more difficult step is to create the required ground truth table:

```
df_labels.rename(columns={"simulant_id": "simulant_id_r"}, inplace=True)
df_predict2 = df_predict.as_pandas_dataframe()
df_chart = pd.merge(df_labels, df_predict2, on='simulant_id_r',
    how='left', indicator=True, validate='1:m')                          ①

df_chart = df_chart.sort_values(by=['simulant_id_l', 'simulant_id_r'], kind='stable',
    na_position='first')  #
df_chart['dup'] = df_chart.duplicated(keep='first', subset=['simulant_id_r']) #
df_chart = df_chart.drop_duplicates(subset=['simulant_id_r'], keep='first')  #

# df.loc[<condition>, 'newvar'] = <expression>
df_chart.loc[df_chart._merge=="left_only", 'simulant_id_l'] = df_chart.simulant_id_r
df_chart.loc[df_chart._merge=="left_only", 'source_dataset_l'] = 'df1_large'
df_chart.loc[df_chart._merge=="left_only", 'source_dataset_r'] = 'df2_small'

df_chart.loc[(df_chart._merge=="left_only") & (df_chart.clerical_match_score==0),
    'simulant_id_l'] = df_chart.simulant_id_l[0]                         ②

labels2 = df_chart.filter(['source_dataset_l', 'simulant_id_l',
    'source_dataset_r', 'simulant_id_r',  'clerical_match_score'])
labels2 = labels2.reset_index(drop=True)                                 ③
labels2.head(5)

labels2.to_csv("labels.csv", index=False)                                ④
labels_table2 = linker.register_labels_table(labels2)

c_threshold = linker.threshold_selection_tool_from_labels_table(labels_table2)  # graph (PDF
c_threshold.save("c_threshold.png", scale_factor=2)                    ②

pd.crosstab(df_chart._merge, df_chart.clerical_match_score, margins=True, margins_name='Tota
```

① 1:m merge
② The first matched value ('1007_1001150') for TN
③ Get back the default index
④ Save to CSV to enable error checking later

	source_dataset_l	simulant_id_l	source_dataset_r	simulant_id_r	clerical_match_score
0	df1_large	1007_1001150	df2_small	1007_1009196	0.0
1	df1_large	1007_121432	df2_small	1007_121432	1.0
2	df1_large	1007_1001150	df2_small	1007_15116	0.0
3	df1_large	1007_164005	df2_small	1007_164005	1.0
4	df1_large	1007_1001150	df2_small	1007_177787	0.0

clerical_match_score	0.0	1.0	Total
_merge			
left_only	8635	12582	21217
both	1352	647645	648997
Total	9987	660227	670214

```
c_threshold = linker.threshold_selection_tool_from_labels_table(labels_table2) ①
c_threshold.save("c_threshold.png", scale_factor=2)                    ②
```

① Threshold selection tool

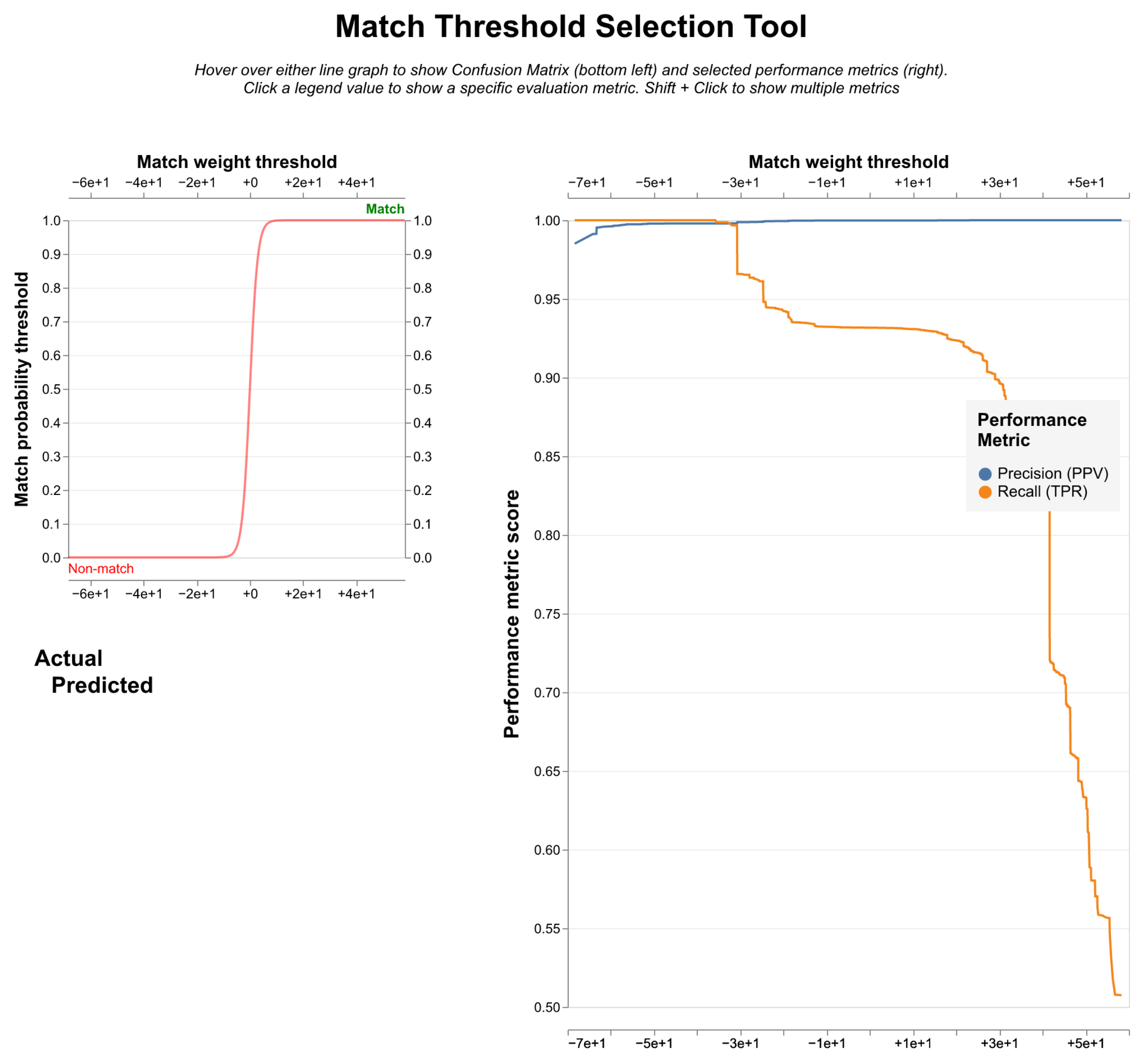


Figure 4: Threshold Selection Tool

The threshold selection tool should use ordinary numbers, not scientific notation. See issues 2112 (resolved in version 3.9.15) and 2070 (not yet resolved).

The Threshold Selection Tool suggests that, as expected, we need at minimum a threshold match probability of 0.95. Therefore, we now predict the results with a specified threshold match probability of 0.95:

```python
# Linker is a duckdb linker with link_type set to "link_only"
results = linker.predict(threshold_match_probability=0.95)     ①
records_to_plot = results.as_record_dict(limit=10)
```

① Predict which records match using the blocking rules in “step 2” and threshold 0.95.

Unlike `fastLink`, `Splink` requires extra code for removing possible duplicates from the record linkage. The required extra code below is commented on little to save space, and to avoid confusion because deduplication is not related to the linkage itself:

```
                                                                          ①
sql = f"""
with ranked as

(
select *,
row_number() OVER (
    PARTITION BY simulant_id_l order by match_weight desc
    ) as row_number
from {results.physical_name}
)

select *
from ranked
where row_number = 1
"""
results = linker.query_sql(sql)                                           ②
```

① SQL “f-string” code
② Remove duplicates using SQL

Step 4: Canonicalization

Step 4 of the linkage process is canonicalization. The `Splink` waterfall chart shows the breakdown of the match weight for pairs of records, which is useful for clerical review:

```
c_waterfall = linker.waterfall_chart(records_to_plot,
    filter_nulls=False, remove_sensitive_data=True)                       ①
c_waterfall.save("c_waterfall.png", scale_factor=2)                       ②
```

① Create variable `c_waterfall` for waterfall chart
② Save the waterfall chart using the new, simpler Altair 5 syntax

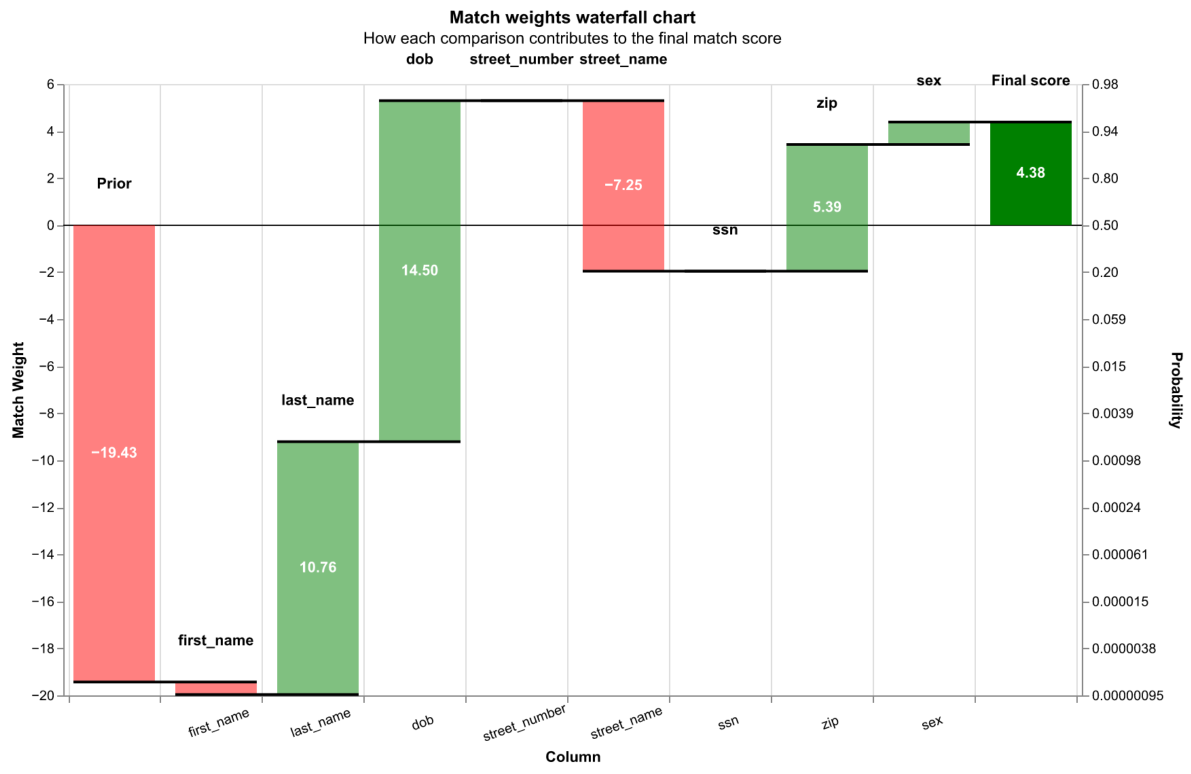


Figure 5: Waterfall Chart

The water fall chart here shows a low-quality match. For example, the Final Score `ssn` has no weight because it is missing in `df2`, and `street_name` has a negative weight (-7.25) because it differs by a typographical error (letter "k" instead of "c"). The waterfall chart is best used interactively, by rendering Quarto to HTML.

This `pandas` code drops the duplicates, saves the dataset as CSV, and it displays the links:

```
# create new variable 'dup', then drop duplicates, and count
results.shape[0]                                                      ①
results2 = results.sort_values(by=['simulant_id_r', 'match_probability'],
    ascending=False, ignore_index=True)
results2['dup'] = results2.duplicated(keep=False, subset=['simulant_id_r']) ②
results2 = results2.drop_duplicates(subset=['simulant_id_r'])         ③
results.to_csv('monograph2024_supplement3_dup.csv', index=False)      ④
results2.shape[0]                                                     ⑤
```

① Links before deduplication
② Create new variable `dup`
③ Drop duplicates in terms of `simulant_id_r`

④ Save dataframe as CSV file
⑤ Links after deduplication

```
646837
```

```
645786
```

Linkage Performance

The test data define a FP as `simulant_id_l` and `simulant_id_r` not matching. There are about 1,000 FP:

```python
results[results.simulant_id_l != results.simulant_id_r].shape[0]    #
# len(results.loc[results.simulant_id_l != results.simulant_id_r])  # same result
```

```
1100
```

Here are the results *after* deduplication but without a clerical review. The code below uses the "stable matching" for deduplication, and it creates a new indicator variable `dup` for easy analysis of the duplicates.

Splink has two known issues:

- One-to-many matching rather than one-to-one matching. The issue with duplicates in record linkage is known in the literature as the "stable marriage" issue, see `Github Splink` discussion issue 1602.
- `Splink` does not work well on correlated variables. First name and sex are correlated.

Therefore, we drop the 2,000+ uncertain observations due to one-to-many matches or with no match on `first_name` and `sex`, which often are correlated. Here is the code and the results:

```python
results2[( (results2.first_name_l != results2.first_name_r) &
    (results2.sex_l != results2.sex_r)) |
    (results2.dup == 1)].shape[0]                                          ①
results3 = results2[~((results2.first_name_l != results2.first_name_r) &
    ( results2.sex_l != results2.sex_r ) |
    (results2.dup == 1))]
results3.shape[0]                                                          ②
results3[results3.simulant_id_l != results3.simulant_id_r].shape[0]        ③
```

```python
results3[results3.simulant_id_l == results3.simulant_id_r].shape[0]           ④
results3.to_csv('monograph2024_supplement3__dup2.csv', index=False)           ⑤
```

① Uncertain observations
② Links after dropping uncertain observations
③ FP after dropping uncertain observations
④ TP after dropping uncertain observations
⑤ CSV file after dropping uncertain obs

```
2340
```

```
643446
```

```
15
```

```
643431
```

Sensitivity Analysis

The analysis without SSN found that a reasonable a threshold match probability for matches is 0.999999999 (9 decimals) because the links drop drastically when we increase from 9 to 10 decimals. The code to produce the links, FP and TP by increasing the threshold match probability from 0.95 to 0.999999999 (9 decimals):

```python
results3[(results3.match_probability >= 0.999999999)].shape[0]
results3[(results3.match_probability >= 0.999999999) &
    (results3.simulant_id_l != results3.simulant_id_r)].shape[0]
results3[(results3.match_probability >= 0.999999999) &
    (results3.simulant_id_l == results3.simulant_id_r)].shape[0]  #
```

```
627453
```

```
3
```

```
627450
```

Table 1 shows the resulting confusion table.

Table 1: Confusion Table with 99.9999999% match probability (N=627,319)

	True Matches	**True Non-Matches**	**Total**
Links	627,315 TP	4 FP	627,319
Non-Links	32,912 FN	9,983 TN	42,895
Total	660,227	9,987	670,214

Note: Match probability is >= 0.999999999 (9 decimals). Only one-to-one matches (no duplicates), and no non-match on both first name and sex. Expected manual review is <= 500 records.

See Table 2 for the confusion table when `ssn` is not available as a linkage variable (and see the section Reproducibility, footnote 1, for how to reproduce the report):

Table 2: Confusion Table with 99.9999999% match probability. Same configuration as Table 1 but without SSN as a linkage variable (N=578,729)

	True Matches	**True Non-Matches**	**Total**
Links	578,726 TP	3 FP	578,729
Non-Links	81,301 FN	9,984 TN	91,485
Total	660,227	9,987	670,214

Note: Match probability is >= 0.999999999 (9 decimals). Only one-to-one matches (no duplicates). Expected manual review is <= 500 records.

Table 3 lists the 4 FP in Table 1:

Table 3: List the 4 FP when SSN is available as a linkage variable

simulant_id_l	**simulant_id_r**	**first_name_l**	**first_name_r**	**ssn_l**	**ssn_r**
7745_206735	7745_206733	CAMERON	WILLIAM	087101610	087101610
4400_278663	4400_278664	JORDAN	JOHN	655243080	655243080
40_28686	40_28685	ANGELA	ANGIE		575867836
1219_351363	1219_351364	JOSHUA	VERONICA	709517582	709517582

Note: Match probability is >= 0.999999999 (9 decimals). Only one-to-one matches (no duplicates), and no non-match on both first name and sex. Expected manual review is <= 500 records.

For the test data, the match probability 0.999999999 (9 decimals of 9s) is a reasonable default cutoff when the goal is to automatically reduce FP, not necessarily all FP. Because of computer precision, it can be better to express match probability as a percentage, that is 99.9999999% rather than 0.999999999.

The focus of the manual review on these test data should be on first name because the FP are mostly in terms of first name. However, it is unrealistic to expect 0 FP in large record linkages, especially when SSN is not available. On the test data, there are 40,152 record pairs with a match probability 99.9999999% that still differ on first name. The hard unresolved issue is how to deal with FP when SSN is not available.

Recommendation

`Splink` has a results variable for Match Probability, `match_probability`. The main concern of the FCDS is no expected FP. With the original match probability threshold of 0.95, the result was 19 FP when `ssn` is available. To reduce FP, we dropped 2,405 observations that were either one-to-many matches or non-matches on both first name and sex. The result was 19 FP. Therefore, we require a match probability of at least 99.9999999%. The result was 4 FP. There were only 19 observations with match probability less than 0.99. Going forward, we recommend the default `threshold_match_probability` of 0.99 (not 0.95) to determine definite non-matches.

`Splink` contains a variety of tools to help visualising predictions such as the match weight chart and the waterfall chart. `Splink` also has functions to perform more formal accuracy analysis such as the threshold selection tool but these functions require the actual (true) match status such our `pseudopeople` test data.

Overall, the `Splink` test results are much better than `Match*Pro` and better than `fastLink`. However, the FCDS does not have a template for using `Splink`. The next step is to get a template for using `Splink`.

! Important – Planned improvements for fall 2024

Update to Splink 4, which is expected in fall 2024.

S4 Match*Pro Results on the pseudopeople Test Data

This fourth part of the supplement, S4, describes the Match*Pro results on the pseudopeople test data. Match*Pro is a Java-based proprietary software provided by the National Cancer Institute (NCI), and it is developed by Information Management Services (IMS) for the Virtual Pooled Registry-Cancer Linkage System (VPR-CLS). Match*Pro has a graphical user interface (GUI). All Match*Pro user settings are in the Match*Pro linkage configuration fi le (*.mplc). Therefore, screenshots from the Match*Pro linkage configuration fi le, ra ther th an co de, will be provided for the four basic linkage steps.

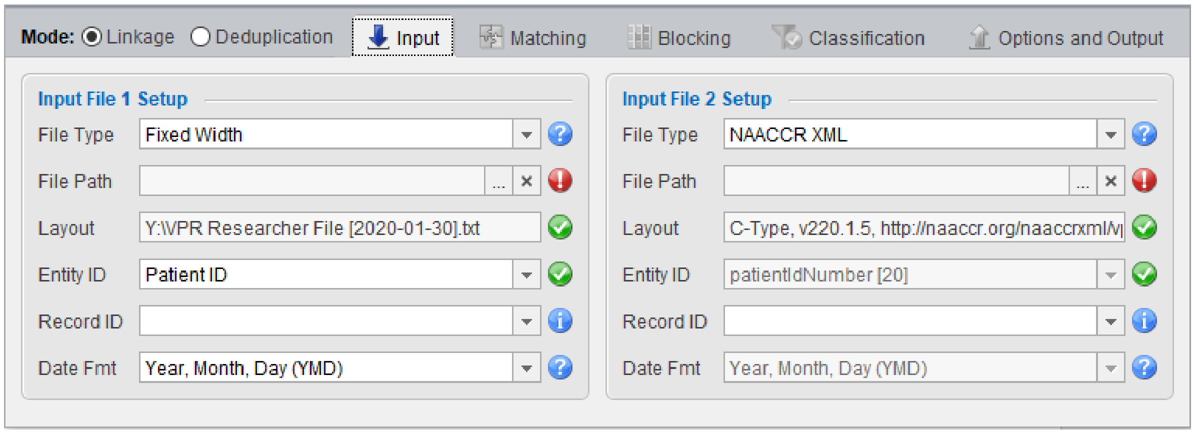
The results are similar when using Social Security Number (SSN) or not. The larger data frame 1 has 1% transposition noise in SSN. In a previous version without this noise, the results were highly dependent on using SSN. There were 436,494 correct matches (true positives, TP), 4 wrong matches (false positives, FP), 223,733 or 33.9% missed matches (false negatives, FN), and 9,983 correct non-matches (true negatives, TN) on the test data if using SSN; see Table 1. There were 442,338 TP, 3 FP, 217,889 or 33.0% FN, and 9,984 TN on the test data if not using SSN; see Table 2. These results assume that all "Matches with Total Score > 40" are matches after a manual review.

In contrast, the default settings without this assumption result in at least 123 FP and require a clerical review of up to 65,566 records; see Tables 3-5. Those results are unaccptable to the FCDS. Therefore, the main text will only report Tables 1 and 2.

It would greatly help if Match*Pro could provide Match Probability as a results variable for more comparable results; see Table 6. In principle, a Match Probability variable would be easy to implement for the optional EM algorithm but hard to implement for the default EpiLink-like frequency-based algorithm.

Step 1: Attribute Alignment

Figure 1 below is a screenshot of the "Input" tab in the default Match*Pro linkage configuration file.



Input File 1 Setup

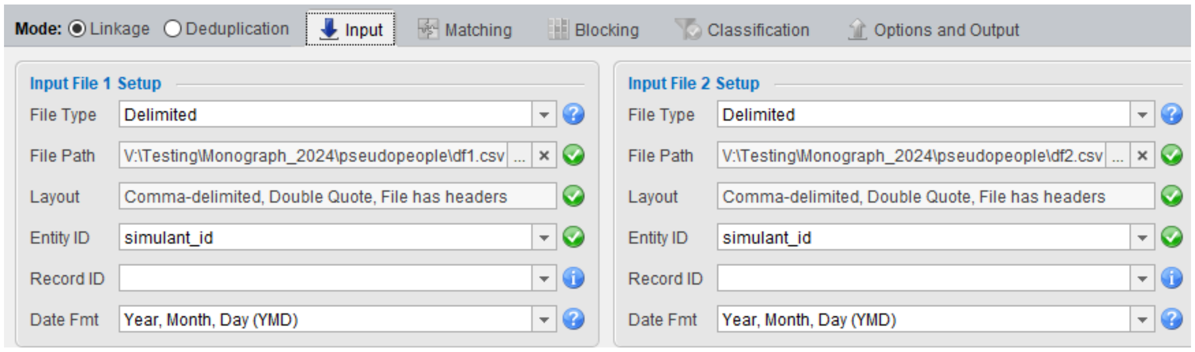
File Type	Fixed Width
File Path	
Layout	Y:\VPR Researcher File [2020-01-30].txt
Entity ID	Patient ID
Record ID	
Date Fmt	Year, Month, Day (YMD)

Input File 2 Setup

File Type	NAACCR XML
File Path	
Layout	C-Type, v220.1.5, http://naaccr.org/naaccrxml/v
Entity ID	patientIdNumber [20]
Record ID	
Date Fmt	Year, Month, Day (YMD)

Figure 1: Match*Pro default for linkage input

The compatible file types for `Match*Pro` input are fixed width, delimited and NAACCR XML. The "NAACCR XML" file type is the NAACCR XML data exchange standard. Figure 2 below is a screenshot of the "Input" tab in the `Match*Pro` linkage configuration file for the `pseudopeople` test data.



Input File 1 Setup

File Type	Delimited
File Path	V:\Testing\Monograph_2024\pseudopeople\df1.csv
Layout	Comma-delimited, Double Quote, File has headers
Entity ID	simulant_id
Record ID	
Date Fmt	Year, Month, Day (YMD)

Input File 2 Setup

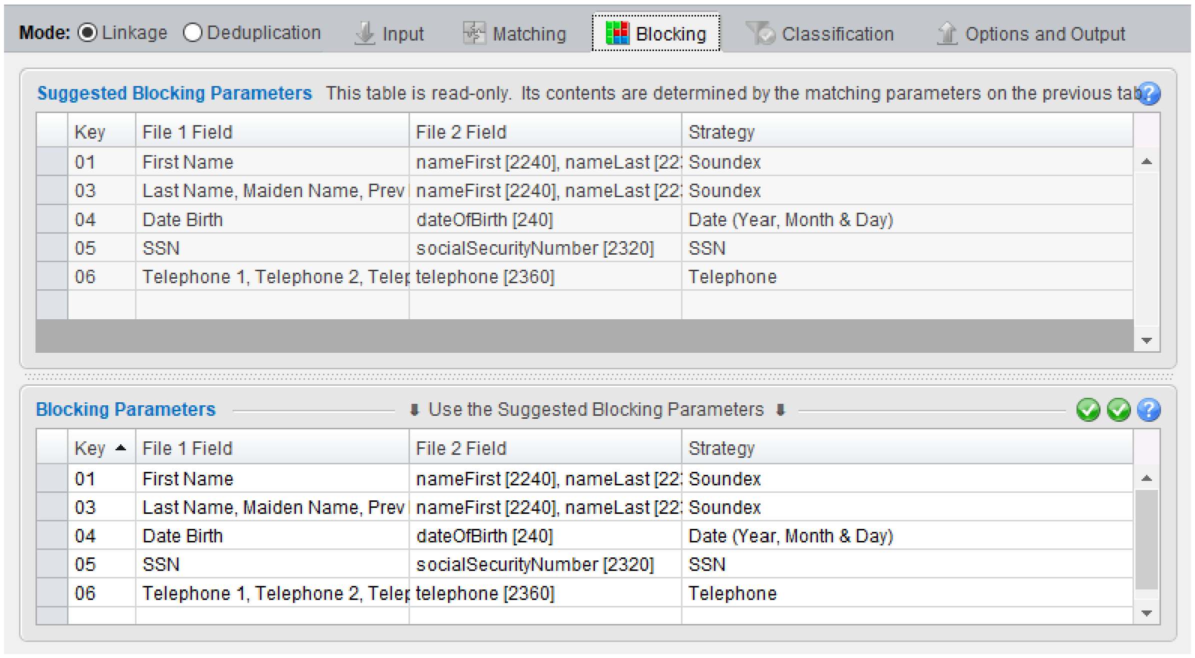
File Type	Delimited
File Path	V:\Testing\Monograph_2024\pseudopeople\df2.csv
Layout	Comma-delimited, Double Quote, File has headers
Entity ID	simulant_id
Record ID	
Date Fmt	Year, Month, Day (YMD)

Figure 2: Match*Pro linkage input configuration for the pseudopeople test data

The "Input" is easy to configure compared with having to write code. For example, for the "Entity ID", the default is `Patient ID` (see Figure 1), whereas it is `simulant_id` in the `pseudopeople` data (see Figure 2). However, data cleaning is limited. For example, the user can rename columns but otherwise cannot standardize columns.

Step 2: Blocking

It would help to better distinguish between blocking and record linkage if blocking was optional and if you could save the blocked data. `Match*Pro` displays the "Blocking" tab after the "Matching" tab despite that blocking is typically done before matching. See Figure 3.


Mode: [✓ Linkage] Deduplication | Input | Matching | Blocking | Classification | Options and Output

Suggested Blocking Parameters This table is read-only. Its contents are determined by the matching parameters on the previous tab

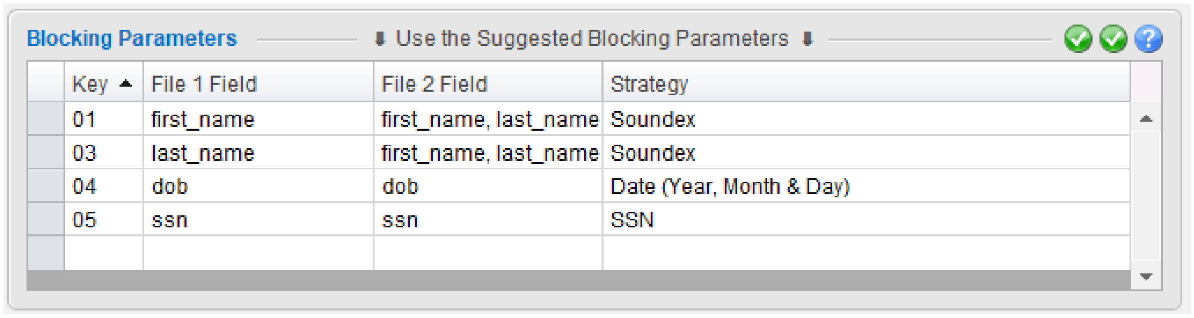
Key	File 1 Field	File 2 Field	Strategy
01	First Name	nameFirst [2240], nameLast [22	Soundex
03	Last Name, Maiden Name, Prev	nameFirst [2240], nameLast [22	Soundex
04	Date Birth	dateOfBirth [240]	Date (Year, Month & Day)
05	SSN	socialSecurityNumber [2320]	SSN
06	Telephone 1, Telephone 2, Telep	telephone [2360]	Telephone

Blocking Parameters — Use the Suggested Blocking Parameters

Key	File 1 Field	File 2 Field	Strategy
01	First Name	nameFirst [2240], nameLast [22	Soundex
03	Last Name, Maiden Name, Prev	nameFirst [2240], nameLast [22	Soundex
04	Date Birth	dateOfBirth [240]	Date (Year, Month & Day)
05	SSN	socialSecurityNumber [2320]	SSN
06	Telephone 1, Telephone 2, Telep	telephone [2360]	Telephone

Figure 3: Match*Pro default for blocking

Figure 4 below is the equivalent screenshot for blocking on the test data.

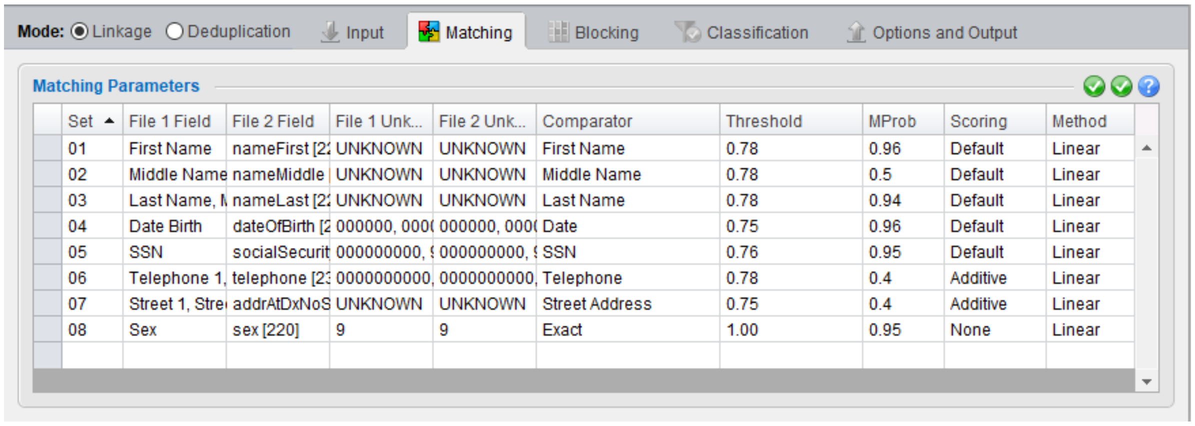

Blocking Parameters — Use the Suggested Blocking Parameters

Key	File 1 Field	File 2 Field	Strategy
01	first_name	first_name, last_name	Soundex
03	last_name	first_name, last_name	Soundex
04	dob	dob	Date (Year, Month & Day)
05	ssn	ssn	SSN

Figure 4: Match*Pro blocking configuration for the pseudopeople test data

Step 3: Record Linkage

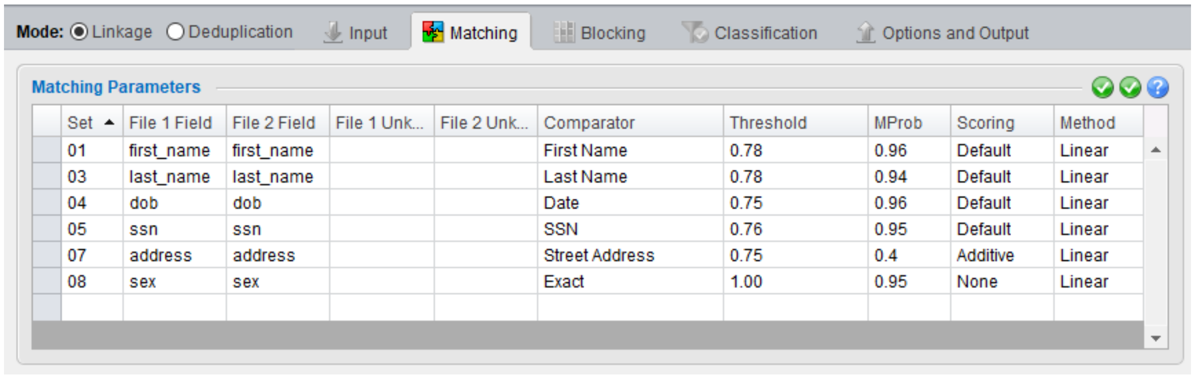
`Match*Pro` by default uses the matching parameters in Figure 5.



Set	File 1 Field	File 2 Field	File 1 Unk...	File 2 Unk...	Comparator	Threshold	MProb	Scoring	Method
01	First Name	nameFirst [22	UNKNOWN	UNKNOWN	First Name	0.78	0.96	Default	Linear
02	Middle Name	nameMiddle	UNKNOWN	UNKNOWN	Middle Name	0.78	0.5	Default	Linear
03	Last Name, M	nameLast [22	UNKNOWN	UNKNOWN	Last Name	0.78	0.94	Default	Linear
04	Date Birth	dateOfBirth [2	000000, 000(	000000, 000(	Date	0.75	0.96	Default	Linear
05	SSN	socialSecurit	000000000, 9	000000000, 9	SSN	0.76	0.95	Default	Linear
06	Telephone 1,	telephone [23	0000000000,	0000000000,	Telephone	0.78	0.4	Additive	Linear
07	Street 1, Stre	addrAtDxNoS	UNKNOWN	UNKNOWN	Street Address	0.75	0.4	Additive	Linear
08	Sex	sex [220]	9	9	Exact	1.00	0.95	None	Linear

Figure 5: Match*Pro default for linkage

`Match*Pro` uses the matching parameters in Figure 6 on the test data.



Set	File 1 Field	File 2 Field	File 1 Unk...	File 2 Unk...	Comparator	Threshold	MProb	Scoring	Method
01	first_name	first_name			First Name	0.78	0.96	Default	Linear
03	last_name	last_name			Last Name	0.78	0.94	Default	Linear
04	dob	dob			Date	0.75	0.96	Default	Linear
05	ssn	ssn			SSN	0.76	0.95	Default	Linear
07	address	address			Street Address	0.75	0.4	Additive	Linear
08	sex	sex			Exact	1.00	0.95	None	Linear

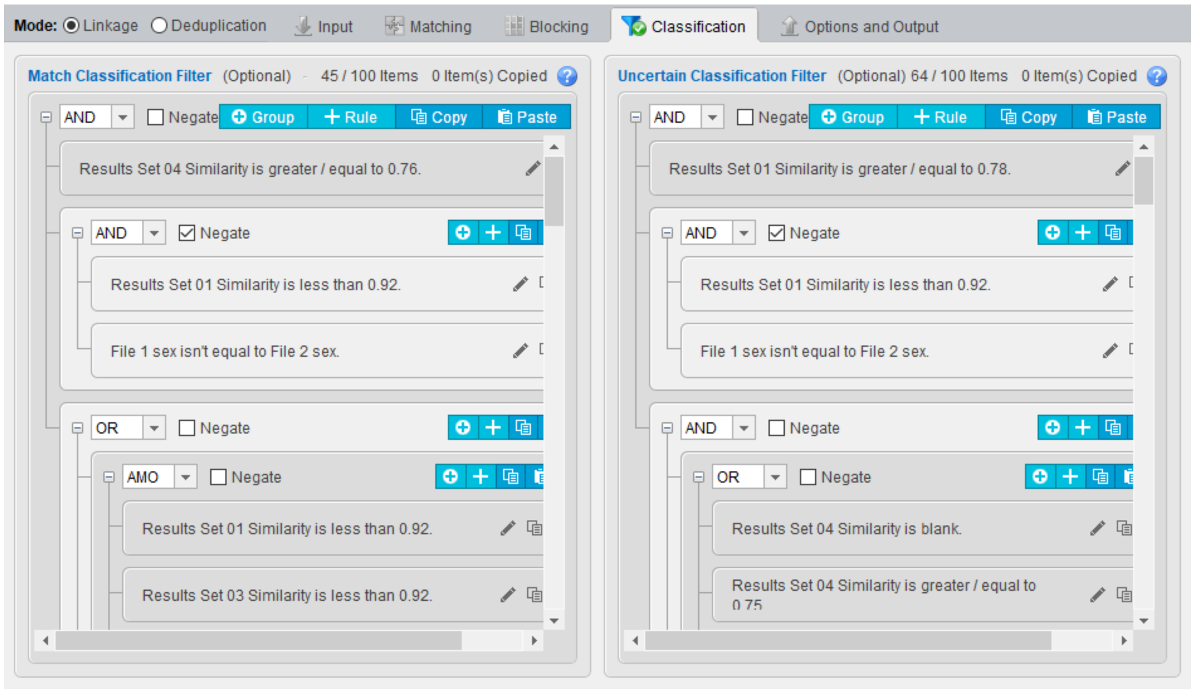
Figure 6: Match*Pro matching configuration on the test data

The column "MProb" in Figure 6, or "m probability", is the pre-determined match probability for each linkage variable. The scores from each set are combined to obtain the Total Score. The scoring values are "Default", "None", and "Additive". The value "Default" always affects Total Score. The value "None" never affects Total Score. The value "Additive" means that the score for a matching parameter will only affect Total Score if it is greater than or equal to zero. The "Linear" scoring method, which is the default, gives partial weight for partial matches.

`Match*Pro` by default uses a frequency-based `EpiLink`-like (Contiero et al. 2005) algorithm for calculating Total Score. `Link Plus` with the Direct Method, the R package `RecordLinkage` with the function epiWeights and the free and open-source (FOSS) Java software `Mainzelliste` are three other record linkage software which also use a frequency-based `EpiLink`-like algorithm.

By default, `Match*Pro` uses a complicated set of 91 deterministic rules for classification (also known as “prediction”), out of which 38 rules classify “Match”, and 53 rules classify “Uncertain”. Remaining linked pairs are classified as “ Non-Match”. We simplified classification by not using “Middle Name” and “Telephone”. The number of rules changed from 91 to 72, with 29 rules in the Match Classification Filter and 43 rules In the Uncertain Classification Filter. The equivalent of Total Score in `fastLink` and `Splink` is Match Weight. `fastLink` and `Splink` rely completely on Match Weight and on a derived “Match Probability” for classification. However, `Match*Pro` does not use Total Score or a derived “Match Probability” for classification. `Match*Pro` does not even have a column or variable for “Match Probability”. `Match*Pro` uses deterministic classification to ensure definite non-match for certain linkage combinations such as complete mismatch on first name. The alternative is to filter out invalid linkage combinations in pre- or post-processing. Regardless, it is critical to lower the Total Score (and Match Probability) appropriately so that Total Score always is higher for “Match” than for “Uncertain” than for “Non-Match”. Using Total Score for classification would be very simple to implement: “`totalscore_match > totalscore_uncertain > totalscore_nonmatch`”. Arguably, using Total Score for classification would also make classification easier and safer to use, understand, and explain.

Because the `Match*Pro` classification rules are so complicated, a single screenshot would result in too small text to read. Therefore, Figure 7 below only shows the first 5 classification rules for each filter.



Match Classification Filter (Optional) - 45 / 100 Items 0 Item(s) Copied

- AND ☐ Negate — Group + Rule Copy Paste
  - Results Set 04 Similarity is greater / equal to 0.76.
  - AND ☑ Negate
    - Results Set 01 Similarity is less than 0.92.
    - File 1 sex isn't equal to File 2 sex.
  - OR ☐ Negate
    - AMO ☐ Negate
      - Results Set 01 Similarity is less than 0.92.
      - Results Set 03 Similarity is less than 0.92.

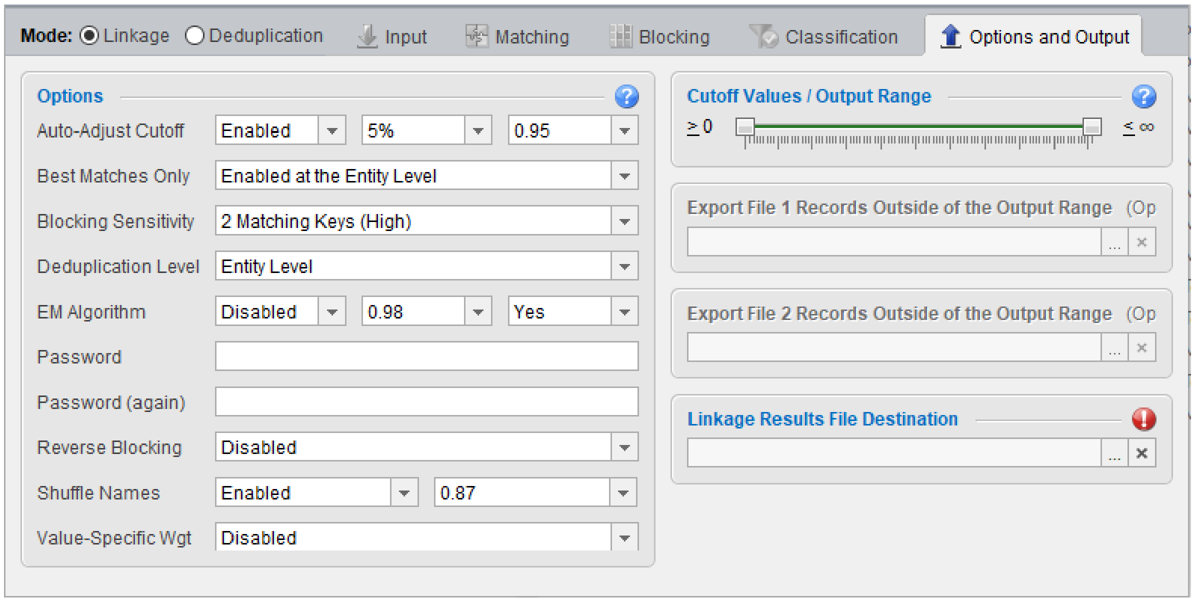
Uncertain Classification Filter (Optional) 64 / 100 Items 0 Item(s) Copied

- AND ☐ Negate — Group + Rule Copy Paste
  - Results Set 01 Similarity is greater / equal to 0.78.
  - AND ☑ Negate
    - Results Set 01 Similarity is less than 0.92.
    - File 1 sex isn't equal to File 2 sex.
  - AND ☐ Negate
    - OR ☐ Negate
      - Results Set 04 Similarity is blank.
      - Results Set 04 Similarity is greater / equal to 0.75

Figure 7: Match*Pro classification on the test data – first 5 rules

Step 4: Canonicalization

Step 4, canonicalization, is the post-processing step. The aim is to get unique and representative, that is, "canonical", records. Figure 8 below is a screenshot of the "Options and Output" tab in the default `Match*Pro` linkage configuration file.



Mode: [✓ Linkage] Deduplication

Input | Matching | Blocking | Classification | Options and Output

Options

Auto-Adjust Cutoff	Enabled	5%	0.95
Best Matches Only	Enabled at the Entity Level		
Blocking Sensitivity	2 Matching Keys (High)		
Deduplication Level	Entity Level		
EM Algorithm	Disabled	0.98	Yes
Password			
Password (again)			
Reverse Blocking	Disabled		
Shuffle Names	Enabled	0.87	
Value-Specific Wgt	Disabled		

Cutoff Values / Output Range

≥ 0 ≤ ∞

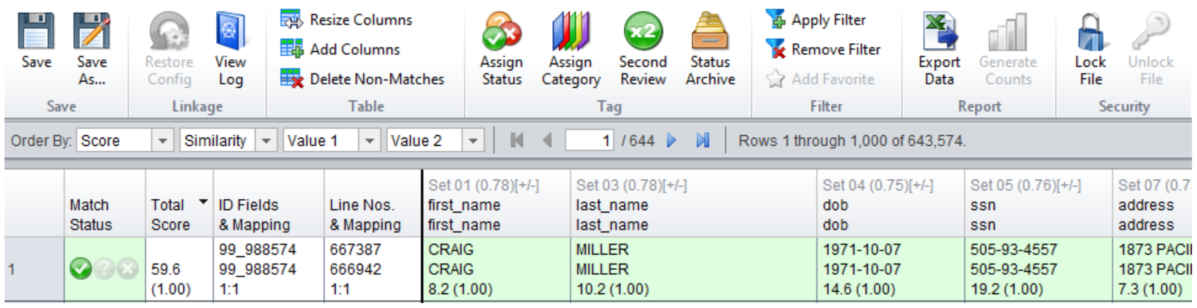
Export File 1 Records Outside of the Output Range (Op

Export File 2 Records Outside of the Output Range (Op

Linkage Results File Destination

Figure 8: Match*Pro default for options and output

The setting "Auto-Adjust Cutoff 5%" means that 5% of the records in the smaller file are expected to be matches. The option "Auto-Adjust Cutoff 0.95" is, according to the User's Guide, the "desired PPV (i.e., the desired probability that two records are NOT matched together by chance)". How does this "5% expectation" and "0.95 probability" affect Total Score? It is undocumented in `Match*Pro`. The setting "EM Algorithm Disabled" means that `Match*Pro` uses a frequency-based `EpiLink`-like algorithm. The `Match*Pro` linkage log and results can be saved separately in a "txt" text file file and in an "mplr" `Match*Pro` file. In the linkage results screen, the first column is Match Status with values "v" for "Match", "?" for "Uncertain" and "x" for "Non-Match". The second column is Total Score with the values for Total Score and, below, "Overall Similarity". The Overall Similarity can be misleading because it is not a Match Probability despite being on the 0-1 scale. In Figure 9 below, from a previous run with no transposition noise in SSN, the record is classified as "Match" with Total Score 59.6 and Overall Similarity 1.00.



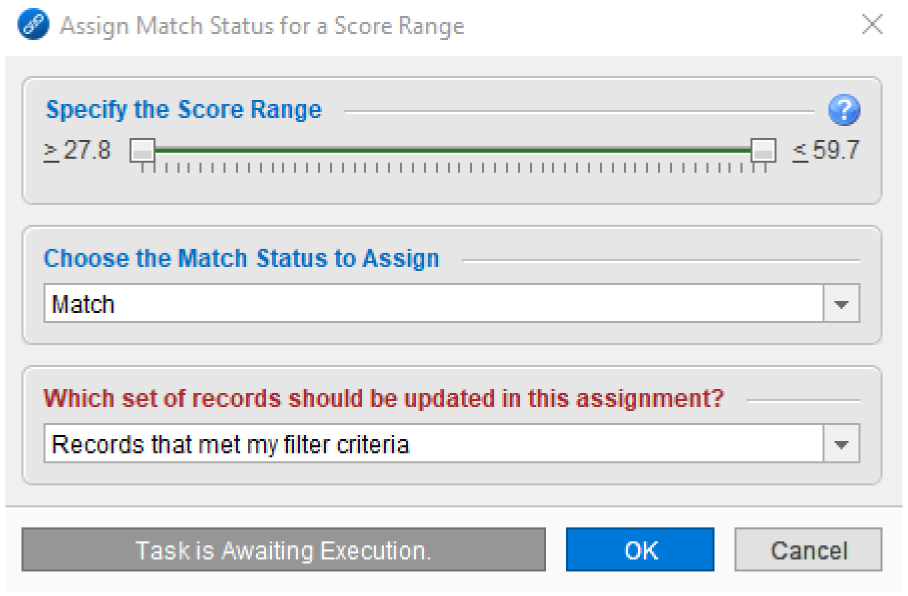
Order By: Score | Similarity | Value 1 | Value 2 | 1 / 644 | Rows 1 through 1,000 of 643,574.

	Match Status	Total Score	ID Fields & Mapping	Line Nos. & Mapping	Set 01 (0.78)[+/-] first_name first_name	Set 03 (0.78)[+/-] last_name last_name	Set 04 (0.75)[+/-] dob dob	Set 05 (0.76)[+/-] ssn ssn	Set 07 (0.7 address address
1	✓	59.6 (1.00)	99_988574 99_988574 1:1	667387 666942 1:1	CRAIG CRAIG 8.2 (1.00)	MILLER MILLER 10.2 (1.00)	1971-10-07 1971-10-07 14.6 (1.00)	505-93-4557 505-93-4557 19.2 (1.00)	1873 PACII 1873 PACII 7.3 (1.00)

Figure 9: Match*Pro results – first 1 record

Overall similarity is sometimes known as "fuzzy matching". The "Total Score" improves on "Overall Similarity" by accounting for the frequencies in the overall dataset (which is sometimes known as "term frequencies") or more broadly measuring how common different scenarios are. However, neither Overall Similarity nor Total Score measures the relative importance of non-matches. It would require `Match*Pro` to estimate the m probabilities based on the data. Instead, `Match*Pro` by default (that is, when EM is disabled), as shown previously in Figure 6, uses pre-determined and fixed m probabilities.

Clicking on the menu icon "Assign Status" brings up the Assign Match Status dialog window as in Figure 10. See the `Match*Pro` User's Guide, page 141, for details. The dialog window allows you to modify Match Status based on Total Score.


Assign Match Status for a Score Range
Task is Awaiting Execution.
OK
Cancel

Specify the Score Range: ≥ 27.8 ≤ 59.7

Choose the Match Status to Assign: Match

Which set of records should be updated in this assignment? Records that met my filter criteria

Figure 10: Match*Pro Assign Match Status dialog window for test data

Results

The `Match*Pro` results are more subjective than necessary due to `Match*Pro` not providing match probability. It is standard to present record linkage results in the form of a confusion table (also known as confusion matrix). The basic measures are TP, FP, FN, and TN. The confusion table consists of number counts. For FP, we generally are interested in the number count. However, for FN we generally are interested in the percentage rate for easier comparisons. Tables 1-5 below illustrate three different scenarios: optimistic, neutral, and conservative. The number of predicted matches differs approximately 65,000 depending on the accepted match probability; see Table 6 below for a summary table.

In the conservative scenario, if only all matched records with Total Score 40 are matches after the manual review, there are 436,498 predicted matches. The manual review would be up to 1,000 "Match" records with the lowest Total Score > 40. See Table 1 below. Similar results are achieved if the match threshold is increased from the default 0.95 to the highest setting 0.99 (see Figure 8); see the sensitivity analysis section (especially Tables 4, 6 and 7) for details.

Table 1: Confusion Table if "predicted matches" include all "Match" records with Total Score > 40 (N=436,498)

	True Matches	**True Non-Matches**	**Total**
Links	436,494 TP	4 FP	436,498
Non-Links	223,733 FN	9,983 TN	233,716
Total	660,227	9,987	670,214

Note: Expected manual review is <= 1,000 records ("Match" with lowest Total Score > 40). The FN rate is 33.9% (that is, 100 * 223,733 / 660,227).

Table 2 below is the same conservative scenario as Table 1 but without using SSN.

Table 2: Confusion Table if "predicted matches" include all "Match" records with Total Score > 40. Same configuration as Table 1 but without SSN (N=442,341)

	True Matches	**True Non-Matches**	**Total**
Links	442,338 TP	3 FP	442,341
Non-Links	217,889 FN	9,984 TN	227,873
Total	660,227	9,987	670,214

Note: Expected manual review is <= 1,000 records ("Match" with lowest Total Score > 40). The FN rate is 33.0% (that is, 100 * 217,889 / 660,227).

Table 2 without `ssn` provides slightly better results than Table 1. In the conservative scenario, `Match*Pro` is unable to take advantage of `ssn` in Table 1. One possible reason is that maybe the "SSN" comparator performs worse than the Damerau-Levenshtein comparator which is available but not used by default.

Sensitivity Analysis

In the *optimistic* scenario, if SSN is used and if all "Uncertain" records are matches after the manual review, there are 502,054 predicted matches. The manual review would be up to all 65,566 "Match" or "Uncertain" records with Total Score <= 40. See Table 3 below.

Table 3: Confusion Table if "predicted matches" include all "Uncertain" records (N= 502,054)

	True Matches	**True Non-Matches**	**Total**
Links	501,927 TP	127 FP	502,054
Non-Links	158,300 FN	9,860 TN	168,160
Total	660,227	9,987	670,214

Note: Expected manual review is <= 65,566 records ("Total Score <=40").

In the *neutral* scenario, if SSN is used and if all "Match" records are matches after the manual review, there are 501,866 predicted matches. The manual review would be up to all 65,368 "Match" records with Total Score <= 40. See Table 4 below.

Table 4: Confusion Table if "predicted matches" include all "Match" records (N=501,866)

	True Matches	**True Non-Matches**	**Total**
Links	501,743 TP	123 FP	501,866
Non-Links	158,484 FN	9,864 TN	168,348
Total	660,227	9,987	670,214

Note: Expected manual review is <= 65,368 records ("Match" with Total Score <= 40).

Table 5 is the same neutral scenario as Table 4 except that SSN is not used.

Table 5: Confusion Table if "predicted matches" include all "Match" records. Same configuration as Table 4 but without SSN (N=502,304)

	True Matches	**True Non-Matches**	**Total**
Links	501,934 TP	370 FP	502,304
Non-Links	158,293 FN	9,617 TN	167,910
Total	660,227	9,987	670,214

Note: Expected manual review is <= 59,963 records ("Match" with Total Score <= 40).

Table 1 using SSN and Table 2 not using SSN are the main results. Tables 1-5 rely on Total Score. The average Total Scores, when rounded to one decimal, are the following for each match status: "Match" = 41.0, "Uncertain" = 32.6 , and "Non-Match" 32.1.[1]

Ideally, we also want a results variable "Match Probability". An approximate `Match*Pro` match probability can be calculated by scaling the range of Total Score. This is known as "probability calibration". The Auto-Adjust Cutoff of 0.95 and the minimal Total Score 27.9 presumably correspond to match probability 0.95, because the `Match*Pro` threshold is 0.95 (see Figure 8), and the maximum Total Score 41.9 presumably corresponds to match probability 1.0. If this assumption holds, then a 2.8 average difference in Total Score corresponds to a 0.01 (1%) difference in match probability, as shown in Table 6.

Table 6: Matches based on Total Score and hypothetical Match Probability

Score	**Probability**	**Matches**
27.9 (min)	0.95	504,769
30.7	0.96	493,329
33.5	0.97	486,238
36.3	0.98	475,648
39.1	0.99	442,052
41.9 (max)	1.0	420,534

Note: The Total Score range is 27.9-41.9. The corresponding hypothetical Match Probability range is 0.95-1.0. A 2.8 average difference in Total Score corresponds to a 0.01 (1%) difference in hypothetical Match Probability.

[1] In the previous version without noise in `ssn`, the averages were higher: "Match" = 55.0, "Uncertain" = 38.7 , and "Non-Match" 34.9. The maximum Total Score was 59.6. It is beyond the scope here to further discuss preliminary results from the previous version. The point here simply is that, as expected, the average Total Scores are higher when there is less noise and they are lower when there is more noise.

Table 6 with the hypothetical match probability 0.99 and 442,052 matches approximately correspond to Table 2 with "Total Score $>$ 40" and 442,341 matches.

Table 7 below uses the "Auto-Adjust Cutoff" 0.99 for the "desired PPV" (see Figure 8) of 0.99, which is the largest cutoff possible in `Match*Pro`. However, the actual result 493,587 links is closer to the calibrated 493,329 links and match probability 0.96 than to the calibrated 442,052 links and match probability 0.99.

Table 7: Confusion Table if "predicted matches" include all "Match" records. Same configuration as Table 4 but with the "Auto-Adjust Cutoff" 0.99 (N=493,587)

	True Matches	**True Non-Matches**	**Total**
Links	493,534 TP	53 FP	493,587
Non-Links	166,693 FN	9,934 TN	176,627
Total	660,227	9,987	670,214

Note: Expected manual review is <= 55,225 records ("Match" with Total Score <= 40).

We also ran `Match*Pro` with the EM algorithm enabled. The results were meaningless to the FCDS, and unstable. We tried three EM settings:

1) The equivalent of Table 1 but with "EM Enabled" resulted in this error message: "The linkage session failed. Encountered an exception while classifying linked pairs. View the log for details." The log provided this error message: "org.h2.jdbc.JdbcSQLNonTransientException: General error:"java.lang.IllegalStateException: File corrupted in chunk 6629, expected page length 4..384, got -1725528077 [1.4.200/6]". It is a well-known problem with the H2 database which comes with `Match*Pro` in the `lib` folder, for example see H2 database issue 2139. The H2 release 2.2.222 from 8/22/2023 claims to fix this"file corrupted in chunk" issue.

2) Emptying the "Classification" tab resulted in max Total Score 35.3, which is below the acceptable threshold 40.

3) Not using `ssn` (and keeping the classification rules) resulted in all observations having Total Score in the range 136.3-164.2. The log file provided these linkage quality measures: 541,561 TP, and 0 FP, and 128,589 TN and 64 FN. These EM results are superior both in terms of Total Scores and the confusion table. Table 8 is the same conservative scenario as Table 2 except that EM is used.

Table 8: Confusion Table if "predicted matches" include all "Match" records with Total Score > 40. Same configuration as Table 2 (that is, without SSN) but EM is used (N=515,590)

	True Matches	**True Non-Matches**	**Total**
Links	515,148 TP	442 FP	515,590
Non-Links	145,079 FN	9,545 TN	154,624
Total	660,227	9,987	670,214

Note: Expected manual review is <= 1,000 records ("Match" with Total Score <= 40). EM is used. Warning: These EM results are not valid because the non-EM and EM results use different scales for Total Score.

In Table 8 with EM enabled, the minimum Total Score is 136.3. In Table 2 with EM disabled, the maximum Total Score is 41. How do we explain the superior EM results? There are only 433,570 links with the highest Total Score 164.2 which is very similar to the non-EM results in Table 2. The non-EM and EM results seem to use different scales for Total Score. IMS should document this and, more importantly, provide a more comparable results variable for match probability.

`Match*Pro` feature request to add Match Probability

`Match*Pro` does not have a results variable for Match Probability. It is needed for more comparable results because most linkage quality measures in the record linkage literature are based on match probability. For example, the confusion table measures, and derived measures such as Positive Predictive Value (PPV, also known as precision) and True Positive Rate (also known as recall and sensitivity), are based on match probability. A `Match*Pro` user could do a min-max scaling of Total Score, or something similar, to create an approximate match probability. Table 6 gives a hypothetical example based on the claimed "linear" (see Figure 5) scoring method. Completely different results than Table 1 are possible with different assumptions of needed manual review and match probability, as shown in Tables 2-6. The lack of a variable Match Probability (or similar) in `Match*Pro` is a serious issue in terms of less transparency, comparability, and reproducibility.
The `Match*Pro` log file for the EM results states: "A true assessment of the linkage quality can only be obtained after a thorough manual review of the linkage results followed by an independent, blind comparison with a reference (gold) standard." We agree. Nevertheless, match probability is an essential concept of probabilistic record linkage – too important to be omitted as a results variable. The results variable match weight should also be provided since it is used to calculate the match probability.

A more critical view is that EpiLink-like frequency-based approaches such as `Match*Pro` (default, non-EM) simply cannot appropriately weight matches as match probabilities. Without mentioning `Match*Pro`, this is the view taken by Robin Linacre, the lead developer of `Splink`, in a blog post titled "Why Probabilistic Linkage is More Accurate than Fuzzy Matching For Data Deduplication".

For examples of FP, Table 9 below lists the 4 FP in Table 1. The first row is a FP because of missing `first_name`, different `sex`, and different `dob`. Rows 2-4 are FP because of different `dob`.

Table 9: List of the 4 FP in Table 1

Total Score	**Predicted**	**Actual**	**File1id**	**File2id**
40.4	Match	Non-match	1235_670643	1235_670644
41.9	Match	Non-match	5348_901917	5348_901918
41.9	Match	Non-match	5594_237736	5594_237734
41.9	Match	Non-match	9272_30612	9272_30613

Note: Predicted match is defined as "Match with Total Score > 40". We use this conservative approach because `Match*Pro` has no results variable Match Probability for more comparable results. Actual non-match is defined as `File1id` not equal to `File2id`.

Recommendation

The FCDS already recommends and uses `Match*Pro` in VPR Phase 1 record linkages where fast and approximate results matter more than slow and accurate results. The `Match*Pro` test results show that the current version 2.4.4 has improved on the previously tested version 1.6.2 mostly by controlling expected FP at the expected expense of more FN. The tests results are acceptable to the FCDS as long as the manual review is feasible. In practice, the expected manual review for `Match*Pro` must be at most approximately 1,000 records or the data request would require a special time and cost approval. The equivalent standard amount of manual review for `fastLink` and `Splink` is approximately at most 500 records.

The author recommends `Match*Pro` if the expected manual review is <1,000 records, and if time and cost (see the FCDS data request fees) or protocol matter more than expected more FN. The main text will present only the `Match*Pro` conservative results of "Matches with Total Score > 40" as in Tables 1-2, unless `Match*Pro` adds a results variable Match Probability for more comparable results.

What's next?

Compare the performance again if a results variable Match Probability becomes available.

References

Alexandersson, Anders. 2024. *Comparing the Linkage Performance of `fastLink`, `Splink`, and `Match*Pro` at the Florida Cancer Registry Using Simulated `pseudopeople` Data.* Florida Cancer Data System, University of Miami, Miami, FL. https://fcds.med.miami.edu/inc/publications.shtml#tabs-3.

Ansolabehere, Stephen, and Eitan Hersh. 2017. "ADGN: An Algorithm for Record Linkage Using Address, Date of Birth, Gender, and Name." *Statistics and Public Policy* 4 (1): 1–10. https://doi.org/10.1080/2330443X.2017.1389620.

Contiero, Paolo, Andrea Tittarelli, G. Tagliabue, A. Maghini, Sabrina Fabiano, P. Crosignani, and R. Tessandori. 2005. "The Epilink Record Linkage Software: Presentation and Results of Linkage Test on Cancer Registry Files." *Methods of Information in Medicine* 44 (1): 66–71. https://doi.org/10.1055/s-0038-1633924.

Harrison, Matt. 2024. *Effective Pandas 2: Opiniated Patterns for Data Manipulation*. Independently published. https://store.metasnake.com/effective-pandas-book.

Kaplan, Adee, Brenda Betancourt, and Rebecca C. Steorts. 2022. "A Practical Approach to Proper Inference with Linked Data." *The American Statistician*, 1–27. https://doi.org/10.1080/00031305.2022.2041482.

Shearer, Michael. 2024. *Hands-On Entity Resolution: A Practical Guide to Data Matching Matching with Python*. Sebastopol, CA: O'Reilly. https://www.oreilly.com/library/view/hands-on-entity-resolution/9781098148478/.