personMatchR: Simplifying matching people across datasets

The NHSBSA holds and has access to numerous data sets, across multiple service areas, where personal information is captured and there are some situations where, subject to Information Governance authorisation, individuals may need to be matched across multiple data sets. Some examples of previous projects include:

- Identifying date of death information from the Personal Demographic Service (PDS) to look for potentially fraudulent activity taking place after an individual's death.
- Identifying activity for a cohort of individuals shared from an external source.

In an ideal world for data analysis, each data set would include consistent unique identifiers (e.g., NHS number, NI number) which would allow the same individual to be identified. However, this is generally not the case and therefore alternative methods need to be considered.

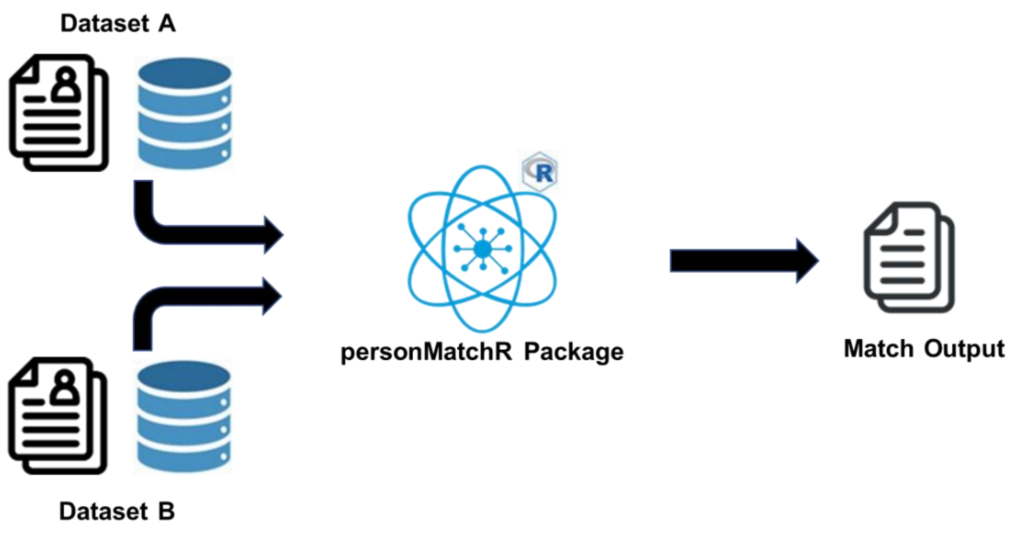
In the past these matching exercises have been handled on a case-by-case basis, which is time consuming and the process that is used for the matching will generally depend on who is developing the scripts.

We wanted to create a process/package that can be reused for any future matching projects including the following benefits:

- Reusable
  - Reduction in time resource as less coding will be required for future projects.
  - Creating a generic approach will allow the process to be used both within DALL and across other teams.
- Consistent
  - Using a single package will allow consistent rules and logic to be applied.
- Efficient
  - By tuning performance up front we can remove bottlenecks in the process.
- Open
  - Using open-source software like R allows this to be reused by anyone.
  - Publishing the code will allow collaboration both internally and externally.

Automating the matching process

Any matching process typically has the same three basic stages: we start with two datasets we want to match individuals across, we perform the matching and handle the output. The first and last stage of this process will generally differ on a case-by-case basis as we could be working with data from many different sources and what we want to do with the output will depend on the project. However, the matching process is typically consistent and therefore this is the stage of the process we are targeting with the personMatchR package.



The latest iteration of the personMatchR identifies both exact and confident matches between datasets based on four key pieces of personal information: the forename, surname, date of birth and postcode. The match output will essentially show which records (individuals) from “Dataset A” could be found in “Dataset B”, and although there are some parameters which can determine what fields are included in the output, it is up to the end user to decide how they then want to go on to use the matching output.

Basic example test case

To show how the personMatchR package can be used we have included some example data within the package repository that can be used as a demo.

First, we need to load the package and the datasets:

```
# Install and load required packages ------------------------------------------------------
devtools::install_github("nhsbsa-data-analytics/personMatchR")
library("dplyr")
library("dbplyr")

# Import datasets for matching ------------------------------------------------------------
df_a <- readRDS(url("https://github.com/nhsbsa-data-analytics/personMatchR/tree/main/R/documentation/TEST_DF_A.rds"))
df_b <- readRDS(url("https://github.com/nhsbsa-data-analytics/personMatchR/tree/main/R/documentation/TEST_DF_B.rds"))

# Review the test datasets ----------------------------------------------------------------
df_a
df_b
```

df_a

ID	SURNAME	FORENAME	POSTCODE	DOB
1	O'Brien	Richard	AA9A 9AA	25/03/1942
2	Bloggs	Joe	A9A 9AA	01/01/1999
3	Watson	David	A9 9AA	01/11/2001
4	Neville	Dylan	A99 9AA	24/05/1941

df_b

ID	SURNAME	FORENAME	POSTCODE	DOB
1	O Brien	Richard	AA9A 9AA	25/03/1942
2	Bloggs	Joseph	A9A 9AA	01/01/1999
3	Brown	John	Z1 1ZZ	01/11/2001
4	Dylan	Neville	A99 9AA	24/05/1941

As we only have four records in each dataset it is easy to manually compare these to see where we would expect to find a match based on the personal information:

- **ID = 1**
  - The only difference between these is some slight formatting on the surname so we would expect these two records to be matched.
- **ID = 2**
  - The only difference is the forename, although these are suitably close that whilst the records are not an exact match we would still hope these records were matched
- **ID = 3**
  - There is no record in df_b that appears to be a suitable match so would not expect any match results
- ID = 4
  - In this example the forename and surname appear to flip between datasets, which will occasionally happen when personal information is being captured, and we would want a matching package to take this into consideration.

Now we know what we may expect to see, we can call the matching function to see what results are produced:

```
# Run the personMatchR package to identify matches ------------------------------------------
df_output <- personMatchR::calc_match_patients(
                              df_one = df_a, # first dataset
                              id_one = ID, # unique id field from first dataset
                              forename_one = FORENAME, # forename field from first dataset
                              surname_one = SURNAME, # surname field from first dataset
                              dob_one = DOB, # date of birth field from first dataset
                              postcode_one = POSTCODE, # postcode field from first dataset
                              df_two = df_b, # second dataset
                              id_two = ID, # unique id field from second dataset
                              forename_two = FORENAME, # forename field from second dataset
                              surname_two = SURNAME, # surname field from second dataset
                              dob_two = DOB, # date of birth field from second dataset
                              postcode_two = POSTCODE, # postcode field from second dataset
                              output_type = "key", # only return the key match results
                              format_data = TRUE, # format input datasets prior to matching
                              inc_no_match = TRUE # return records from first dataset without matches
                              )

# Review the match output ------------------------------------------------------------------
df_output
```

df_output

DF1_INPUT_ID	DF2_INPUT_ID	MATCH_TYPE	MATCH_COUNT	MATCH_SCORE
1	1	Exact	1	1.00
2	2	Confident	1	0.96
3	NA	No Match	0	0.00
4	4	Exact	1	1.00

The match results show that the results are as we would have expected based on the input data, with the output including some fields to provide additional context:

- **MATCH_COUNT**
  - In some datasets a record may find multiple potential matches and this field will show how many matches were found for the record from DF1.
- **MATCH_SCORE**
  - This provides a general weighted score for the match which may help compare instances where multiple potential matches are identified. However, please note that this is for guide purposes only and a higher score will not always mean that the match is more likely to be correct.

Matching function parameters

The example code highlights several parameters that need to be defined when calling the matching function:

- **Required parameters:**
  - **Dataset definitions** (df_one, id_one, forename_one, …, …, postcode_two)
    - These identify the datasets for matching and which fields within those datasets should be used for the matching.
  - **Output formatting** (output_type = “key” / “match” / “all”)
    - This defines what data is to be included in the output results where matches were identified.
    - “Key” will only return the basic fields including only the IDs for the records matched.
    - “Match” will return the key fields, plus the personal information fields used in the matching.
    - “All” will return the key fields, plus all other fields from both datasets where a match was found.
  - **Input formatting** (format_data = TRUE / FALSE)
    - Not used for the database version of the matching function.
    - Identifies if the data should be formatted prior to matching. The formatting will aim to ensure both datasets are consistently formatting to maximise chances of finding matches.
    - Please note, if formatting is not applied prior to matching the results may be limited where differences exist due to data formatting (case, whitespace, special characters etc.).
  - **Include null matches** (inc_no_match = TRUE / FALSE)
    - Identifies if records from the first dataset should be included in the match results where no match was found for the record.

Collecting input data: flat files versus database tables

To accommodate as many use cases as possible, the personMatchR package has different functions available to handle matching whether the input data is supplied as a flat file directly within R or via a connection to database tables (currently only set up and tested for Oracle database infrastructure used by NHSBSA).

Importing flat files, such as .csv files to R may be suitable when the data sets are not too large, and in testing we were able to handle files up to around one million records before any notable performance issues were identified. One advantage to using this method is that the data formatting can be handled within the **calc_match_patients()** function, reducing the amount of code required.

For larger datasets it may not be viable to use flat files due to system memory limitations and this is why the **calc_match_patients_db()** function was created, allowing the data sets to be access direct from database tables and pushing most of the calculation to be handled on the database server side using the **dbplyr** package in R. In testing we were successful in matching using database tables containing over 80 million records with results within a couple of hours. One limitation with the database matching function is that all formatting must have been applied to the data prior to the matching function being called, although the package does include functions that allow this. Combining the formatting within the matching function was producing a considerable overhead in terms of performance and therefore these actions needed to be separated.

Quality and performance based on real world data

With any matching exercise it is difficult to say with 100% certainty if the matches are accurate, without a common unique identifier, as there is always scope that people may have very similar information or data quality issues may be present. To test the accuracy of the personMatchR package we ran a test using electronic prescribing data and PDS information, which both contain the patient's NHS number which could be used to confirm accuracy. In this test case 99.9% of records could be accurately matched between the two datasets.

In addition to test cases being very accurate, when using the personMatchR package to complete a patient matching initiative we identified substantial resource savings, completing the initiative in a fraction of the time this type of work would have taken previously.

What are the next steps for the personMatchR package?

In the spirit of collaboration and development the personMatchR package is something that we expect to continue to develop over time. This could include adding new features or simply refining/improving the process.

As the package is hosted on GitHub any feedback is welcome and this could include suggestions for potential improvements.

There are two areas that have already been flagged for potential development, either as part of the key matching process or as an optional feature:

- Incorporation of "Gender"
  - Gender is often suggested as an additional piece of personal information that could be used in matching, alongside name, date of birth and postcode.
  - Gender has been omitted so far as it is not always captured directly within NHSBSA systems, and as we move to more digital systems, we may find this information is captured less, especially as the traditional male/female classifications may not always be deemed inclusive for people identifying as non-binary.
  - If gender was to be included in the matching process, it may be of more use as a restriction in collaboration with the forename. For example, where forenames were only similar, we may want to ignore a match where the genders were different (e.g. Stephen & Stephanie).

- Incorporation of "name diminutives"
  - Name diminutives refer to occasions where people may go by either abbreviated or alternative versions of a name. For example, somebody called Alison may also go by Ali, Ally, or Allie.
  - Sometimes people may use different versions of their name across different datasets which can make matching difficult.
  - Incorporating name diminutives would allow all possible versions of a name to be considered in the matching process.
  - We did include this in a previous patient matching exercise but found that it only offered limited benefits with only a small number of additional matches being identified, probably because many name diminutives are very similar and would therefore still pass the current match scoring thresholds.
  - The main limitations with including name diminutives are sourcing an appropriate list of name pairs and the potential impact of performance as this could lead to many more comparisons needing to take place.