NHS
Business Services Authority

personMatchR Package
Use Case

Using personMatchR to identify date of death information for medical exemption certificate holders

Introduction

The personMatchR package (available on GitHub) provides functionality to allow two datasets to be compared to identify where individuals can be matched across both datasets, based on the individuals forename, surname, date of birth and postcode.

This document shows how the personMatchR package has been used to solve a real-world problem.

Using personMatchR to identify date of death information for medical exemption certificate holders

A request coming into the NHSBSA DALL team posed the following question:
"If we have a list of medical exemption certificate holders, can we look to see if any of them have been flagged as deceased within the Personal Demographic Service (PDS) data."

Medical exemption certificates provide entitlement to free NHS prescriptions and are available for people who have any of the qualifying medical conditions. A certificate will typically last for a period of five years and where certificates have been issued for long term conditions, reminder letters will normally be sent out to prompt the certificate holder to reapply for a new certificate when the expiration data is approaching. In some instances a reminder letter may be issued to certificate holders who are deceased, which may cause distress to the family and in some cases lead to complaints to the NHSBSA.

The aim of this initiative was to scope the viability of trying to identify cases where the certificate holder could be identified as deceased to prevent a reminder being issued.

This is a similar person matching exercise to those we have handled in the past and a perfect chance to test out the personMatchR package. But before we can run the data through the matching functions, we need to make sure we have prepped the input data first.

Data Preparation: Medical exemption Data

Data was shared for just over 31.5 thousand medical exemption certificate holders, representing certificates expiring in a single month, and a quick review of the data shows that only minimal pre-processing is required:

CERT_NUMBER	SURNAME	FIRST_NAME	DATE_OF_BIRTH	ADDRESS_LINE1	ADDRESS_LINE2	ADDRESS_LINE3	POSTCODE
99999999999	SMITH	JOHN	01-Jan-99	Stella House	Goldcrest Way	Newcastle-Upon-Tyne	NE15 8NY
88888888888	O'Brien	Richard	01-Jan-99	Bridge House	152 Pilgrim St	Newcastle-Upon-Tyne	NE1 6SN

***NB**: data is for illustrative purposes only and does not reflect any real customer information*

The four key pieces of personal information (forename, surname, date of birth and postcode) as all present and the "CERT_NUMBER" field is unique across all records and therefore suitable as the ID field for the matching.

We can see that the name fields contain a mixture of cases and can contain special characters so this will need to be addressed but the formatting functions including in the personMatchR package can handle this. We could simply apply the formatting functions as part of the matching function but for this example we chose to apply the formatting functions, available within the personMatchR package, prior to the matching function, partly to show what is being done and partly to improve the speed of the matching process.

The following code block shows the initial steps to process the medical exemption certificate data, connecting to the database using the nhsbsaR package and then applying the formatting functions:

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

# Load Data: MEDEX ----------------------------------------------------------------------------
# connection using nhsbsaR package
con <- nhsbsaR::con_nhsbsa(database = "DALP")
df_MEDEX <- con %>%
  dplyr::tbl(from = dbplyr::in_schema("STBUC", "DAL1262_MEDEX_RAW"))

#  pass data through the data formatting functions
df_MEDEX <- df_MEDEX %>%
  personMatchR::format_name_db(., FIRST_NAME) %>%
  personMatchR::format_name_db(., SURNAME) %>%
  personMatchR::format_date_db(., DATE_OF_BIRTH) %>%
  personMatchR::format_postcode_db(., POSTCODE)
```

If we looked at the data at this point, we would see the outcome of the formatting operations and be confident that our medical exemption certificate data is in the best state ready for matching against the PDS data:

CERT_NUMBER	SURNAME	FIRST_NAME	DATE_OF_BIRTH	POSTCODE
99999999999	SMITH	JOHN	19990101	NE158NY
88888888888	OBRIEN	RICHARD	19990101	NE16SN

***NB**: data is for illustrative purposes only and does not reflect any real customer information*

Data Preparation: PDS Data

The NHSBSA gets monthly data from PDS for all patients identified on NHS Prescriptions during the previous month, with the PDS data including the latest name and address information available for each identified NHS number.

Looking at the historic PDS response data means that where any of the personal information has changed (e.g. name or address change) we may have multiple sets of data for each patient. This is good news from a patient matching perspective as it means we have more chances of finding a correct match.

The other key piece of information available from PDS for this initiative is the date of death. As with the other personal information, this data can change over time as it reflects the latest known data at the point of response. However, for the date of death we are only really interested in the data from the latest PDS response as this will reflect the latest known status for the patient.

The following table shows an example of what the data from PDS may look like for a single patient:

NHS Number	Month	Surname	Forename	Date of Birth	Postcode	Date of Death	
99999999999	202101	SMITH	JANE	01-Jan-99	NE15 8NY	n/a	
99999999999	202102	BROWN	JANE	01-Jan-99	NE1 6SN	n/a	*changed surname and address
99999999999	202111	BROWN	JANE	01-Jan-99	NE1 6SN	11/10/2021	
99999999999	202112	BROWN	JANE	01-Jan-99	NE1 6SN	10/11/2021	*date of death corrected

***NB**: data is for illustrative purposes only and does not reflect any real customer information*

Ultimately, we want to take all the combinations of the personal information but only the latest known date of death:

NHS Number	Surname	Forename	Date of Birth	Postcode	Date of Death
99999999999	SMITH	JANE	01-Jan-99	NE15 8NY	10/11/2021
99999999999	BROWN	JANE	01-Jan-99	NE1 6SN	10/11/2021

***NB**: data is for illustrative purposes only and does not reflect any real customer information*

We could perform this manipulation as part of the R script, but for simplicity we have pre-processed this data and saved a table in our database containing 93.0 million records and covering 56.5 million distinct NHS numbers. Therefore, we can simply reference this table and apply the same formatting functions as we did to the medical exemption certificate dataset:

```
# Load Data: PDS ------------------------------------------------------------------------
# connection using nhsbsaR package
con <- nhsbsaR::con_nhsbsa(database = "DALP")
df_PDS <- con %>%
  dplyr::tbl(from = dbplyr::in_schema("STBUC", "DAL1262_PDS_DATA"))

#  pass data through the data formatting functions
df_PDS <- df_PDS %>%
  personMatchR::format_name_db(., FORENAME) %>%
  personMatchR::format_name_db(., SURNAME) %>%
  personMatchR::format_date_db(., DOB) %>%
  personMatchR::format_postcode_db(., POSTCODE)
```

Saving prepped datasets

The database functions within the personMatchR use “lazy queries” and therefore the code so far has essentially built a list of conditions that will not be applied until we try to return the data.

Therefore, the formatted datasets need to be saved to database tables, which are what will then ultimately be passed into the matching function.

Please note, the formatting procedure can take some time to run, especially on a dataset the size of PDS with 93 million records, taking just under an hour to complete.

```
# Write data back to database ---------------------------------------------------------
df_MEDEX %>% compute(name = "DAL1262_MEDEX_PF", temporary = FALSE)

df_PDS %>% compute(name = "DAL1262_PDS_PF", temporary = FALSE)
```

Formatting data before versus during matching function

When using the personMatchR package to match across database tables, the formatting steps shown so far must be applied prior to the matching to ensure accurate/efficient operation.

If the personMatchR package is being used for non-database datasets (e.g. imported CSV files) there is a parameter (format_data) in the **calc_match_patients()** function that will simply apply all the relevant formatting as part of the matching process.

This parameter is not included in the **calc_match_patients_db()** function as performing the formatting as part of the matching script produced a considerable performance overhead, in some cases pushing run time from minutes into many hours.

Calling the matching function

Now we have two formatted datasets we can reload these as dbplyr objects which can then be passed as parameters to the ***personMatchR::calc_match_patients_db()*** function alongside the parameters to identify which columns in the datasets contain the key personal information.

For this run we will choose to return all fields from both datasets in the output (output_type = “all”), and we will include details of any medical exemption certificate certificates that could not be matched (inc_no_match = TRUE).

Remember that the functions create a “lazy query” object so the actual processing of the data does not take place until we ask for the results to be returned or saved to a database table, as performed by the compute statement at the end of the code below:

```
# Reload formatted datasets ------------------------------------------------------------------
df_MEDEX <- con %>%
  dplyr::tbl(from = dbplyr::in_schema("STBUC", "DAL1262_MEDEX_PF"))

df_PDS <- con %>%
  dplyr::tbl(from = dbplyr::in_schema("STBUC", "DAL1262_PDS_PF"))


# Call matching function ---------------------------------------------------------------------
df_match_output <- personMatchR::calc_match_patients_db(
  # Data to be matched
  df_one = df_MEDEX,
  id_one = CERT_NUMBER,
  forename_one = FIRST_NAME,
  surname_one = SURNAME,
  dob_one = DATE_OF_BIRTH,
  postcode_one = POSTCODE,
  # Lookup data
  df_two = df_PDS,
  id_two = RECORD_ID,
  forename_two = FORENAME,
  surname_two = SURNAME,
  dob_two = DOB,
  postcode_two = POSTCODE,
  # Other Information
  output_type = "all",
  inc_no_match = TRUE
)

# Save output to database table --------------------------------------------------------------
df_match_output %>% compute(name = "DAL1262_MATCH_OUTPUT", temporary = FALSE)
```

What is the matching function doing?

At a very high level, the activity being performed by the matching functions can be broken down into four main stages.

First the process will identify any records across both datasets where all the pieces of personal information are identical:

Dataset A				
ID	Forename	Surname	Date of Birth	Postcode
1	**JOHN**	**SMITH**	**19990101**	**NE158NY**
2	AMY	SMITH	19991225	NE158NY
3	AMY	BROWN	19990101	NE158NY

Dataset B				
ID	Forename	Surname	Date of Birth	Postcode
1	AMIEE	BROWN	19990101	NE158NY
2	**JOHN**	**SMITH**	**19990101**	**NE158NY**
3	JOHN	SMITH	20001001	NE61SN

Then any remaining records in Dataset A would be checked to see if they could potentially be aligned where some of the information was similar:

Dataset A				
ID	Forename	Surname	Date of Birth	Postcode
~~1~~	~~JOHN~~	~~SMITH~~	~~19990101~~	~~NE158NY~~
2	AMY	SMITH	19991225	NE158NY
3	**AMY**	**BROWN**	**19990101**	**NE158NY**

Dataset B				
ID	Forename	Surname	Date of Birth	Postcode
1	**AMIEE**	**BROWN**	**19990101**	**NE158NY**
2	JOHN	SMITH	19990101	NE158NY
3	JOHN	SMITH	20001001	NE61SN

All potential matches are then scored by comparing the same field in each dataset (e.g. Forename.A v Forename.B) using one of two matching algorithms:

- Forename, Surname and Postcode are scored based on Jaro Winkler similarity
  - 0 = no similarity : 1 = exact match
- DOB is compared using a custom approach similar to Levenshtein Distance, calculating how many characters are different.
  - 0 = exact match

Finally, matches are only included in the results set if they meet one of six rules:

Rule	Forename Score	Surname Score	Postcode Score	DOB Difference
All fields match exactly	1	1	1	0
All fields match exactly - excluding postcode	1	1	n/a	0
All fields match exactly - excluding surname	1	n/a	1	0
DOB is close and all other fields exact	1	1	1	<=2
Forename is close and all other fields exact	>=0.75	1	1	0
DOB is exact and all other fields close	>=0.9	>=0.9	>=0.9	0

Handling the match output

We passed data for 31.5 thousand medical exemption certificate certificates and 93 million PDS records for matching and following a runtime of roughly 30 minutes we received an output table containing 37.5 thousand records.

Clearly this is more than a single record per medical exemption certificate, and although this may be expected where multiple potential “confident” matches were found, in this scenario the excess is due to multiple records in the PDS dataset for each patient where there were only minor differences such as a change of address.

Therefore, we need to perform some manipulation to keep only a single match result for each combination of medical exemption certificate and NHS number, retaining the best scoring match for each combination of records.

```
# Output analysis ------------------------------------------------------------------------
# collect the results into a dataframe
df_results <- con %>% dplyr::tbl(from = dbplyr::in_schema("STBUC", "DAL1262_MATCH_OUTPUT")) %>% collect()

# handle cases where the same MEDEX and NHS_NO were matched more than once
df_results <- df_results %>%
  # rank results by descending MATCH_SCORE for each combination of MEDEX and NHS_NO
  dplyr::group_by(DF1_ID, DF2_NHS_NO_PDS) %>%
  dplyr::arrange(desc(MATCH_SCORE)) %>%
  dplyr::mutate(rank = dense_rank(desc(MATCH_SCORE))) %>%
  dplyr::ungroup() %>%
  # keep only the best match
  dplyr::filter(rank == 1) %>%
  # to tie-break apply second rank to just take first record where scores are tied
  dplyr::group_by(DF1_ID, DF2_NHS_NO_PDS) %>%
  dplyr::arrange(DF2_ID) %>%
  dplyr::mutate(rank = dense_rank(DF2_ID)) %>%
  dplyr::ungroup() %>%
  # ensure "No Match" are not exluded from the results
  dplyr::mutate(rank = tidyr::replace_na(rank, 1)) %>%
  dplyr::filter(rank == 1)

# aggregate by MEDEX certificate and summarise results
df_results <- df_results %>%
  dplyr::group_by(DF1_ID) %>%
  # add fields to show the number of matches and whether a date of death was potentially found
  dplyr::mutate(NHS_NO_MATCHES = case_when(is.na(DF2_NHS_NO_PDS) ~ "No Match",
                                           n_distinct(DF2_NHS_NO_PDS[!is.na(DF2_NHS_NO_PDS)]) == 1 ~ "Single Match",
                                           TRUE ~ "Multiple Matches"),
                DOD_FLAG = case_when(is.na(DF2_NHS_NO_PDS) ~ "No Match",
                                     n_distinct(DF2_DOD[!is.na(DF2_DOD)]) == 0 ~ 'N',
                                     TRUE ~ 'Y')
                ) %>%
  dplyr::ungroup()

# produce a summary output
df_summary <- df_results %>%
  dplyr::group_by(MATCH_TYPE, NHS_NO_MATCHES, DOD_FLAG) %>%
  dplyr::summarise(CERT_COUNT = n_distinct(DF1_ID))
```

Final match results

Once we had cleaned the match output to handle any duplicates, we could then analyse the results and share the results back with the customer.

Of the 31.5 thousand medical exemption certificate holders, 99% could be matched to a patient in the PDS dataset, with almost 86% of certificate holders having an exact match for the personal information checked (forename, surname, date of birth and postcode).

Over 13% of certificate holders had a confident match, where the matching process identified a close match between the two datasets, and only around 1% of certificate holders could not be matched at all to PDS.

A date of death could be identified for just over one thousand certificate holders, representing over 3% of the certificate checked.

Match Outcome	No. of MEDEX certificates	Proportion of MEDEX certificates
MEDEX matched exactly to a single NHS number which has a date of death	901	2.9%
MEDEX matched exactly to a single NHS number which has no date of death	26,061	82.7%
MEDEX matched exactly to multiple NHS numbers, of which none have a date of death	10	0.0%
MEDEX matched confidently to a single NHS number which has a date of death	139	0.4%
MEDEX matched confidently to a single NHS number which has no date of death	4,089	13.0%
MEDEX matched confidently to multiple NHS numbers, of which none have a date of death	13	0.0%
MEDEX unable to be matched to PDS	314	1.0%
Total	**31,527**	**100.0%**

Conclusion

Using the personMatchR package we were able to complete the matching exercise in hours rather than the days that this type of task had taken previously.

As no code needing producing to perform the actual matching, this also reduced the resource required to review this and limited the scope for errors to be made.

For this particular use case, around 3% of certificate holders checked had a date of death this suggests there could be some value in identifying this information prior to issuing certificate renewal reminders. If this process was to become part of the business-as-usual approach it would be possible to include the personMatchR process as part of a monthly run to check this information as part of the reminder process.