Record Linkage Using R

Jared Parrish, PhD

August 12, 2024

Contents

This work was funded by the Office of the Secretary Patient-Centered Outcomes Research Trust Fund (OS-PCORTF). The OS-PCORTF funding was made available to Centers for Disease Control and Prevention by the Office of the Assistant Secretary for Planning and Evaluation (ASPE) through Interagency Agreement (IAA) 750120PE090052. This report was funded by the Centers for Disease Control and Prevention under the OT18-1802 Cooperative Agreement, Grant #6NU38OT000290 The findings and conclusions in this document are those of the authors and do not necessarily represent the official position of Centers for Disease Control and Prevention, or the other organizations involved, nor does the mention of trade names, commercial products, or organizations imply endorsement by the U.S. Government.

Background

Record linkage is the process of identifying and merging records that refer to the same entity across different data sets. This is particularly useful in scenarios where unique identifiers are not available or reliable. In R, two popular packages for performing record linkage are RecordLinkage and FastLink.

Here is a user-friendly HTML version of this document, as well as the source code and files used in the linkages.

RecordLinkage Package

The RecordLinkage package provides tools for probabilistic record linkage and deduplication. It allows users to compare records based on various attributes, calculate similarity scores, and determine matches based on thresholds. Some of the key features included in this package are:

- **Pairwise Comparison:** Compares all pairs of records from different datasets or within the same dataset.
- **Similarity Measures:** Utilizes various similarity measures such as Jaro-Winkler, Levenshtein distance, and Soundex for string comparisons.
- **Blocking:** Reduces the number of comparisons by grouping records into blocks based on certain criteria, thereby improving efficiency.
- **Probabilistic Matching:** Implements the Fellegi-Sunter model for probabilistic record linkage and other methods, which calculates the likelihood of matches based on attribute similarities.
- **Other Features:** Both supervised and unsupervised machine learning methods, manual review match status editing, optimized threshold review specification (e.g., ParetoThreshold), and other linkage tuning and assessment tools.

FastLink Package

The fastLink package is designed for large-scale record linkage tasks and focuses on scalability and speed. It extends the functionalities of traditional probabilistic linkage methods and provides tools for estimating and visualizing the linkage process. Some of the key features included in this package are:

- **Scalability:** Efficiently handles large datasets with millions of records by leveraging parallel computing.
- **Advanced Blocking**: Implements multiple blocking schemes to ensure that the linkage process remains computationally feasible without sacrificing accuracy.
- **Unsupervised Learning:** Automatically estimates parameters required for record linkage without the need for training data.
- **Visualization Tools:** Provides tools for visualizing the linkage results and understanding the matching process.

Both RecordLinkage and fastLink packages offer robust solutions for record linkage tasks in R. While RecordLinkage is more flexible and offers a variety of similarity measures and probabilistic matching, fastLink is optimized for performance and scalability, making it suitable for large data sets.

This exercise will focus on and provide an example using the RecordLinkage package as the approach is largely extendable to the fastLink package. For those interested in learning and using the fastLink package, check out these resources:

- An introduction to fastLink for probabilistic record linkage
- R package fastLink: Fast Probabilistic Record Linkage

The example below demonstrates one approach to using the RecordLinkage package to perform record linkage in R. This includes loading data, pre-processing (cleaning) the data, comparing records, and identifying matches. While this example is not exhaustive and does not cover all the features provided by the package, it serves as a solid starting point for conducting record linkages.

Linkage Set-up

Prior to any linkage project, it is important to learn about each dataset as well as the elements contained within them, and have context for population distributions in the records being linked. It is also critical that you have your linkage purpose and expectation well-defined.

The code and output below assume at a least a basic understanding of R; for example, how to install and load packages. The data sets used for this exercise were generated using two different generative models (ChatGPT and Google Gemini) and do not reflect actual individuals.

For this example, the datasets were created to represent the real-life scenario where a health agency draws a sample from a birth record (dat2) and then links those sampled birth records to a population source (dat2) (e.g., hospital discharge records) where the individual may have multiple visits. The expectation is that we will link and retain all individuals from the sample and only the matching population records. This is important to establish up front because during the linkage process, we will identify the matching records and need to be sure to retain all non-matches from the sample (their values will be NULL/NA for all elements in the population source).

We first need to load a few packages and data for this exercise. If you don't have these packages, you'll need to install the libraries. This exercise was completed using R version 4.4.0.

Load packages

```
library(RecordLinkage)
library(stringdist) # for example with cooperators.

library(dplyr) library(tidyr)
library(lubridate)     # working with dates
library(stringi)       # for cleaning strings


library(knitr)         # for printing basic tables in the markdown
library(kableExtra) # for improved tables in the markdown document
library(ggplot2)       # for plotting

library(summarytools) # For summarizing data frames
library(skimr)           # for summarizing data frames

# Shut off scientific notation, this is critical for long numeric ID's
options(scipen=999)
```

Install Data

```
# Read in data sets to be linked for this
example # Population source
git1<-'https://raw.githubusercontent.com/parrish-epi/R-recordLinkage/main/SourceA.csv'
dat1<-read.csv(git1, fileEncoding = "ISO-8859-1")

# Sample source
git2<-'https://raw.githubusercontent.com/parrish-epi/R-recordLinkage/main/SourceB.csv'
dat2<-read.csv(git2, fileEncoding = "ISO-8859-1")

# add the data set to the start of the ID for convenience.
dat1$UNID <- paste0("dat1_",dat1$UNID)
dat2$ID_source <-
paste0("dat2_",dat2$ID_source)

#remove url sets
rm('git1','git2')
```

Review Datasets

After the data have been loaded, the first step is to inspect and review the data. During this process, we want to make sure that the data elements to be used for linkages (partial identifiers) are all formatted the same and all column names are identical. For these examples we have two datasets: dat1 and dat2.

dat1 has 1183 records and 9 variables.
dat2 has 302 records and 8 variables.

Let's first start by checking what names are in each of the datasets.

```
# View contents of data set 1
names(dat1)
```

```
## [1] "ID"                "Last.Name"        "First.Name"
                            "Middle.Name" ## [5] "Date.of.Birth" "Sex" "Race"
                            "UNID"
## [9] "InSample"
```

```
# View contents of data set 2
names(dat2)
```

```
## [1] "ID"              "ID_source"     "GivenName"   "MiddleName"
"FamilyName" ## [6] "Sex"             "DOB"           "Race"
```

Review Data Structure

The base r function str() is helpful for viewing the contents of the data but dplyr::glimpse is also useful.

```
# View contents of data set 1
str(dat1)
```

```
## 'data.frame':      1183 obs. of    9 variables:
##   $ ID              : int    1 2 3 4 6 7 8 9 10 12
##   $ Last.Name       : chr    "Patel" "Kim" "García" "Chan" ...
##   $ First.Name      : chr    "Amara " "Malik " "Sofia " "Hiro " ... ##   $
Middle.Name         : chr    "Mei" "Giovanni" "" "" ...
##   $ Date.of.Birth: chr       "6/28/2016" "8/18/2017" "9/22/2019" "7/17/2018" ... ##
      $ Sex             : chr    "Female" "Male" "Female" "Male" ...
##   $ Race            : chr    "Black" "White" "Black" "NHOPI" ...
##   $ UNID            : chr    "dat1_OFDEBNBNST" "dat1_98AQ2AIWZX" "dat1_R8EFSTH5I0" "dat1_MBR9ABVV5I" ...
##   $ InSample        : int    NA NA NA 1 NA NA 1 1 NA NA ...
```

```
# View contents of data set 2
str(dat2)
```

```
## 'data.frame':      302 obs. of    8 variables:
##   $ ID           : int    4 8 9 15 16 22 23 24 28 34 ...
##   $ ID_source : chr       "dat2_2018AB29797" "dat2_2019AB15569" "dat2_2019AB07650" "dat2_2016AB37928" ...
##   $ GivenName : chr       "HIRO " "JOON " "KATARINA " "NADIA " ...
##   $ MiddleName: chr       "NIKITA" "LEILA" "WOLFGANG" "RAVI" ... ##
      $ FamilyName: chr       "CHAN" "SMITH" "GONZÁLEZ" "LEE" ...
##   $ Sex          : chr    "M" "M" "F" "F" ...
##   $ DOB          : chr    "2018-07-17" "2019-07-22" "2019-11-09" "2016-04-16" ...
##   $ Race         : chr    "NHOPI" "Black" "Asian" "NHOPI" ...
```

Align Feature Order
For the RecordLinkage package, we need to align the elements. In this exercise, we will align dat1 variable order to dat2.

```
# Making the order of the variables in the data set the same.
dat1 <- dat1[,c(1,8,3,4,2,6,5,7,9)]
str(dat1)
```

```
## 'data.frame':      1183 obs. of    9 variables:
##   $ ID              : int    1 2 3 4 6 7 8 9 10 12 ...
##   $ UNID            : chr    "dat1_OFDEBNBNST"  "dat1_98AQ2AIWZX"  "dat1_R8EFSTH5I0"  "dat1_MBR9ABVV5I"  ...
##   $ First.Name      : chr    "Amara " "Malik " "Sofia " "Hiro " ... ##   $
Middle.Name         : chr    "Mei" "Giovanni" "" "" ...
##   $ Last.Name       : chr    "Patel" "Kim" "García" "Chan" ... ##
      $ Sex             : chr    "Female" "Male" "Female" "Male" ...
##   $ Date.of.Birth: chr       "6/28/2016" "8/18/2017" "9/22/2019" "7/17/2018" ... ##
      $ Race            : chr    "Black" "White" "Black" "NHOPI" ...
##   $ InSample        : int    NA NA NA 1 NA NA 1 1 NA NA ...
```

Identify duplicates

```
# Duplicates in dat1
table(duplicated(dat1$UNID))
```

```
##
## FALSE
         TRUE ##
         986
         197
```

```
# Duplicates in dat2
table(duplicated(dat2$ID_source))
```

```
##
## FALSE
##
         30
2
```

Align Datasets

Next, we are going to align dat1 variable names and structure to that of dat2. Remember that the variable names, order, and structure must be the same in both datasets for the RecordLinkage package.

Modify dat1

```
dat1a <- dat1 %>%
    # ensure names of variables are the same between the two data sets.
    rename(
        GivenName = First.Name,
        MiddleName = Middle.Name,
        FamilyName = Last.Name, DOB
        = Date.of.Birth, ID_source =
        UNID
    ) %>%
    # ensure coding of variables is the same between the two data sets.
    # perform basic data cleaning (up case, remove white space and spcial characters)
    mutate(
        Sex = ifelse(Sex == "Female", "F", ifelse(Sex == "Male", "M", NA)), DOB = mdy(DOB),
        across(contains("Name"), ~ toupper(.)), across(everything(),
        stringi::stri_trim),
        GivenName = gsub("[[:punct:][:blank:]]+", "", GivenName),
        MiddleName = gsub("[[:punct:][:blank:]]+", "", MiddleName),
        FamilyName = gsub("[[:punct:][:blank:]]+", "", FamilyName),
        across(c("GivenName",  "MiddleName",  "FamilyName"),
                   ~ stringi::stri_trans_general(str = ., id = "Latin-ASCII")),
        # Break apart date if needing to block to facilitate iterative block linking.
        DOB_YR = as.integer(year(DOB)), DOB_MO =
        as.integer(month(DOB)), DOB_DY =
        as.integer(day(DOB))
        ) %>%
  # Remove indicator specifying correct linkages (part of the test data for checking #      accuracy but will
not have this in a real data linkage)
  select(-InSample)
```

De-duplicate records

It is easiest to link de-duplicated records. This reduces redundancy and improves the estimated probability scores. It is important in a duplicated data set that represents unique events (e.g., hospital visits) to have an individual unique ID representing the patient, and a unique record level ID representing the visit (event).

```
dat1a.dedup <- dat1a %>% unique()
```

Modify dat2

```
dat2a <- dat2 %>%
  # conduct the same data cleaning for dat2 as completed in dat1.
  mutate(across(everything(), stringi::stri_trim)) %>% mutate(
    DOB_YR = as.integer(year(DOB)), DOB_MO =
    as.integer(month(DOB)), DOB_DY =
    as.integer(day(DOB)),
    across(c("GivenName", "MiddleName", "FamilyName"),
           ~ gsub("[[:punct:][:blank:]]+", "", .)),
    GivenName = stringi::stri_trans_general(str = GivenName, id = "Latin-ASCII"), MiddleName
    = stringi::stri_trans_general(str = MiddleName, id = "Latin-ASCII"), FamilyName =
    stringi::stri_trans_general(str = FamilyName, id = "Latin-ASCII")
  )
```

Review Aligned Data Sets

View contents of the two data sets

Here we will use the dplyr::glimpse() function so we can quickly see the number of rows and columns as well.

```
# data structure for dat1 that has been cleaned and de-duplicated.
dplyr::glimpse(dat1a.dedup)
```

```
## Rows: 986
## Columns: 11
## $ ID                <chr> "1", "2", "3", "4", "6", "7", "8", "9", "10", "12", "13", "~
## $ ID_source <chr> "dat1_OFDEBNBNST", "dat1_98AQ2AIWZX", "dat1_R8EFSTH5I0", "d~ ## $
GivenName <chr> "AMARA", "MALIK", "SOFIA", "HIRO", "ALEJANDRO", "AISHA", "J~ ## $
MiddleName <chr> "MEI", "GIOVANNI", "", "", "", "", "", "", "", "", "", "RAV~ ## $ FamilyName
<chr> "PATEL", "KIM", "GARCIA", "CHAN", "SANCHEZ", "NGUYEN", "SMI~ ## $ Sex        <chr>
"F", "M", "F", "M", "M", "F", "M", "F", "M", "M", "F", "F",~ ## $ DOB     <chr> "2016-06-28",
"2017-08-18", "2019-09-22", "2018-07-17", "20~
## $ Race              <chr> "Black", "White", "Black", "NHOPI", "AIAN", "AIAN", "Black"~ ## $
DOB_YR    <int> 2016, 2017, 2019, 2018, 2017, 2018, 2019, 2019, 2017, 2019,~
## $ DOB_MO    <int> 6, 8, 9, 7, 9, 11, 7, 11, 12, 11, 11, 4, 4, 11, 5, 10, 11, ~
## $ DOB_DY    <int> 28, 18, 22, 17, 27, 7, 22, 9, 27, 24, 29, 16, 4, 23, 31, 29~
```

```
# data structure for dat2 that has been cleaned.
dplyr::glimpse(dat2a)
```

```
## Rows: 302
## Columns: 11
## $ ID            <chr> "4", "8", "9", "15", "16", "22", "23", "24", "28", "34", "3~
## $ ID_source  <chr> "dat2_2018AB29797", "dat2_2019AB15569", "dat2_2019AB07650",~ ##
$ GivenName <chr> "HIRO", "JOON", "KATARINA", "NADIA", "EMRE", "ALIYAH", "OLI~ ## $
MiddleName <chr> "NIKITA", "LEILA", "WOLFGANG", "RAVI", "AMALIA", "ALINA", "~ ## $
FamilyName <chr> "CHAN", "SMITH", "GONZALEZ", "LEE", "JOHNSON", "HERNANDEZ",~ ## $
Sex              <chr> "M", "M", "F", "F", "M", "F", "M", "F", "F", "F", "M", "F",~ ## $ DOB
                 <chr>  "2018-07-17",  "2019-07-22",  "2019-11-09",  "2016-04-16",  "20~
## $ Race        <chr> "NHOPI", "Black", "Asian", "NHOPI", "Black", "Black", "Whit~ ## $
DOB_YR    <int> 2018, 2019, 2019, 2016, 2020, 2017, 2016, 2018, 2018, 2020,~
## $ DOB_MO    <int> 7, 7, 11, 4, 4, 2, 8, 10, 3, 8, 4, 10, 9, 6, 3, 12, 7, 10, ~
## $ DOB_DY    <int> 17, 22, 9, 16, 4, 2, 14, 15, 3, 23, 18, 23, 5, 25, 19, 31, ~
```

Data preparation is critical for linkages. Before we move forward with the actual linkages, let's first inspect the first five records of each dataset. Table 1 below displays the first five records from dataset 1, and Table 2 displays the first five records from dataset 2.

Table 1: *First five records of aligned dataset 1 (dat1)*

ID	ID_source	GivenName	MiddleName	FamilyName	Sex	DOB	Race	DOB_YR	DOB_MO	DOB_DY
908	dat1_9UTHP2F0IN	AHMED	N	ABDULLAH	M	2016-09-14		2016	9	14
725	dat1_WVW407E1DJ	DANIELA	ROCIO	ACOSTA	F	2017-12-26	NHOPI	2017	12	26
137	dat1_AYPEOEQLU0	ROBIN	EMILY	ADAMS	F	2020-09-23	Black	2020	9	23
263	dat1_YQC5VAP7S3	MONIQUE	DANIELLE	ADAMS	F	2016-06-07	Black	2016	6	7
484	dat1_F6SMJX9ZAC	JASON	A	ADAMS	F	2020-12-18	White	2020	12	18

Table 2: *First five records of aligned dataset 2 (dat2)*

ID	ID_source	GivenName	MiddleName	FamilyName	Sex	DOB	Race	DOB_YR	DOB_MO	DOB_DY
263	dat2_2016AB35131	MONIQUE	DANIELLE	ADAMS	F	2016-06-07	Black	2016	6	7
484	dat2_2020AB59638	JASON	AARON	ADAMS	M	2020-12-18	White	2020	12	18
750	dat2_2019AB06664	CAMILA	JOSEFINA	AGUAYO	F	2019-03-16	Asian	2019	3	16
729	dat2_2019AB74312	ANABEL	LOURDES	AGUILAR	F	2019-03-09	Black	2019	3	9
36	dat2_2016AB66469	YARA	ADITI	AHMED	F	2016-10-23	Black	2016	10	23

Finally, the skim function in the skimr() package provides a really nice tool for reviewing the data elements in the data frame.

Skim function output for Dat1 - normalized and de-duplicated

```
-- Data Summary
                              --------------------------
   Values
Name                          dat1a.dedup
Number of rows                986
Number of columns             11
_______________________
Column type frequency:
  character                   8
  numeric                     3
________________________
Group variables
--                            Non
   e Variable type: character
```

```
                                   ---------------------------------------------------------------------------------------
   skim_variable       n_missing   complete_rate  min  max  empty   n_unique  whitespace
 1 ID                          0               1    1    4      C        986           0
 2 ID_source                   0               1   15   15      C        986           0
 3 GivenName                   0               1    2   11      C        471           0
 4 MiddleName                  0               1    0   11    131        353           0
 5 FamilyName                  0               1    2   17      C        355           0
 6 Sex                        54           0.945    1    1      C          2           0
 7 DOB                         8           0.992   10   10      C        728           0
 8 Race                        0               1    0    5     19          6           0

     Variable type: numeric
   skim_variable  n_missing        complete_rate   mean    sd    p0   p25   p50   p75  p100
 1 DOB_YR                      8           0.992  2018.    1.31  2015  2017  2018  2019  2020
 2 DOB_MO                      8           0.992    6.83   3.51     1     4     7    10    12
 3 DOB_DY                      8           0.992   15.2    8.77     1     8    15    23    31

--                                 ------------------------------------------------------
```

Skim function output for Dat2 - normalized

```
-- Data Summary
                          --------------------------
     Values
Name                          dat2a
Number of rows                302
Number of columns             11
___________________________
Column type frequency:
   character                  8
   numeric                    3
___________________________
Group variables

--                            Non

     e Variable type: character

-- Variable type: numeric   ---------------------------------------------------------------------------------------
     skim_variable    n_missing    complete_rate  min  max  empty   n_unique  whitespace
1  ID                         0                1    1    3      C        302           0
2  ID_source                  0                1   16   16      C        302           0
3  GivenName                  0                1    3   11      C        222           0
4  MiddleName                 0                1    2   11      C        177           0
5  FamilyName                 0                1    3   17      C        161           0
6  Sex                        0                1    1    1      C          2           0
7  DOB                        0                1   10   10      C        270           0
8  Race                       0                1    4    5      C          5           0

                              ---------------------------------------------------------------------------------------
     skim_variable    n_missing    complete_rate   mean    sd    p0   p25   p50   p75  p100
1  DOB_YR                     0                1  2018.    1.32  2016  2017  2018  2019  2020
2  DOB_MO                     0                1    6.96   3.51     1     4     7    10    12
3  DOB_DY                     0                1   15.5    8.60     1     8    15    23    31
```

Data linkage

Now that the two data sets to be combined are cleaned in the same way (normalized) and have the same naming and ordering of variables (standardized) we are ready to begin our linkages. The following figure provides a basic linkage flow diagram outlining the major points of a linkage project.

Identifier Assessment

It is helpful to start by reviewing how many duplicate records have the same combination of identifiers. This will help establish how "unique" the combination of identifiers is and what identifiers are needed to support accurate results.

For this exercise, we will use the GivenName, MiddleName, FamilyName, Sex, and DOB for our linkages. We'll then look at the combination uniqueness without DOB and then with just year of birth and then day of birth. This exercise is just to demonstrate the investigations that one can do to get an idea of the types of linkages to expect with the combination of identifiers to be used.

Review of dat 1 identifiers

```
# start by creating a little function to summarize and view each variable
```

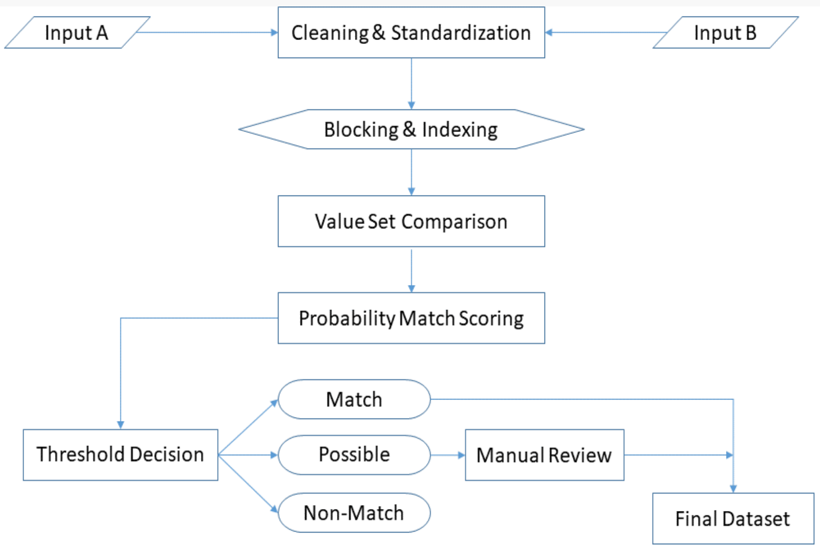


Figure 1: *General Data Linkage Flow Diagram*

```
# based on GivenName, FamilyName, Sex, and DOB
dat1a.dedup %>% group_by(GivenName,FamilyName,Sex,DOB) %>%
                        summarise(Count = n()) %>%
                        ungroup() %>%
                        group_by(Count) %>%
                        summarise(Dist = n())
```

```
## # A tibble: 1 x 2 ##
       Count   Dist
##     <int> <int>
## 1        1
       986
```

```
# based on GivenName, FamilyName, and Sex
dat1a.dedup %>% group_by(GivenName,FamilyName,Sex) %>%
                      summarise(Count = n()) %>%
                      ungroup() %>%
                      group_by(Count) %>%
                      summarise(Dist = n())
```

```
## # A tibble: 3 x 2 ##
       Count   Dist
##     <int> <int>
## 1        1
       881
## 2        2      30
## 3        3      15
```

```
# based on GivenName, FamilyName, Sex, and year
dat1a.dedup %>% group_by(GivenName,FamilyName,Sex, DOB_YR) %>% summarise(Count
                      = n()) %>%
                      ungroup() %>%
                      group_by(Count) %>%
                      summarise(Dist = n())
```

```
## # A tibble: 3 x 2 ##
       Count   Dist
##     <int> <int>
## 1        1
       963
## 2        2      10
## 3        3       1
```

```
# based on GivenName, FamilyName, Sex, and month
dat1a.dedup %>% group_by(GivenName,FamilyName,Sex, DOB_DY) %>% summarise(Count
                      = n()) %>%
                      ungroup() %>%
                      group_by(Count) %>%
                      summarise(Dist = n())
```

```
## # A tibble: 2 x 2 ##
       Count   Dist
##     <int> <int>
## 1        1
       974
## 2        2       6
```

From this quick assessment, we see that for the combination of identifiers to be used we have zero (0) duplicates which is great. However, we may need to "relax" the matching criteria by limiting the number of variables used. We can see that dropping date of birth (DOB) will induce some ubiquity for potentially matching records; however, use of day may be a feasible approach if we're trying to increase the sensitivity of detection.

Review of dat2 identifiers

```
# start by creating a little function to summarize and view each variable

# based on GivenName, FamilyName, Sex, and DOB
dat2a %>% group_by(GivenName,FamilyName,Sex,DOB) %>%
                    summarise(Count = n()) %>%
                    ungroup() %>% group_by(Count)
                    %>% summarise(Dist = n())
```

```
## # A tibble: 1 x 2 ##
      Count   Dist
##    <int> <int>
## 1      1
      302
```

```
# based on GivenName, FamilyName, and Sex
dat2a %>% group_by(GivenName,FamilyName,Sex) %>%
                    summarise(Count = n()) %>%
```

```
                    ungroup()  %>%
                    group_by(Count)  %>%
                    summarise(Dist  =  n())
```

```
## # A tibble: 2 x 2 ##
      Count   Dist
##    <int> <int>
## 1      1
      288
## 2      2      7
```

```
# based on GivenName, FamilyName, Sex, and year
dat2a %>% group_by(GivenName,FamilyName,Sex, DOB_YR) %>%
                    summarise(Count = n()) %>%
                    ungroup() %>% group_by(Count)
                    %>% summarise(Dist = n())
```

```
## # A tibble: 2 x 2 ##
      Count   Dist
##    <int> <int>
## 1      1
      300
## 2      2      1
```

```
# based on GivenName, FamilyName, Sex, and month
dat2a %>% group_by(GivenName,FamilyName,Sex, DOB_DY) %>%
                        summarise(Count = n()) %>%
                        ungroup() %>% group_by(Count)
                        %>% summarise(Dist = n())
```

```
## # A tibble: 1 x 2 ##
       Count    Dist
##     <int> <int>
## 1         1
       302
```

Like we saw for dat1, to use the partial identifiers we have zero (0) duplicates and similar results when dropping date of birth and looking at individual components of the date.

Deterministic Matching

It is always a good idea to do an exact match (deterministic link) on the combination of identifiers that will be used to set a baseline. This baseline will help you determine how much different (hopefully better) the linkages are by incorporating probabilistic matching. We've seen from our assessment above that using GivenName, LastName, Sex, and DOB to merge will not result in duplicate matches.

We are going to demonstrate a simple baseline deterministic match by merging the data sets together.

Here we merge dat1a.dedup into dat2a (right join), based on GivenName, LastName, Sex, and DOB. Because we don't want duplicate variables, aside from the ID and ID_source variables, we drop those that aren't involved with the merge.

Table 3: *First five records of merged datasets*

ID.x	ID_source.x	GivenName	FamilyName	Sex	DOB	ID.y	ID_source.y	MiddleName	Race	DOB_YR	DOB_MO	DOB_DY
263	dat1_YQC5VAP7S3	MONIQUE	ADAMS	F	2016-06-07	263	dat2_2016AB35131	DANIELLE	Black	2016	6	7
NA	NA	JASON	ADAMS	M	2020-12-18	484	dat2_2020AB59638	AARON	White	2020	12	18
NA	NA	CAMILA	AGUAYO	F	2019-03-16	750	dat2_2019AB06664	JOSEFINA	Asian	2019	3	16
729	dat1_WNT1JPAJ5F	ANABEL	AGUILAR	F	2019-03-09	729	dat2_2019AB74312	LOURDES	Black	2019	3	9
NA	NA	YARA	AHMED	F	2016-10-23	36	dat2_2016AB66469	ADITI	Black	2016	10	23

```
MergedDat  <-  dat1a.dedup  %>%
                   select(-MiddleName, -Race, -DOB_YR, -DOB_MO, -DOB_DY) %>%
                   right_join(dat2a,  by  =  c("GivenName"  =  "GivenName",
                                               "FamilyName"  =  "FamilyName",
                                               "Sex"  =  "Sex",
                                               "DOB"  =  "DOB"))
```

Because we did a right join, we should retain the same number of records as we had in dat2. Let's not assume anything and check. Afterward, we'll explore how many individuals in our dat2 did not merge with a record in dat1. From this we'll calculate the linkage rate.

```
nrow(MergedDat) == nrow(dat2a)
```

```
## [1] TRUE
```

```
# number of our sample source (dat2) that linked with a record in the # population source
(dat1).

SumLink <-
MergedDat
%>%
  summarise(
    Linked = sum(!is.na(ID_source.x)), nonLinked =
    sum(is.na(ID_source.x))
  ) %>%
  pivot_longer(cols = everything(), names_to = "value", values_to = "count") %>% mutate(
    proportion = count / sum(count)
  )
as.data.frame(SumLink)
```

```
##          value count proportion
## 1       Linked     205
            0.6788079
## 2 nonLinked        97    0.3211921
```

Using an exact match linkage (merge), the linkage "rate" was 67.9%

Probabilistic Matching

Now that we've normalized, standardized, and conducted a baseline merge, we are ready to conduct a probabilistic linkage. We are only going to go through a single approach to showcase how some of the tools work.

The RecordLinkage package has a lot of nice features but does have some limitations. The most notable is it can be slow and require a lot of memory for large data sets. The authors provide two approaches

"RLBigDataLinkage" and "compare.linkage." With most public health data sets the RLBigDataLinkage() will need to be used. Because of this, although the example could use the compare.linkage, we will use the former for the example.

Another limitation is the available string comparisons are limited to Jaro-Winkler, Levenshtein, and Soundex. Only a single comparator can be applied (i.e, no option to apply different comparators to different variables).

We will use the Jaro-Winkler comparator without any blocking variables.

Create matched pairs

```
#' first create a record pairs object. We are specifying the two data sets, the #' string comparator, and
the specifying the referential location of the columns #' to NOT include when considering a match. This
is why it is critical that the #' data sets are ordered and named the same!
rpairs <- RecordLinkage:: RLBigDataLinkage(dataset1 = dat2a,
                                           dataset2 = dat1a.dedup,
                                           strcmp = TRUE, strcmpfun = "jarowinkler", exclude
                                           = c(1,2,4,8,9,10,11))

# If using a really large sample, block and iterate
# (blocking subsets by exact match): this would block on Year
## Not RUN ##
# rpairs <- RecordLinkage:: RLBigDataLinkage(dataset1 = dat2a, #      dataset2
= dat1a.dedup,
#                                            strcmp = TRUE, strcmpfun = "jarowinkler",
#                                            exclude = c(1,2,4,8,9,10,11),
#                                            blockfld = c(9)
#                                            )
```

Once the comparisons have been made using the criteria specified, we will next apply the algorithm for calculating the probability of a match. Probably the most used is the Fellegi and Sunter stochastic frame-work. The RecordLinkage package, however, also provides the EpiLink and EM algorighms (see package documentation for details).

The selected cutoff level depends on the algorithm choice, type of comparator, and level of desired sensitiv- ity/specificity.

For this example, we'll use the EpiLink algorithm.

```
# create weights for each record pair
rpairs <- RecordLinkage::epiWeights(rpairs)
print(rpairs)
```

```
##
## Linkage Data Set for large number of data ##
## 302 records in first data set ## 986
records in second data set ## 297772
record pairs
```

```
# wts_fs <- RecordLinkage::fsWeights(rpairs)          #Fellegi and Sunter (pair with fsClassify) # wts_eu <-
RecordLinkage::emWeights(rpairs)                      #M and U probabillity assessment
#                   using the EM algorithm (pair with emClassify)
```

The respective weight functions calculate the probability of a match for each record in data set 1 with each record in data set 2 based on the comparator patterns specified. We need to classify the weights as links, potential links, or non-links. To make these classifications, we need to establish our acceptance threshold. The threshold is the probability level that above would be accepted as a link or below would be rejected. The threshold may also be a region where above would be accepted as a link, within would be potential, and below would be rejected. The area of uncertainty (potential matches) requires some sort of reconciliation (record review).

As indicated above, choosing the thresholds is usually done through record review and classification. One of the tools in the RecordLinkage package is the getParetoThreshold() function, which can also be useful for identifying acceptance thresholds. However, with large data sets it can take a long time to run.

It is often best to run it a few times with samples taken from the data or a smaller training data set.

```
# due to the time it takes to run, the function has been commented out and # replaced with the
result after running it once.

# pth <- RecordLinkage::getParetoThreshold(rpairs,interval=c(0.75, 1.0))
   pth <- 0.875
# note you can leave out the interval statement to choose the threshold from a plot. #
RecordLinkage::getParetoThreshold(rpairs)
```

Based on the image from the getParetoThreshold() (see below) the long flat arm suggesting that it will be unlikely to identify correct matches below a weight of around 0.8. Based on the interval selected the suggested automatic acceptance threshold is 0.875.

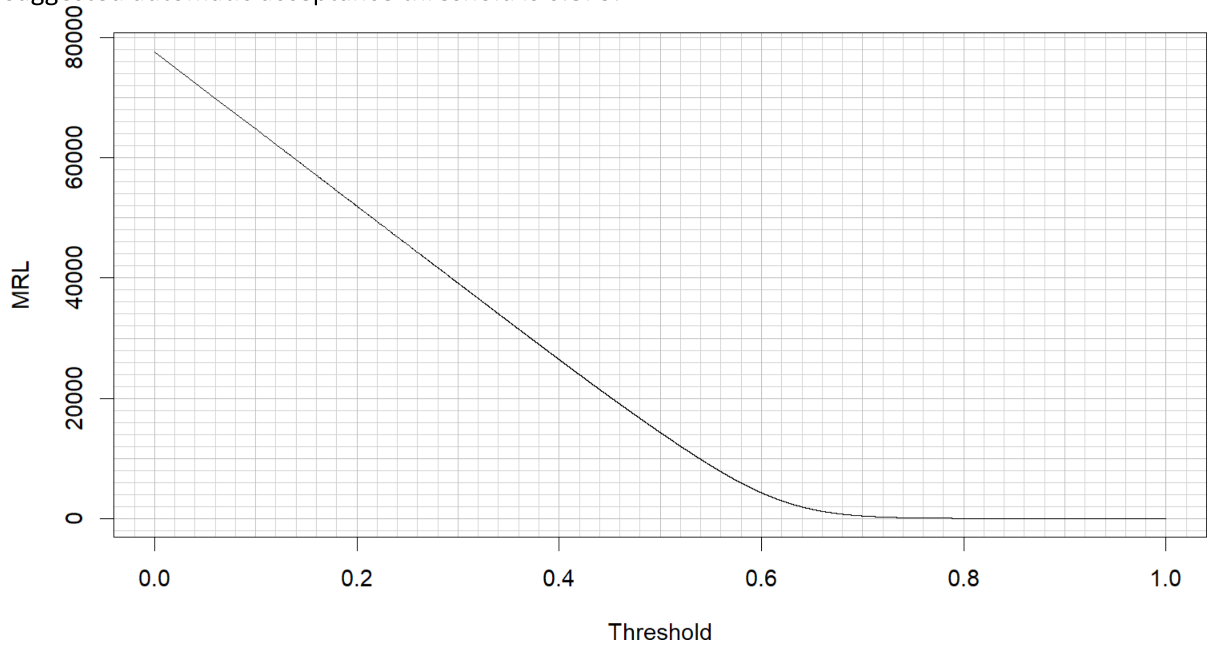


Figure 2: *Mean Residual Life (MRL) plot.*

Classify Weights

Now we'll use the information and classify linkages. According to the RecordLinkage documentation for the epiClassify() function. "All record pairs with weights greater or equal threshold.upper are classified as links. Record pairs with weights smaller than threshold.upper and greater or equal threshold.lower are classified as possible links. All remaining records are classified as non-links."

Classify based on the ParetoThreshold

```
# First classify links based on the ParetroThreshold
eclsf <- RecordLinkage::epiClassify(rpairs, pth)
# Next classify and return pairs with a probability between 0.75 and 1.0
RlMtch <- RecordLinkage::getPairs(
   eclsf, min.weight=0.75, max.weight=1, single.row=FALSE
   )
```

```
## ===========================================================================
```

```
# Finally, update is_match based on the Class
RlMtch$is_match <- with(RlMtch, ifelse(is.na(is_match) & Class=="L",1,""))
```

Using the ParetoThreshold() as an automatic acceptance alone will likely result in poor results. As we can see. The linkages "L" in the table below suggest 389 linkages, when we only have 302 records. This is incorrect and leads to false positive matches. However, it is a great starting point for considering what levels will optimize manual review and minimize the number of reviews needed.

```
table(RlMtch$Class,useNA = "always")
```

```
##
##              L       N
<NA> ## 8164        389
3693        0
```

Classify informed by the ParetoThreshold with manual review Now we'll specify intervals. We'll automatically accept those with a probability of 1.0 and then review those with a probability between 0.8 and 0.99.

```
# First classify links based on the ParetroThreshold
eclsf_1 <- RecordLinkage::epiClassify(
   rpairs, threshold.upper = 1.0, threshold.lower = 0.75
   )
# Next classify and return pairs with a probability between 0.75 and 1.0
RlMtch_1  <-  RecordLinkage::getPairs(
   eclsf_1, min.weight=0.75, max.weight=1, single.row=FALSE
   )
```

```
## ================================================================================
```

```
# Finally, update is_match based on the Class.
RlMtch_1$is_match  <-  with(RlMtch_1,  ifelse(is.na(is_match)  &  Class=="L",1,""))
```

Let's look at the numbers linked and possible matches that need review by probability score.

```
table(RlMtch_1$Class)
```

```
##
##              L       P
## 8164     205 3877
```

```
# create bins for the probability scores (weights)
RlMtch_1$cutpnt  <-  with(RlMtch_1,  cut(round(as.numeric(Weight),2),
                                         breaks=(seq(0.75,1, by = .01)), include.lowest =
                                         TRUE, right = TRUE), "")
# print the counts by bin
table(RlMtch_1$cutpnt)
```

```
##
## [0.75,0.76]   (0.76,0.77]   (0.77,0.78]   (0.78,0.79]   (0.79,0.8]    (0.8,0.81]
## 1154          592           474           362           309           248
## (0.81,0.82]   (0.82,0.83]   (0.83,0.84]   (0.84,0.85]   (0.85,0.86]   (0.86,0.87]
## 180           136           95            68            47            28
## (0.87,0.88]   (0.88,0.89]   (0.89,0.9]    (0.9,0.91]    (0.91,0.92]   (0.92,0.93]
## 18            11            6             9             13            0
## (0.93,0.94]   (0.94,0.95]   (0.95,0.96]   (0.96,0.97]   (0.97,0.98]   (0.98,0.99]
## 57            13            9             12            16            20
## (0.99,1]
## 205
```

```
# print the counts by bin by matching status (L = Link, N = nonLink, P = possibleLink)
table(RlMtch_1$cutpnt,RlMtch_1$Class,useNA = "always")
```

```
##
##   L                          P     <NA>
##     [0.75,0.76]     0     0  1154  0
##     (0.76,0.77]     0     0  592   0
##     (0.77,0.78]     0     0  474   0
##     (0.78,0.79]     0     0  362   0
##   (0.79,0.8]        0     0  309   0
##   (0.8,0.81]        0     0  248   0
##     (0.81,0.82]     0     0  180   0
##     (0.82,0.83]     0     0  136   0
##     (0.83,0.84]     0     0  95    0
##     (0.84,0.85]     0     0  68    0
##     (0.85,0.86]     0     0  47    0
##     (0.86,0.87]     0     0  28    0
##     (0.87,0.88]     0     0  18    0
##     (0.88,0.89]     0     0  11    0
##   (0.89,0.9]        0     0  6     0
##   (0.9,0.91]        0     0  9     0
##     (0.91,0.92]     0     0  13    0
##     (0.92,0.93]     0     0  0     0
##     (0.93,0.94]     0     0  57    0
##     (0.94,0.95]     0     0  13    0
##     (0.95,0.96]     0     0  9     0
##     (0.96,0.97]     0     0  12    0

##   (0.97,0.98]        0      0     16     0
##   (0.98,0.99]        0      0     20     0
##   (0.99,1]           0    205      0     0
##   <NA>            8164      0      0     0
```

Histogram of counts by bin for possible matches.

```
# Plot histogram of counts by bin for possible matches

RlMtch_1 %>%
        filter(Class == "P") %>% group_by(cutpnt)
        %>% summarize(count = n()) %>%
        ggplot() +
          geom_col(aes(x = cutpnt, y = count)) +
          theme(axis.text.x = element_text(angle = 30, vjust = 0.95, hjust = 1)) + xlab("Probability match
          bin") +
          ylab("Count of possible matches")
```

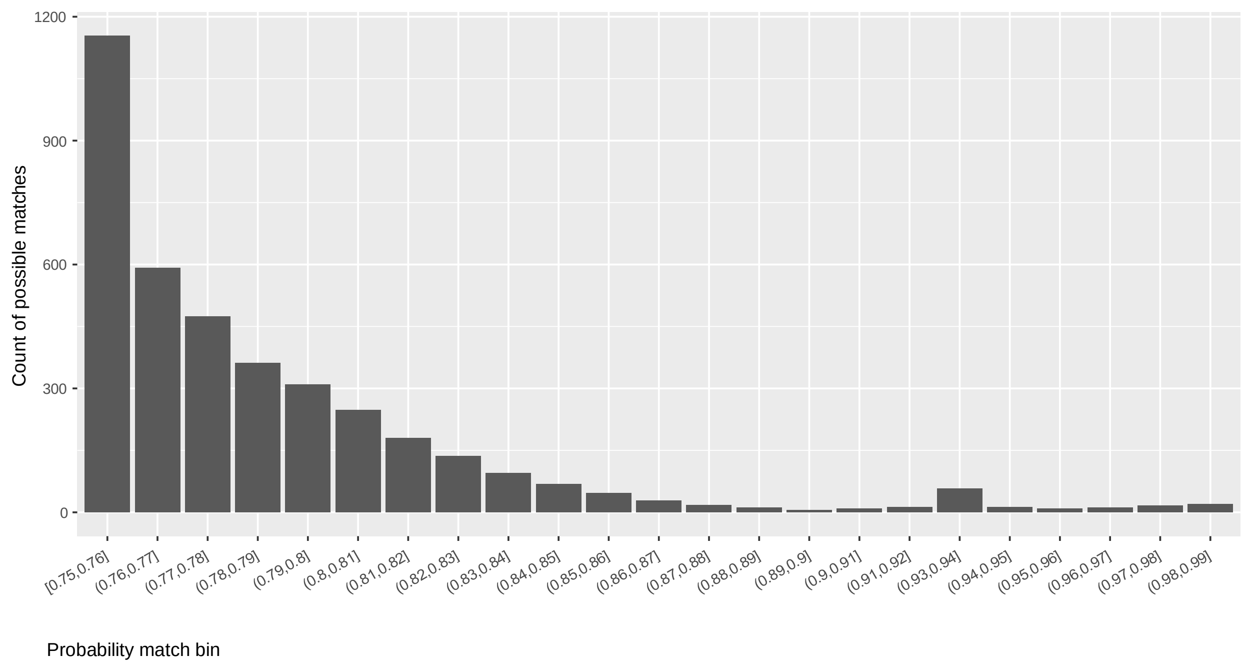


Figure 3: *Histogram of counts by bin for possible matches.*

From the histogram and table, it is easy to see how quickly the amount of manual review can balloon to where thousands of records need reconciled. For this example, we will conduct manual reviews to 0.85 and then proceed to lower match probability bins. We'll stop once we've reviewed two lower bins with no successful matches. We will then assume all lower matches are non-matches.

Depending on the goal of the linkage, the purpose of manual review may be different. A manual review may be conducted to establish the automatic acceptance level and quantify the error. It may also be a part of the normal linkage process where specificity of the linkages is critical.

There are a couple options for conducting manual review. You can use the base r edit() function, export the file as a csv or excel file conduct the review and import back in, or track and update with r code based on the row reference. The editMatch() function in the RecordLinkage package opens up the r data editor and is typically used for creating minimal training datasets.

Depending on your preference, the getPairs() function can present the pairs in a single row “getPairs(…, single.row=TRUE)” or multiple rows “getPairs(…, single.row=FALSE).” It is generally visually easier to work with the multiple rows for reviewing but does require more data wrangling once completed. This exercise will present one way of working with the multiple rows for review.

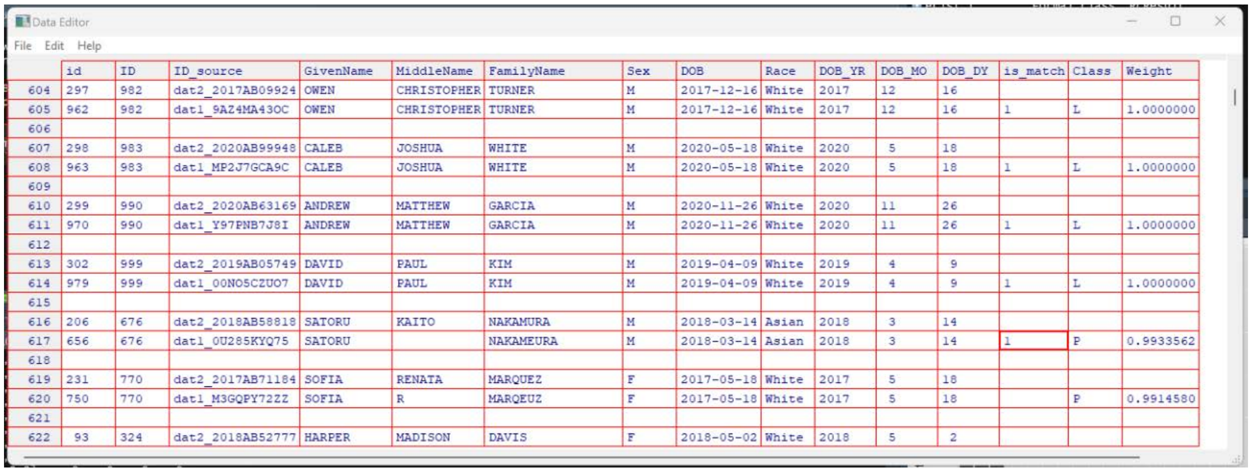
If using the edit() in R to directly modify the underlying data, save your work as a new object (especially if you will not be finishing the review in a single sitting).

We typically use the is_match field and set 1 = match, 2 = needs additional review, 3 = nonmatch, and leave blank for those not reviewed. With the multi-row view we only update the second record of the presented match set.

Below is a screen shot of the edit() indicating how we update using the basic editor. For large projects however, we save the pairs as a csv file, conduct the manual review and then load the reviewed file for finalization.

```
# Example of using the R editor.
# mlr <- edit(RlMtch_1)

# Saving a csv file and conducting manual review externally.
#  write.csv(RlMtch_1,  "ManualReviewMatch.csv",  row.names=FALSE)
```



	id	ID	ID_source	GivenName	MiddleName	FamilyName	Sex	DOB	Race	DOB_YR	DOB_MO	DOB_DY	is_match	Class	Weight
604	297	982	dat2_2017AB09924	OWEN	CHRISTOPHER	TURNER	M	2017-12-16	White	2017	12	16			
605	962	982	dat1_9AZ4MA43OC	OWEN	CHRISTOPHER	TURNER	M	2017-12-16	White	2017	12	16	1	L	1.0000000
606															
607	298	983	dat2_2020AB99948	CALEB	JOSHUA	WHITE	M	2020-05-18	White	2020	5	18			
608	963	983	dat1_MP2J7GCA9C	CALEB	JOSHUA	WHITE	M	2020-05-18	White	2020	5	18	1	L	1.0000000
609															
610	299	990	dat2_2020AB63169	ANDREW	MATTHEW	GARCIA	M	2020-11-26	White	2020	11	26			
611	970	990	dat1_Y97PNB7J8I	ANDREW	MATTHEW	GARCIA	M	2020-11-26	White	2020	11	26	1	L	1.0000000
612															
613	302	999	dat2_2019AB05749	DAVID	PAUL	KIM	M	2019-04-09	White	2019	4	9			
614	979	999	dat1_00NO5CZUO7	DAVID	PAUL	KIM	M	2019-04-09	White	2019	4	9	1	L	1.0000000
615															
616	206	676	dat2_2018AB58818	SATORU	KAITO	NAKAMURA	M	2018-03-14	Asian	2018	3	14			
617	656	676	dat1_0U285KYQ75	SATORU		NAKAMEURA	M	2018-03-14	Asian	2018	3	14	1	P	0.9933562
618															
619	231	770	dat2_2017AB71184	SOFIA	RENATA	MARQUEZ	F	2017-05-18	White	2017	5	18			
620	750	770	dat1_M3GQPY72ZZ	SOFIA	R	MARQEUZ	F	2017-05-18	White	2017	5	18		P	0.9914580
621															
622	93	324	dat2_2018AB52777	HARPER	MADISON	DAVIS	F	2018-05-02	White	2018	5	2			

Figure 4: *Image of potential matches from the r base editor.*

As a side note: make sure you develop a set of rules for your manual review that can be replicated. For the manual review in this exercise the following rules for matches were followed:

1. Character(s) change in variable(s) that do not result in a known valid spelling of the name(s).
2. A single change in date of birth year, month, or day (exception for 12/31 and 1/1 transpositions +/- 5 days).
3. A difference in sex specification with all other variables matching.
4. Character(s) change in variables(s) that result in a valid name, but a valid middle name is the same.
5. All other non-discernible potential matches reviewed by two people and classified.

Summarize Linkage

After the manual review, we can summarize the review findings.

```
# load manual reviewed data
# Rev_pairs <- read.csv("ManualReviewMatch.csv") if ran locally. # retrieve from
Github
git3 <-
'https://raw.githubusercontent.com/parrish-epi/R-recordLinkage/main/ManualReviewMatch.csv'
Rev_pairs <-read.csv(git3, fileEncoding = "ISO-8859-1")
rm(git3)
## re-run the cutpnt code to ensure we have an ordered factor
Rev_pairs$cutpnt <- with(Rev_pairs, cut(round(as.numeric(Weight),2),
                                        breaks=(seq(0.75,1, by = .01)), include.lowest =
                                        TRUE, right = TRUE), "")
# summarize links by weight bin.

addmargins(table(Rev_pairs$cutpnt,Rev_pairs$is_match))
```

```
##
##  1                3    Sum
##     [0.75,0.76]   0    0      0
##     (0.76,0.77]   0    0      0
##     (0.77,0.78]   0    0      0
##     (0.78,0.79]   0    0      0
##     (0.79,0.8]    0    0      0
##     (0.8,0.81]    0    0      0
##     (0.81,0.82]   0    0      0
##     (0.82,0.83]   0    136  136
##     (0.83,0.84]   0    95    95
##     (0.84,0.85]   1    67    68
##     (0.85,0.86]   1    46    47
##     (0.86,0.87]   1    27    28
##     (0.87,0.88]   0    18    18
##     (0.88,0.89]   0    11    11
##     (0.89,0.9]    1    5      6
##     (0.9,0.91]    1    8      9
##     (0.91,0.92]   1    12    13
##     (0.92,0.93]   0    0      0
##     (0.93,0.94]   31   26    57
##     (0.94,0.95]   7    6     13
##     (0.95,0.96]   4    5      9
##     (0.96,0.97]   8    4     12
##     (0.97,0.98]   12   4     16
##     (0.98,0.99]   20   0     20
##     (0.99,1]      205  0    205
##     Sum           293  470  763
```

Next, let’s figure out the linkage match % for each bin and then cumulatively.

```
kable(
Rev_pairs %>% filter(!is.na(is_match)) %>%
                    group_by(cutpnt,is_match)  %>%


                    summarise(Count = n()) %>% ungroup()
                    %>%
                    pivot_wider(names_from = is_match, values_from = Count, values_fill = 0) %>% mutate(total
                    =`3` + `1`) %>%
                    mutate(PropLink = `1`/total,) %>% rename(nonLink = `3`,
                    Link = `1`) %>% arrange(desc(as.numeric(row.names(.))))
                    %>% mutate(Clink = cumsum(Link),
                              Ctotal = cumsum(total)) %>%
                    mutate(cPropLink = Clink/Ctotal),
          caption = "Cumulative linkage match % by bin"
)
```

Table 4: *Cumulative linkage match % by bin*

cutpnt	nonLink	Link	total	PropLink	Clink	Ctotal	cPropLink
(0.99,1]	0	205	205	1.0000000	205	205	1.0000000
(0.98,0.99]	0	20	20	1.0000000	225	225	1.0000000
(0.97,0.98]	4	12	16	0.7500000	237	241	0.9834025
(0.96,0.97]	4	8	12	0.6666667	245	253	0.9683794
(0.95,0.96]	5	4	9	0.4444444	249	262	0.9503817
(0.94,0.95]	6	7	13	0.5384615	256	275	0.9309091
(0.93,0.94]	26	31	57	0.5438596	287	332	0.8644578
(0.91,0.92]	12	1	13	0.0769231	288	345	0.8347826
(0.9,0.91]	8	1	9	0.1111111	289	354	0.8163842
(0.89,0.9]	5	1	6	0.1666667	290	360	0.8055556
(0.88,0.89]	11	0	11	0.0000000	290	371	0.7816712
(0.87,0.88]	18	0	18	0.0000000	290	389	0.7455013
(0.86,0.87]	27	1	28	0.0357143	291	417	0.6978417
(0.85,0.86]	46	1	47	0.0212766	292	464	0.6293103
(0.84,0.85]	67	1	68	0.0147059	293	532	0.5507519
(0.83,0.84]	95	0	95	0.0000000	293	627	0.4673046
(0.82,0.83]	136	0	136	0.0000000	293	763	0.3840105

From this, we can see we detected fewer matches at the lower match probabilities. If we were going to be developing an automatic acceptance threshold, this would be much more informative than relying only on the ParetoThreshold calculated previously. For example, if I decided to set a threshold at >0.95, 95% of the record pairs in this range would be expected to be true matches.

To finish this off and calculate our linkage rates for dat2 (our sample), which is the proportion that success- fully matched with the dat1 (population source), we need to merge our linked data to the full data. To do this we need to do a little bit of data wrangling.

```
# remove blank rows
Matched_pairs  <-  Rev_pairs  %>%  filter(!is.na(id))
# split into two dataframes
MP_1 <- Matched_pairs %>% filter(as.numeric(row.names(.)) %% 2!=0) MP_2 <-
Matched_pairs %>% filter(as.numeric(row.names(.)) %% 2==0) ## verify that they have
the same number of records AND DO NOT sort nrow(MP_1)
```

```
## [1] 4082
```

```
nrow(MP_2)
```

```
## [1] 4082
```

```
## combine the ID's and include the matching criteria contained in MP_2
MatchedPairsFinal <- cbind(MP_2[,c(3,13:16)], MP_1[,3])

# First Remove blank rows
Matched_pairs  <-  Rev_pairs  %>%  filter(!is.na(id))

# Add a row index to keep track of the pairs
Matched_pairs <- Matched_pairs %>% mutate(row_index = row_number())

# Separate the data into two data frames
MP_1 <- Matched_pairs %>% filter(row_index %% 2 != 0) MP_2 <-
Matched_pairs %>% filter(row_index %% 2 == 0)

# Verify that they have the same number of records AND DO NOT sort
if (nrow(MP_1) == nrow(MP_2)) {
  # Combine the ID's and include the matching criteria contained in MP_2
  MatchedPairsFinal <- MP_2 %>%
    select(ID_source, is_match, Class, Weight, cutpnt) %>% rename(ID_source1 =
    "ID_source") %>%
    bind_cols(MP_1  %>%
                        select(ID_source) %>% rename(ID_source2 =
                        "ID_source")
                   ) %>%
    filter(is_match == 1)
} else {
  stop("The two data frames do not have the same number of records")
}
```

Now that we have the data in a nice format, before we merge back to our original sample (dat2), we need to check if we created any duplicates. This can be done at different times in the linkage process, but for simplicity, we’re just doing it once here.

Check for duplicated records

```
table(duplicated(MatchedPairsFinal$ID_source1))
```

```
##
## FALSE
##
     29
```

3

This is great; no duplicates. This means we didn't link an individual twice. If we had, a simple strategy would be to retain the match with the highest probability match score.

Let's now merge the linked data back to dat2 (our sample) and calculate the linkage rate.

```
FinalMatchedData  <-  merge(dat2,  MatchedPairsFinal,
                            by.x = "ID_source",
                            by.y = "ID_source2",

                            all.x = TRUE)

# create a summary table of the linked data
SumLinkT <-
FinalMatchedData %>%
    summarise(
      Linked = sum(!is.na(ID_source1)), nonLinked
      = sum(is.na(ID_source1))
  ) %>%
  pivot_longer(cols = everything(), names_to = "value", values_to = "count") %>% mutate(
    proportion = count / sum(count)
  )
as.data.frame(SumLinkT)
```

```
##          value count proportion
## 1       Linked   293 0.96381579
## 2 nonLinked       11 0.03618421
```

With manual review based on our probability scores we returned a linkage 'rate' of 96.4% compared to 67.9% using a deterministic exact match. If we were feeling particularly ambitious, we could inspect the remaining nonlinked records or conduct a second round of linkages with the remaining records using a different, relaxed set of criteria.

Conclusion

In this example, we demonstrated the use of the RecordLinkage() package in R to perform record linkage a crucial process for identifying and merging duplicate records across data sets. We started by loading and pre-processing the data, ensuring it was clean and standardized. We then created a record linkage object, specifying the fields for comparison and string comparator to enhance efficiency. By computing comparison weights and classifying the matches, we were able to identify and review potential matches between records from different data sets. This example serves as a basic approach, showcasing the essential steps and functions of the RecordLinkage package, and can be extended to more complex scenarios and data sets as needed.

Jared Parrish, PhD
Parrish Analytics and Epidemiology Consulting
Email: parrish.epi@gmail.com
URL: https://github.com/parrish-epi/R-recordLinkage