SHINJINI SEN-21BCI0039
VINEET SINGH- 21BCI0130

DATA PRIVACY- BCSE318L

A Comprehensive Report on Record Linkage

"How can we efficiently merge and deduplicate datasets containing similar but not identical records?"

"In the vast ocean of data, each record is a drop, seemingly insignificant on its own. Yet, when linked together with precision and purpose, they form the intricate web of knowledge, revealing patterns and connections that were previously hidden. Much like stars in the night sky, the art of record linkage illuminates the path to understanding."

NAME- SHINJINI SEN
REGISTRATION NUMBER- 21BCI0039
NAME- VINEET SINGH
REGISTRATION NUMBER- 21BCI0130

COURSE NAME- DATA PRIVACY
COURSE CODE- BCSE318L
CLASS NUMBER- VL2023240500995
PROFESSOR- GEETHA MARY A

INTRODUCTION:

Record linkage, also known as data matching or entity resolution, is the process of identifying and merging duplicate or related records from multiple datasets. It is a crucial step in data integration and cleaning, particularly when dealing with large and heterogeneous datasets from different sources.
In record linkage, records are compared based on common identifiers or similarity measures, such as names, addresses, or other attributes. The goal is to accurately link records that refer to the same entity, despite variations or errors in the data.

The process of record linkage typically involves several steps:

- Preprocessing: Data preprocessing steps may include standardization of data formats, cleaning to remove inconsistencies or errors, and feature extraction to identify relevant attributes for comparison.
- Blocking: Blocking is a technique used to reduce the computational complexity of record linkage by partitioning the datasets into smaller subsets based on certain criteria, such as common attributes or proximity in a feature space.
- Comparison: Records within each block are compared pairwise to determine their similarity or likelihood of being a match. Various similarity measures, such as edit distance or string similarity, can be used depending on the data and application.
- Classification: Based on the comparison results, records are classified as matches or non-matches using classification algorithms or threshold-based rules.
- Clustering and Merging: Matched records are clustered together to form groups representing unique entities. Duplicate records within each cluster are merged or consolidated into a single, clean representation.

ABSTRACT:

Record linkage, also known as data matching or entity resolution, is a critical process in data management aimed at identifying and merging duplicate or related records from disparate data sources. In this report, we present an overview of record linkage techniques and discuss the usage of the Python-based Record Linkage Toolkit (Record Linkage) for performing record linkage tasks.

The report begins by outlining the challenges associated with managing large datasets containing duplicate or similar records. It highlights the importance of data quality and consistency in various domains, including healthcare, finance, and marketing, and emphasizes the need for efficient record linkage tools to address these challenges.

We introduce the concept of record linkage and discuss different types of record linkage methods, such as deterministic and probabilistic matching algorithms. We delve into the process of data preprocessing, feature selection, and similarity calculation involved in record linkage operations.

The main focus of the report is on the practical usage of the Record Linkage toolkit. We provide step-by-step instructions on how to install and use the toolkit, along with real-world examples and case studies demonstrating its effectiveness in data integration and deduplication tasks.

Furthermore, we discuss the key features of the Record Linkage toolkit, including its support for various record linkage methods, customization options, and performance optimization techniques. We also explore the integration of the toolkit with other Python libraries for data analysis and visualization.

Overall, this report serves as a comprehensive guide to record linkage techniques and the practical application of the Record Linkage toolkit. It aims to equip data professionals and researchers with the knowledge and tools necessary to effectively manage and integrate large datasets while ensuring data accuracy and consistency.

KEYWORDS:

- Record Linkage
- Data Matching
- Entity Resolution
- Duplicate Records
- Data Deduplication
- Data Integration
- Python Toolkit
- Record Linkage
- Data Quality
- Data Consistency

Why do we need Record Linkage?

Record linkage is essential in various real-life scenarios where data from different sources need to be integrated or matched.

Data Integration: Different organizations often maintain separate databases containing related information. Record linkage helps integrate these databases to create a more comprehensive view of the data.

In the healthcare sector, integrating patient records from multiple hospitals and healthcare providers can provide a complete medical history for each patient, leading to better patient care and outcomes. For instance, in the case of a medical emergency where a patient is unable to provide their medical history, having access to integrated records from various healthcare providers can be life-saving.

Duplicate Detection: Duplicate records in databases can lead to inefficiencies and errors in data analysis. Record linkage helps identify and remove duplicate entries.
Example: In the financial sector, detecting duplicate accounts or fraudulent activities is crucial for maintaining the integrity of financial systems.
In 2018, Wells Fargo faced scrutiny for opening unauthorized accounts on behalf of customers. Record linkage techniques could have helped identify and prevent such unauthorized activities by detecting duplicate accounts.

Data Quality Improvement: Record linkage can help improve the quality of data by identifying inconsistencies, errors, or missing information.

In government agencies, integrating data from different departments can help identify individuals or entities involved in illegal activities. In 2019, the U.S. Department of Justice indicted dozens of wealthy parents in the college admissions bribery scandal. Integrating data from various sources, including education records, financial transactions, and communication logs, played a crucial role in uncovering the scheme and prosecuting those involved. By linking disparate data sources, authorities were able to piece together the evidence needed to build a case against the perpetrators.

Factors Contributing to the Discovery of Record Linkage

The discovery and evolution of record linkage as a crucial process for integrating and analyzing data from disparate sources have been influenced by various factors. Firstly, the exponential growth in data volume and variety due to technological advancements has necessitated methods for consolidating and analyzing disparate datasets. Additionally, the fragmentation of data, wherein information is stored in different systems and formats, has posed challenges in accessing and integrating data across organizational boundaries. The rise of big data has further accentuated these challenges, driving the exploration of innovative approaches for data integration and analysis. Moreover, the demand for comprehensive insights from businesses and research institutions has fueled the need to combine data from various sources to make informed decisions and gain competitive advantages. Concurrently, researchers and practitioners in database management, statistics, and information retrieval have recognized the importance of developing techniques to link and analyze data from heterogeneous sources. Government agencies involved in census and demographic studies have also sought methods to merge and analyze data from different surveys and administrative records to produce accurate and reliable statistics. Lastly, advancements in computer science, particularly in algorithms, data structures, and computational techniques, have played a crucial role in enabling the development of efficient record linkage methods. Together, these factors have contributed to the discovery and evolution of record linkage as an essential component of modern data analysis and decision-making processes.

Challenges in record linkage

The main challenge in record linkage is to establish whether records from different sources concern the same person. If there is no unique identifier across the data sources1), a set of variables (or fields/attributes) that exist in all records can be used to assist in the decision process. The variables used for linking can be referred to as linkage variables, while the set of all these variables together is called a linkage key: every variable provides a piece of information, and together they form certain information about a specific person (or subject, or entity in a more general sense)

Deterministic record linkage

This method is used when datasets contain one or more attributes that uniquely identify the records. This way, if two records have the same unique attribute, they can be said to be a match and classified as the same entity. If datasets have multiple attributes that uniquely identify a record, then comparisons can be performed based on all these columns. Records can be considered a match if they match on a single attribute or any set threshold value. Data attributes such as social security number and national ID are good examples of uniquely identifying attributes which can be used for deterministic record linkage.

We can choose this method of record linkage when you have high quality data and simple SQL queries can help you make the matching decision. There are two ways to evaluate agreement between linkage variables:

Exact matching: Agreement or disagreement is determined by directly observing whether the values of every linkage variable are exactly the same. Generally, this can be done in two ways: fully matching or partial matching. Fully matching uses the complete or full value of the linkage variables; for example, matching on the full surname, the complete date of birth, the complete address. Partial matching, on the other hand, uses only a partial value of the linkage variables, such as a substring of the first four characters of the surname.

Similar matching: While exact matching compares the value directly, similar matching compares the value in a less stringent way. It makes use of a number of criteria to judge whether two different values can be considered similar, i.e. whether their difference is still within an acceptable margin.

In the realm of healthcare data integration, the utilization of deterministic record linkage methods is pivotal for establishing robust linkages between registries and claims databases. While the concept of a definitive gold standard for such linkages remains contentious, the iterative deterministic approach championed by the National Cancer Institute (NCI) for the creation of the SEER-Medicare linked dataset stands as a testament to its efficacy, validity, and reliability across multiple dataset updates.

The NCI's algorithmic framework comprises a series of deterministic matching iterations, each employing distinct match criteria to progressively refine linkage accuracy.

For instance, in the initial step of the linkage process, two records are deemed to match if they share a Social Security Number (SSN) and meet one of the following criteria:
- Exact or fuzzy match on first and last name, accommodating variations such as nicknames
- Consistency in last name, month of birth, and sex
- Consistency in first name, month of birth, and sex

In scenarios where SSN information is absent or incongruent, or initial match criteria are not met, a secondary round of deterministic linkages is initiated. Here, records are considered a match if they align on last name, first name, month of birth, sex, and fulfill additional criteria such as:
- Agreement on seven to eight digits of the SSN
- Consistency in two or more of the following: year of birth, day of birth, middle initial, or date of death

Moreover, in instances where complete or partial identifiers are available but restricted from release, deterministic linkage based on encrypted identifiers may be employed. Pioneering work by Quantin and collaborators has elucidated procedures for encrypting identifiers using cryptographic hash functions like the Secure Hash Algorithm version 2 (SHA-2). These functions ensure that identifiers required for linkage can be shared with researchers while safeguarding patient confidentiality. The deterministic nature of cryptographic hashing, coupled with its irreversible transformation of input data into a unique output, renders it a cornerstone of security protocols. Quantin's research underscores the feasibility of linking records via deterministic algorithms utilizing encrypted identifiers.

However, it's imperative to acknowledge that record pairs matched on encrypted identifiers preclude manual review or validation, underscoring the trade-offs inherent in privacy-preserving linkage methodologies.

Probabilistic record linkage

As opposed to the deterministic method, in the probabilistic method each linkage variable has a certain weight. These weights are determined by the discriminative power and possible errors.

The overall weight of the linkage variables is used to decide whether or not a corresponding record pair can be linked. Users of probabilistic methods must take into consideration the following aspects of m and u estimation, weight assignment, and the choice of cut-off value. Estimation on m and u.

The complete data log-likelihood takes all record pairs into account. Because the number of non-links is very dominant, there will be bias in the estimation of m and p. To correct for this, the data log-likelihood should be adjusted to obtain sensible m and p estimates. There are a number of ways to obtain m and u:

- Using prior information on the probability distribution of the linkage variables as well as the probabilities of different type of error resulting from the record generation process). For example, one can calculate m as equal to one minus the error rate of the identifier, if this is known.

- Using standard estimation methods, such as expectation maximization (EM) algorithm and maximum likelihood estimation (MLE), with some adjustment. Thus, instead of using all record pairs, only the frequency of the patterns will be used.

- Using a fuzzy algorithm. For example, by observing the number of agreements, disagreements, and no-decisions (when at least one value is missing) on each linkage variable, for each pair of records selected by a series of random sampling (with replacement) and pairing them as a Cartesian product. The average value of these numbers is used to estimate m (in this case m refers to the reliability of the linkage variables) and u (the probability of matching by chance).

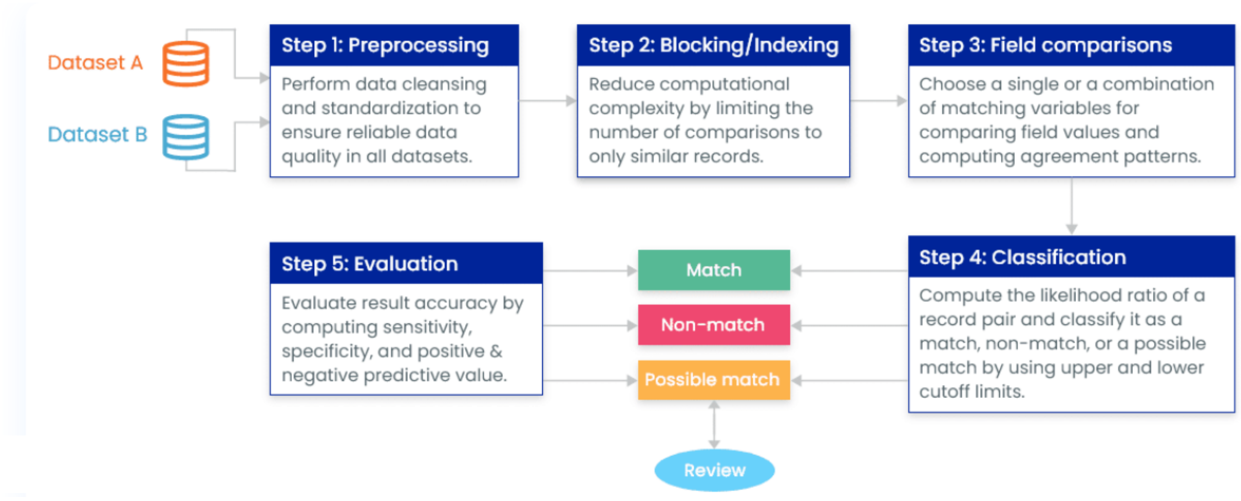
A practical example of probabilistic record linkage involving two datasets: voter registration records and census data.

1. **Data Preprocessing**:
    - Voter registration records may contain variations in names and addresses due to manual entry errors or differences in data collection processes. Preprocessing involves standardizing names (e.g., converting "Robert" to "Rob") and addresses (e.g., abbreviating street names) to enhance comparability.
    - Similarly, census data may have inconsistencies in formatting or encoding, necessitating preprocessing steps to ensure uniformity across datasets.
2. **Blocking / Indexing**:
    - Both datasets are partitioned into blocks based on shared attributes such as postal codes or birth years. For instance, individuals residing in the same postal

code area or born in the same year may be grouped together.
- By limiting comparisons to records within the same block, computational resources are optimized, as only potentially relevant records are compared.

3. **Field Matching**:
    - Records from voter registration and census datasets are compared based on multiple attributes, including name, date of birth, address, and electoral district.
    - For example, a record with the name "John Smith," born on January 1, 1980, residing at "123 Main Street," and registered in the "Central District," is compared with similar records in the census dataset.
4. **Similarity Computation**:
    - Similarity metrics such as Jaccard similarity or edit distance are applied to quantify the resemblance between field values. These metrics measure how closely two values match.
    - Using edit distance, the similarity between "John Smith" and "Jon Smyth" might be calculated, considering the number of character substitutions required to transform one into the other.
5. **Classification**:
    - Probabilistic classification algorithms, such as Expectation Maximization or Fellegi-Sunter, assign probabilities to record pairs being a match or non-match.
    - Based on the computed similarities and probability distributions, each record pair is assigned a likelihood of being a match.
6. **Threshold Determination**:
    - Thresholds are set to classify record pairs as matches or non-matches based on their assigned probabilities. These thresholds are determined empirically through validation experiments or domain knowledge.
    - For instance, pairs with probabilities above a certain threshold may be classified as matches, while those below the threshold are considered non-matches.
7. **Post-Matching Evaluation**:
    - The accuracy of the probabilistic approach is evaluated using metrics such as precision, recall, and F1-score. This assessment helps gauge the method's effectiveness and identify areas for improvement in future iterations.

Steps involved in Record Linkage process-


Dataset A
Dataset B
Step 1: Preprocessing
Perform data cleansing and standardization to ensure reliable data quality in all datasets.
Step 2: Blocking/Indexing
Reduce computational complexity by limiting the number of comparisons to only similar records.
Step 3: Field comparisons
Choose a single or a combination of matching variables for comparing field values and computing agreement patterns.
Step 4: Classification
Compute the likelihood ratio of a record pair and classify it as a match, non-match, or a possible match by using upper and lower cutoff limits.
Match
Non-match
Possible match
Review
Step 5: Evaluation
Evaluate result accuracy by computing sensitivity, specificity, and positive & negative predictive value.

Step 01: Data Preparation

At the outset, the process involves meticulous data refinement endeavors aimed at augmenting data quality. This encompasses an array of activities such as rectifying discrepancies such as missing or erroneous data points, expunging irrelevant or outdated information, and ensuring uniformity across disparate datasets.

Step 02: Blocking / Indexing

Efficient record linkage hinges on the strategic application of blocking and indexing methodologies to alleviate the computational burden. These stratagems are instrumental in curtailing the number of comparisons by selectively targeting records with discernible similarities. By virtue of blocking dissimilar records, the computational overhead is notably mitigated.

Step 03: Field Matching

The crux of this phase lies in the judicious selection of data fields for comparison, typically predicated on unique identifiers. In scenarios bereft of such identifiers, amalgamations of fields are harnessed for probabilistic matching endeavors Employing similarity metrics such as Jaro Winkler or Levenshtein facilitates the evaluation of data congruity. Subsequently, scrutiny of agreement patterns ensues to ascertain the congruence of records pertaining to identical entities.

Step 04: Classification

The Fellegi and Sunter framework serves as the basis for many record linkage classifiers, such as Naive Bayes, Support Vector Machine (SVM), and Expectation Conditional Maximization, etc. According to this framework, the agreement likelihood ratio (R(γj)) that an algorithm will correctly classify a pair (*j*) of true matches as matches and would not incorrectly classify them as a nonmatch is: m/u. Mathematically: *R(γj) = m/u*, where:

1. The *m-probability* is the conditional probability that a record pair *j* has an agreement pattern *γj* given that it is a match (*M*), denoted as *m = P(γj| M),*
2. The *u-probability* is the conditional probability that a record pair j has an agreement pattern *γj* given that it is not a match (*NM*), denoted as *u = P(γj| NM).*

Placing the values of m and u in above equation:

R(γj) = P(γj|M)/P(γj|NM) —– (i)

The likelihood ratio from the above equation gives a numerical value that represents how likely two records belong to the same entity. But for you to classify the records as a match or nonmatch, you need to run record linkage on a test dataset and compute a series of likelihood ratios. This data will allow you to determine the upper cutoff (*W*+) and a lower cutoff (*W*−) for classifying records as matches and nonmatches correctly.

These cutoff values highly depend on the type of data your datasets contain. And so, once you have estimated the cutoff values for your dataset, you can first calculate the likelihood ratio of a pair using the above formula and then easily map record pairs to three possible outcomes:

1. The pair is a match if likelihood ratio *R(γj)* is greater than or equal to upper cutoff *W+*
2. The pair is a nonmatch if likelihood ratio *R(γj)* is less than or equal to lower cutoff *W-*
3. The pair belongs to the undecided cases for manual clerical reviews, if otherwise

Let's expand equation *(i)* to accommodate multiple matching variables, and take log on both sides to reduce skew in the data:

$$\log\left[R\left(\gamma^j\right)\right] = \log\left(\frac{P\left(\gamma_1^j|M\right)}{P\left(\gamma_1^j|NM\right)}\right) + \log\left(\frac{P\left(\gamma_2^j|M\right)}{P\left(\gamma_2^j|NM\right)}\right) + \cdots + \log\left(\frac{P\left(\gamma_k^j|M\right)}{P\left(\gamma_k^j|NM\right)}\right)$$

Step 05: Evaluation

While record linkage can significantly enhance business intelligence processes, the potential inaccuracies inherent in linkage algorithms underscore the importance of method and attribute selection to maximize accuracy. The outcomes of a record linkage decision fall into four categories:

1. Correctly identifying true matches as matches (A)
2. Erroneously identifying true matches as nonmatches (B)
3. Correctly identifying true nonmatches as nonmatches (C)
4. Erroneously identifying true nonmatches as matches (D)

Evaluating the efficacy of a record linkage method involves assessing four key metrics:

1. Sensitivity, which gauges the algorithm's accuracy in identifying true matches as matches (A/(A+B)).
2. Specificity, which measures the algorithm's accuracy in identifying true nonmatches as nonmatches (C/(C+D)).
3. Positive Predictive Value (PPV), indicating the proportion of nonmatched pairs correctly identified as matches (A/(A+C)).
4. Negative Predictive Value (NPV), indicating the proportion of matched pairs correctly identified as nonmatches (D/(B+D)).

Implementation on Dataset-1

The submodule recordlinkage.datasets contains several datasets that can be used for testing. For this example, we use the Febrl dataset 1. This dataset contains 1000 records of which 500 original and 500 duplicates, with exactly one duplicate per original record. This dataset can be loaded with the function load_febrl1.

Link to the Google Collab py notebook-
https://colab.research.google.com/drive/1Xu9ob0TLQGSyBOpHIriTSDFAbulO1Cqg?usp=sharing

Step 1- Importing necessary packages

```
In [ ]: import numpy

        import pandas
```

Step 2- Install Record linkage package

```
In [ ]: pip install recordlinkage
```

```
Collecting recordlinkage
  Downloading recordlinkage-0.16-py3-none-any.whl (926 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 926.9/926.9 kB 5.1 MB/s eta 0:00:00
Collecting jellyfish>=1 (from recordlinkage)
  Downloading jellyfish-1.0.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.2/1.2 MB 23.1 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.13 in /usr/local/lib/python3.10/dist-packages (from recordlinkage) (1.25.2)
Requirement already satisfied: pandas<3,>=1 in /usr/local/lib/python3.10/dist-packages (from recordlinkage) (2.0.3)
Requirement already satisfied: scipy>=1 in /usr/local/lib/python3.10/dist-packages (from recordlinkage) (1.11.4)
Requirement already satisfied: scikit-learn>=1 in /usr/local/lib/python3.10/dist-packages (from recordlinkage) (1.2.2)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from recordlinkage) (1.4.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas<3,>=1->r
ecordlinkage) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas<3,>=1->recordlinka
ge) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas<3,>=1->recordlin
kage) (2024.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1->
recordlinkage) (3.4.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->panda
s<3,>=1->recordlinkage) (1.16.0)
Installing collected packages: jellyfish, recordlinkage
Successfully installed jellyfish-1.0.3 recordlinkage-0.16
```

Step 3- Load the dataset from a CSV file

```
In [ ]: import recordlinkage
        from recordlinkage.datasets import load_febrl1
```

```
In [ ]: dfA = load_febrl1()
        dfA
```

The dataset head from the CSV file is displayed as-

Out[ ]:

	given_name	surname	street_number	address_1	address_2	suburb	postcode	state	date_of_birth	soc_sec_id
rec_id										
rec-223-org	NaN	waller	6	tullaroop street	willaroo	st james	4011	wa	19081209	6988048
rec-122-org	lachlan	berry	69	giblin street	killarney	bittern	4814	qld	19990219	7364009
rec-373-org	deakin	sondergeld	48	goldfinch circuit	kooltuo	canterbury	2776	vic	19600210	2635962
rec-10-dup-0	kayla	harrington	NaN	maltby circuit	coaling	coolaroo	3465	nsw	19150612	9004242
rec-227-org	luke	purdon	23	ramsay place	mirani	garbutt	2260	vic	19831024	8099933
...	...	...	...	...	...	...	...	...	...	...
rec-188-dup-0	stephanie	geu	28	bainton crescent	masonic memorial village	maryborough	2541	sa	19421008	3997529
rec-334-dup-0	nicholas	NaN	289	britten-jonues drive	jabaru court	paddington	2000	vic	19970422	5062738
rec-469-dup-0	lachlan	katsiavos	29	paul coe cdrescent	NaN	casual	2913	nsw	19380406	4112327
rec-350-dup-0	monique	gergely	21	harwoos court	hyberni a park	sherwood	2207	nsw	19790807	7375144
rec-212-org	NaN	mcveigh	45	bougainville street	kimberley	ourimbah	6060	wa	19360219	8243761

1000 rows × 10 columns

It is very intuitive to start with comparing each record in DataFrame dfA with all other records in DataFrame dfA. In fact, we want to make record pairs. Each record pair should contain two different records of DataFrame dfA. This process of making record pairs is also called "indexing". With the recordlinkage module, indexing is easy. First, load the recordlinkage.Index class and call the .full method. This object generates a full index on a .index(...) call. In case of deduplication of a single dataframe, one dataframe is sufficient as input argument.

In [ ]:

```
indexer = recordlinkage.Index()
indexer.full()
candidate_links = indexer.index(dfA)
```

```
WARNING:recordlinkage:indexing - performance warning - A full index can result in large number of record pairs.
WARNING:recordlinkage:indexing - performance warning - A full index can result in large number of record pairs.
```

In [ ]:

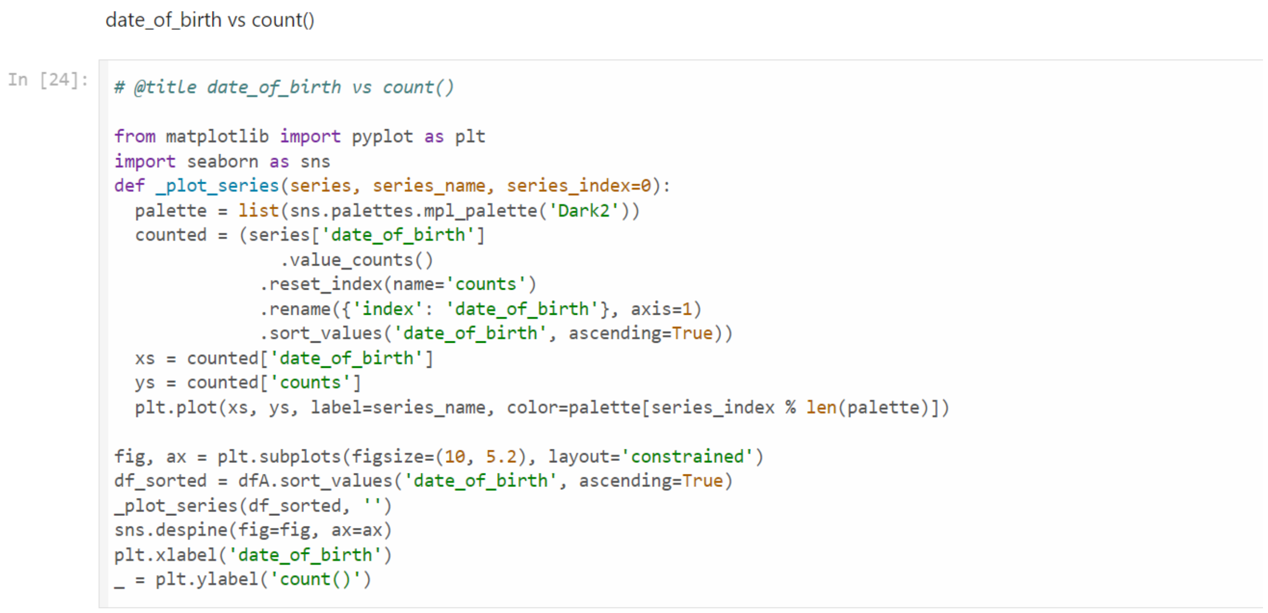
```
print(len(dfA), len(candidate_links))
# (1000*1000-1000)/2 = 499500
```

```
1000 499500
```

Step 4- Identification of potential matches

- Configures the indexer to perform a full indexing. In record linkage, full indexing means that every record in the dataset is compared with every other record.
- Generates candidate links based on the indexing method specified earlier. It applies the indexing method to the DataFrame dfA to identify potential matches between records
- Prints the number of records in dfA and the number of candidate links generated by the indexing process.

Step 5- Plotting Date_of_Birth vs count graph:



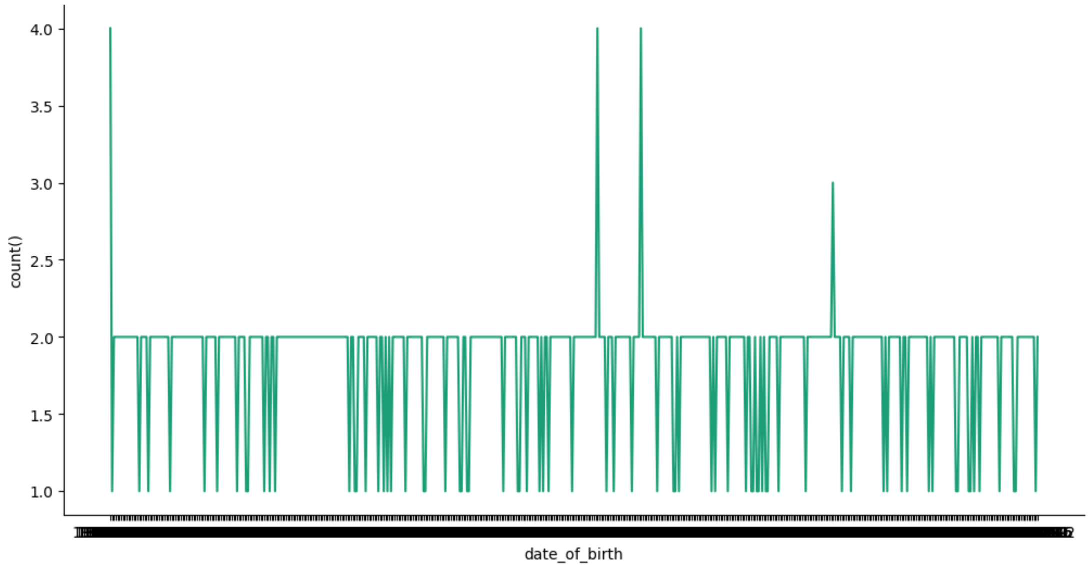
date_of_birth vs count()

In [24]:

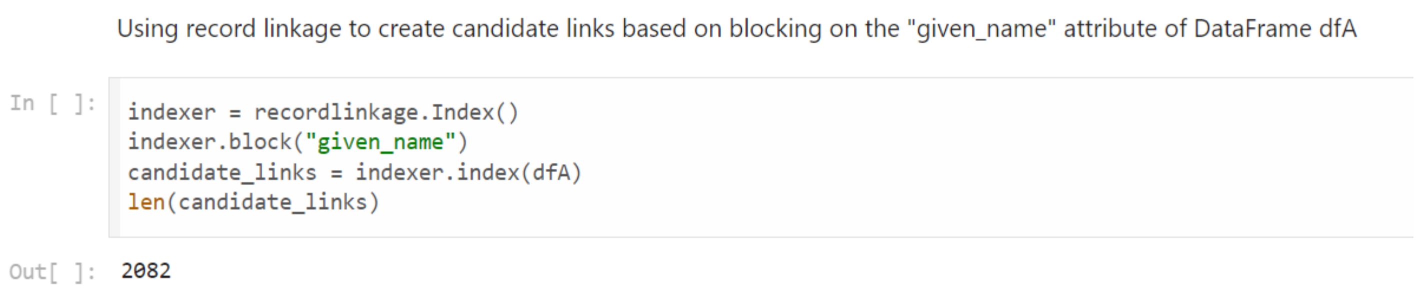
```
# @title date_of_birth vs count()

from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  palette = list(sns.palettes.mpl_palette('Dark2'))
  counted = (series['date_of_birth']
                .value_counts()
              .reset_index(name='counts')
              .rename({'index': 'date_of_birth'}, axis=1)
              .sort_values('date_of_birth', ascending=True))
  xs = counted['date_of_birth']
  ys = counted['counts']
  plt.plot(xs, ys, label=series_name, color=palette[series_index % len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = dfA.sort_values('date_of_birth', ascending=True)
_plot_series(df_sorted, '')
sns.despine(fig=fig, ax=ax)
plt.xlabel('date_of_birth')
_ = plt.ylabel('count()')
```



Step 6- Identify the number of links

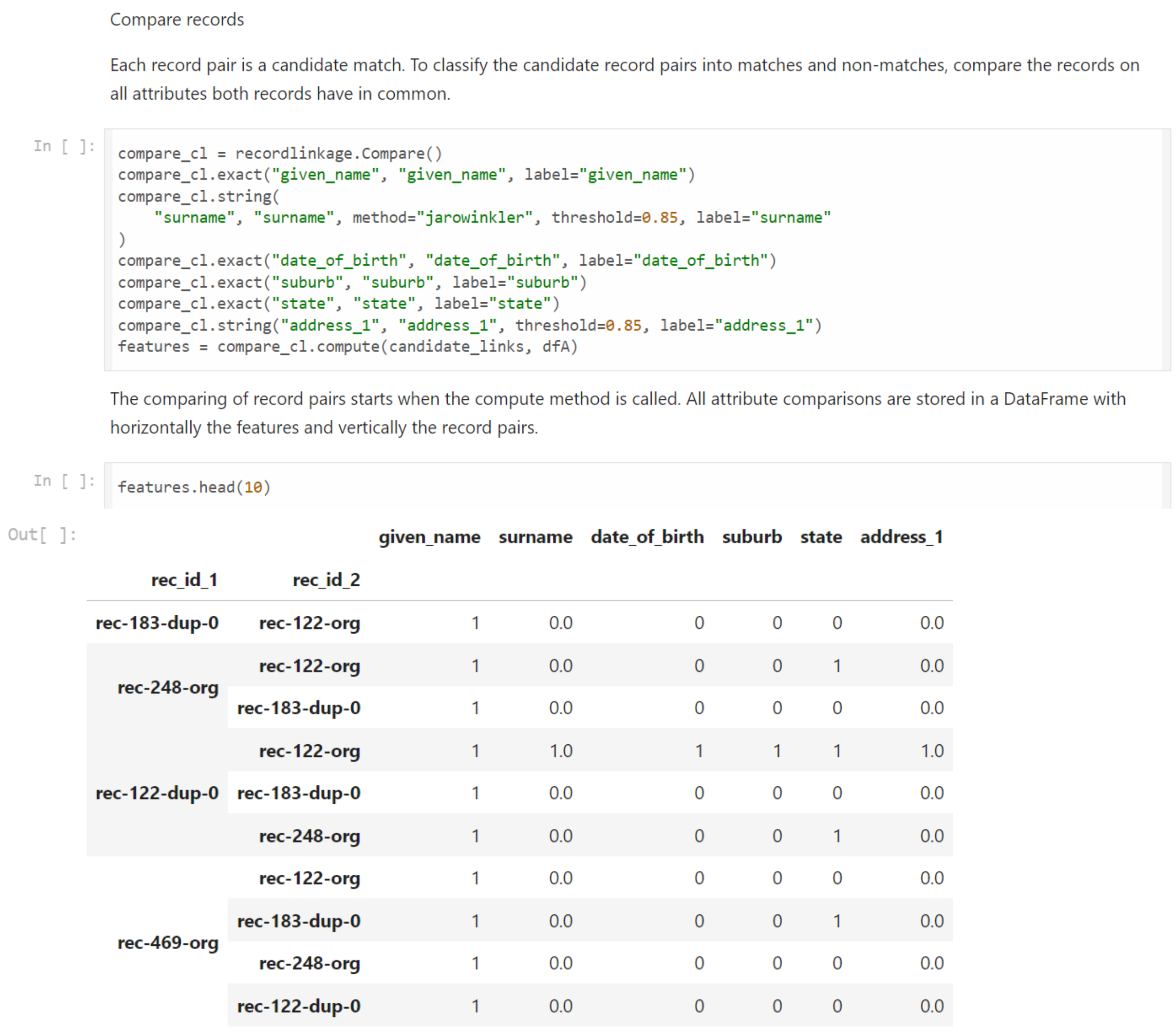


Using record linkage to create candidate links based on blocking on the "given_name" attribute of DataFrame dfA

```
In [ ]: indexer = recordlinkage.Index()
        indexer.block("given_name")
        candidate_links = indexer.index(dfA)
        len(candidate_links)

Out[ ]: 2082
```

Step 7- Comparing Records



Compare records

Each record pair is a candidate match. To classify the candidate record pairs into matches and non-matches, compare the records on all attributes both records have in common.

```
In [ ]: compare_cl = recordlinkage.Compare()
        compare_cl.exact("given_name", "given_name", label="given_name")
        compare_cl.string(
            "surname", "surname", method="jarowinkler", threshold=0.85, label="surname"
        )
        compare_cl.exact("date_of_birth", "date_of_birth", label="date_of_birth")
        compare_cl.exact("suburb", "suburb", label="suburb")
        compare_cl.exact("state", "state", label="state")
        compare_cl.string("address_1", "address_1", threshold=0.85, label="address_1")
        features = compare_cl.compute(candidate_links, dfA)
```

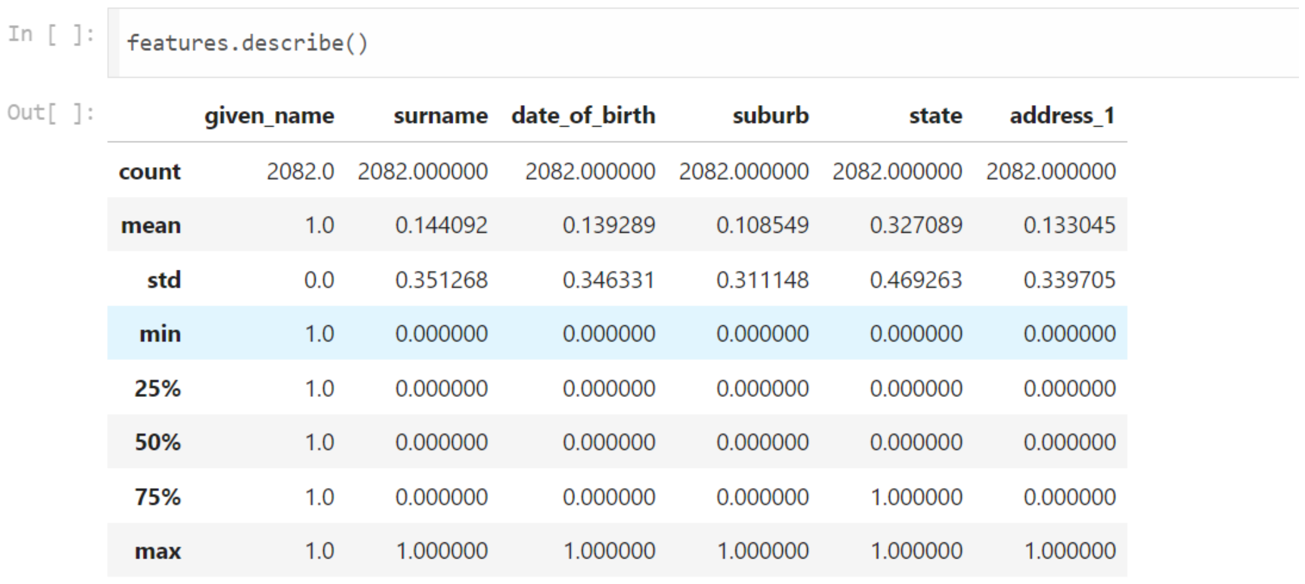
The comparing of record pairs starts when the compute method is called. All attribute comparisons are stored in a DataFrame with horizontally the features and vertically the record pairs.

```
In [ ]: features.head(10)
```

Out[ ]:

		given_name	surname	date_of_birth	suburb	state	address_1
rec_id_1	**rec_id_2**						
rec-183-dup-0	**rec-122-org**	1	0.0	0	0	0	0.0
rec-248-org	**rec-122-org**	1	0.0	0	0	1	0.0
	rec-183-dup-0	1	0.0	0	0	0	0.0
rec-122-dup-0	**rec-122-org**	1	1.0	1	1	1	1.0
	rec-183-dup-0	1	0.0	0	0	0	0.0
	rec-248-org	1	0.0	0	0	1	0.0
rec-469-org	**rec-122-org**	1	0.0	0	0	0	0.0
	rec-183-dup-0	1	0.0	0	0	1	0.0
	rec-248-org	1	0.0	0	0	0	0.0
	rec-122-dup-0	1	0.0	0	0	0	0.0

Step 8- Describing the types of features the records show

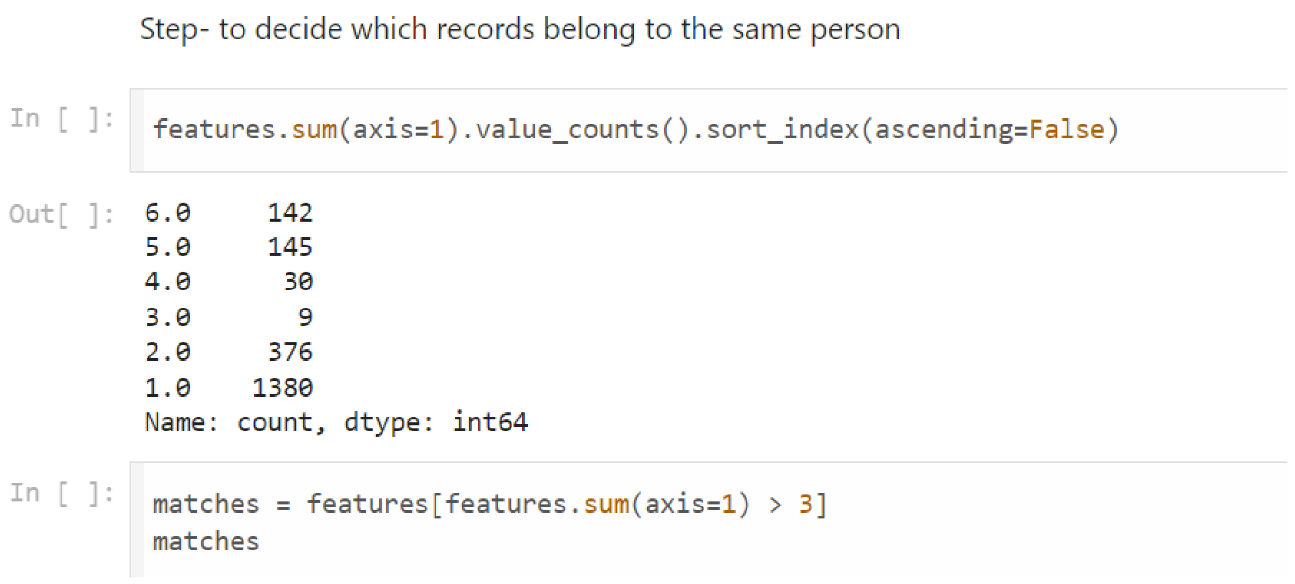


```
In [ ]: features.describe()
```

Out[ ]:

	given_name	surname	date_of_birth	suburb	state	address_1
count	2082.0	2082.000000	2082.000000	2082.000000	2082.000000	2082.000000
mean	1.0	0.144092	0.139289	0.108549	0.327089	0.133045
std	0.0	0.351268	0.346331	0.311148	0.469263	0.339705
min	1.0	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.0	0.000000	0.000000	0.000000	0.000000	0.000000
50%	1.0	0.000000	0.000000	0.000000	0.000000	0.000000
75%	1.0	0.000000	0.000000	0.000000	1.000000	0.000000
max	1.0	1.000000	1.000000	1.000000	1.000000	1.000000

Step 9- Deciding which records belong to the same person



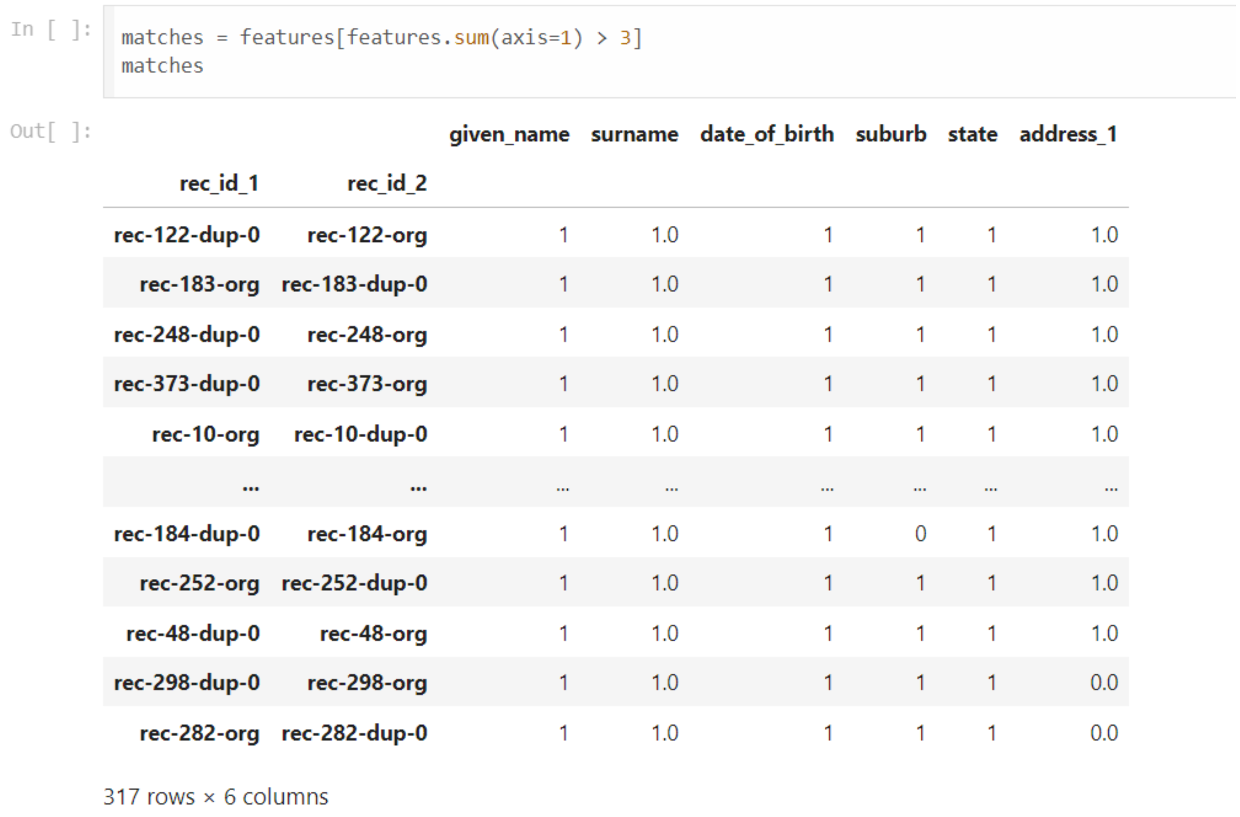
Step- to decide which records belong to the same person

```
In [ ]: features.sum(axis=1).value_counts().sort_index(ascending=False)

Out[ ]: 6.0     142
        5.0     145
        4.0      30
        3.0       9
        2.0     376
        1.0    1380
        Name: count, dtype: int64

In [ ]: matches = features[features.sum(axis=1) > 3]
        matches
```

Step 10- Filtering the Data frame



```
In [ ]: matches = features[features.sum(axis=1) > 3]
        matches
```

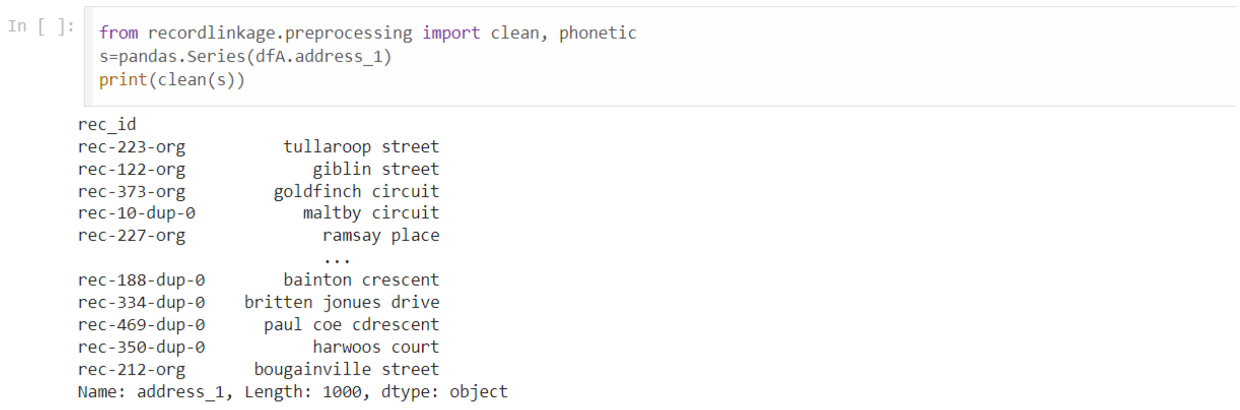
Out[ ]:

rec_id_1	rec_id_2	given_name	surname	date_of_birth	suburb	state	address_1
rec-122-dup-0	rec-122-org	1	1.0	1	1	1	1.0
rec-183-org	rec-183-dup-0	1	1.0	1	1	1	1.0
rec-248-dup-0	rec-248-org	1	1.0	1	1	1	1.0
rec-373-dup-0	rec-373-org	1	1.0	1	1	1	1.0
rec-10-org	rec-10-dup-0	1	1.0	1	1	1	1.0
...	...	...	...	...	...	...	...
rec-184-dup-0	rec-184-org	1	1.0	1	0	1	1.0
rec-252-org	rec-252-dup-0	1	1.0	1	1	1	1.0
rec-48-dup-0	rec-48-org	1	1.0	1	1	1	1.0
rec-298-dup-0	rec-298-org	1	1.0	1	1	1	0.0
rec-282-org	rec-282-dup-0	1	1.0	1	1	1	0.0

317 rows × 6 columns

-Filters the DataFrame, selecting only the rows where the sum of values is greater than 3, and assigns this filtered DataFrame to matches.

Step 11- Perform pre-processing on the Dataset



```
In [ ]: from recordlinkage.preprocessing import clean, phonetic
        s=pandas.Series(dfA.address_1)
        print(clean(s))
```

```
rec_id
rec-223-org          tullaroop street
rec-122-org             giblin street
rec-373-org          goldfinch circuit
rec-10-dup-0            maltby circuit
rec-227-org              ramsay place
                              ...
rec-188-dup-0         bainton crescent
rec-334-dup-0     britten jonues drive
rec-469-dup-0       paul coe cdrescent
rec-350-dup-0            harwoos court
rec-212-org         bougainville street
Name: address_1, Length: 1000, dtype: object
```

SHINJINI SEN- 21BCI0039
VINEET SINGH- 21BCI0130

```
In [ ]: recordlinkage.preprocessing.clean(dfA.given_name, lowercase=True, replace_by_none='[^ \-\_A-Za-z0-9]+', replace_by_whitespac
```

```
Out[ ]: rec_id
        rec-223-org            NaN
        rec-122-org        lachlan
        rec-373-org         deakin
        rec-10-dup-0         kayla
        rec-227-org           luke
                           ...
        rec-188-dup-0    stephanie
        rec-334-dup-0     nicholas
        rec-469-dup-0      lachlan
        rec-350-dup-0      monique
        rec-212-org            NaN
        Name: given_name, Length: 1000, dtype: object
```

```
In [ ]: recordlinkage.preprocessing.clean(dfA.date_of_birth)
```

```
Out[ ]: rec_id
        rec-223-org      19081209
        rec-122-org      19990219
        rec-373-org      19600210
        rec-10-dup-0     19150612
        rec-227-org      19831024
                           ...
        rec-188-dup-0    19421008
        rec-334-dup-0    19970422
        rec-469-dup-0    19380406
        rec-350-dup-0    19790807
        rec-212-org      19360219
        Name: date_of_birth, Length: 1000, dtype: object
```

```
In [ ]: recordlinkage.preprocessing.value_occurence(dfA.date_of_birth)
```

```
Out[ ]: rec_id
        rec-223-org      2
        rec-122-org      2
        rec-373-org      2
        rec-10-dup-0     2
        rec-227-org      2
                        ..
        rec-188-dup-0    2
        rec-334-dup-0    2
        rec-469-dup-0    2
        rec-350-dup-0    2
        rec-212-org      2
        Name: date_of_birth, Length: 1000, dtype: int64
```

```
In [ ]: recordlinkage.preprocessing.value_occurence(dfA.surname)
```

```
Out[ ]: rec_id
        rec-223-org       1
        rec-122-org      13
        rec-373-org       2
        rec-10-dup-0      5
        rec-227-org       2
                         ..
        rec-188-dup-0     1
        rec-334-dup-0    18
        rec-469-dup-0     1
        rec-350-dup-0     2
        rec-212-org       4
        Name: surname, Length: 1000, dtype: int64
```

```
In [ ]: recordlinkage.preprocessing.phonetic(dfA['surname'], method='nysiis', concat=True, encoding='utf-8', decode_error='strict')
```

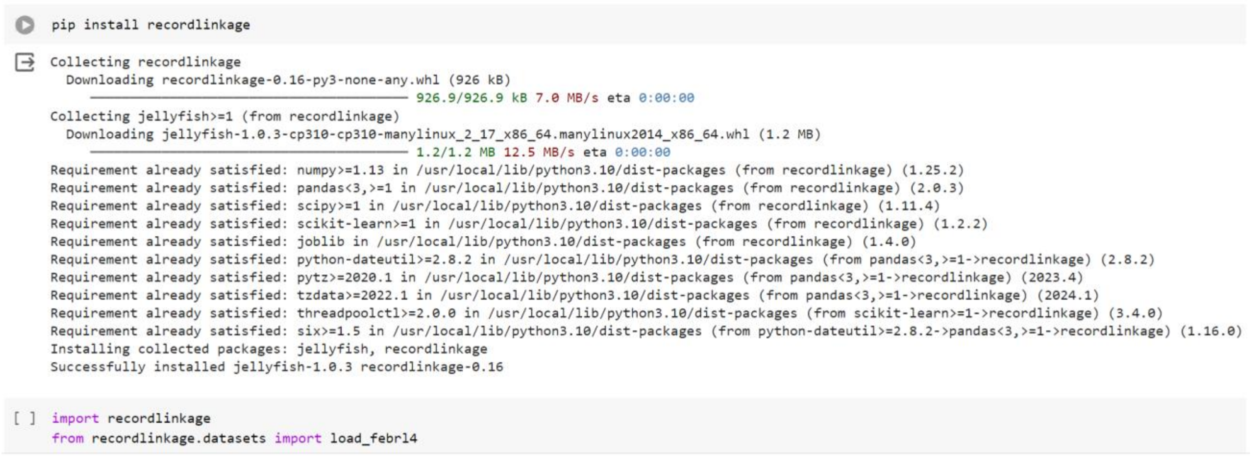
```
Out[ ]: rec_id
        rec-223-org            WALAR
        rec-122-org             BARY
        rec-373-org       SANDARGALD
        rec-10-dup-0       HARANGTAN
        rec-227-org           PARDAN
                             ...
        rec-188-dup-0              G
        rec-334-dup-0            NaN
        rec-469-dup-0         CATSAV
        rec-350-dup-0        GARGALY
        rec-212-org            MCVAG
        Name: surname, Length: 1000, dtype: object
```

Implementation on Dataset-2

The submodule recordlinkage.datasets contains several datasets that can be used for testing. For this example, we use the Febrl dataset 14.

Link to google colab – https://colab.research.google.com/drive/13hPoDXDEAF_WfrckgVvMbV5ekgRULM8I?usp=sharing

- **Install record linkage and import dataset**

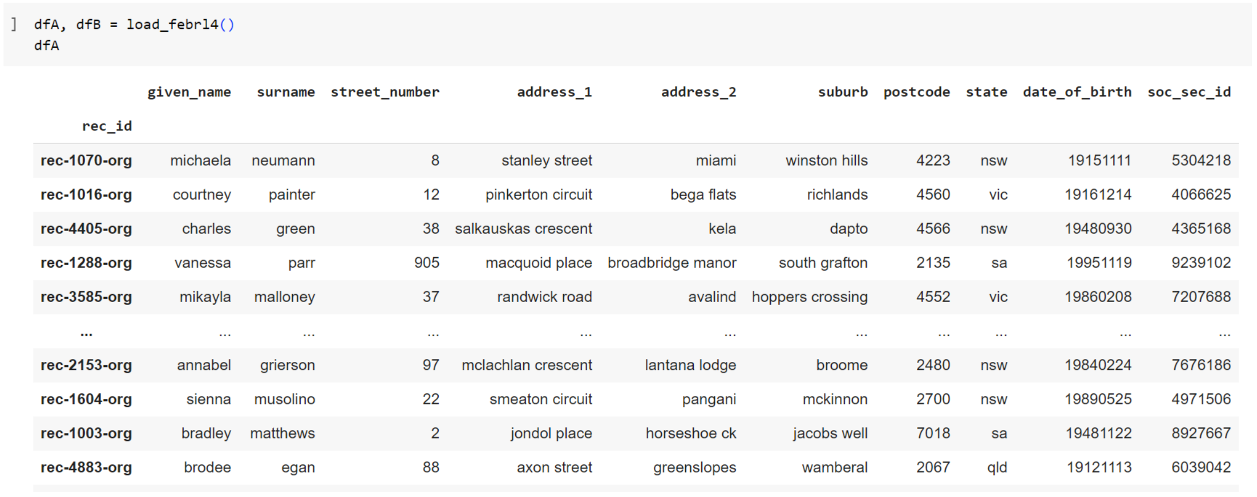


```
pip install recordlinkage

Collecting recordlinkage
  Downloading recordlinkage-0.16-py3-none-any.whl (926 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 926.9/926.9 kB 7.0 MB/s eta 0:00:00
Collecting jellyfish>=1 (from recordlinkage)
  Downloading jellyfish-1.0.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.2/1.2 MB 12.5 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.13 in /usr/local/lib/python3.10/dist-packages (from recordlinkage) (1.25.2)
Requirement already satisfied: pandas<3,>=1 in /usr/local/lib/python3.10/dist-packages (from recordlinkage) (2.0.3)
Requirement already satisfied: scipy>=1 in /usr/local/lib/python3.10/dist-packages (from recordlinkage) (1.11.4)
Requirement already satisfied: scikit-learn>=1 in /usr/local/lib/python3.10/dist-packages (from recordlinkage) (1.2.2)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from recordlinkage) (1.4.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas<3,>=1->recordlinkage) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas<3,>=1->recordlinkage) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas<3,>=1->recordlinkage) (2024.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1->recordlinkage) (3.4.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas<3,>=1->recordlinkage) (1.16.0)
Installing collected packages: jellyfish, recordlinkage
Successfully installed jellyfish-1.0.3 recordlinkage-0.16
```

```
[ ] import recordlinkage
    from recordlinkage.datasets import load_febrl4
```

- **Loading data from the Febrl4 dataset into two separate DataFrames, dfA and df**



```
] dfA, dfB = load_febrl4()
  dfA
```

rec_id	given_name	surname	street_number	address_1	address_2	suburb	postcode	state	date_of_birth	soc_sec_id
rec-1070-org	michaela	neumann	8	stanley street	miami	winston hills	4223	nsw	19151111	5304218
rec-1016-org	courtney	painter	12	pinkerton circuit	bega flats	richlands	4560	vic	19161214	4066625
rec-4405-org	charles	green	38	salkauskas crescent	kela	dapto	4566	nsw	19480930	4365168
rec-1288-org	vanessa	parr	905	macquoid place	broadbridge manor	south grafton	2135	sa	19951119	9239102
rec-3585-org	mikayla	malloney	37	randwick road	avalind	hoppers crossing	4552	vic	19860208	7207688
...	...	...	...	...	...	...	...	...	...	...
rec-2153-org	annabel	grierson	97	mclachlan crescent	lantana lodge	broome	2480	nsw	19840224	7676186
rec-1604-org	sienna	musolino	22	smeaton circuit	pangani	mckinnon	2700	nsw	19890525	4971506
rec-1003-org	bradley	matthews	2	jondol place	horseshoe ck	jacobs well	7018	sa	19481122	8927667
rec-4883-org	brodee	egan	88	axon street	greenslopes	wamberal	2067	qld	19121113	6039042

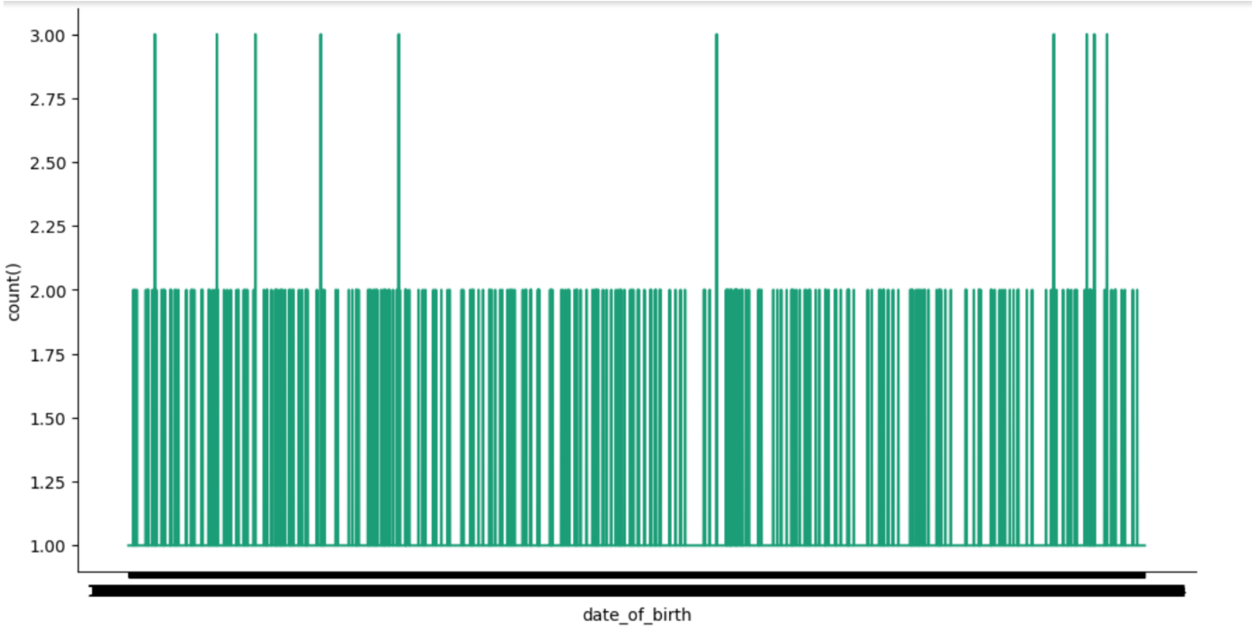
- **Generating a plot for date of birth vs count**

date_of_birth vs count()

```
[ ] # @title date_of_birth vs count()

    from matplotlib import pyplot as plt
    import seaborn as sns
    def _plot_series(series, series_name, series_index=0):
      palette = list(sns.palettes.mpl_palette('Dark2'))
      counted = (series['date_of_birth']
                    .value_counts()
                  .reset_index(name='counts')
                  .rename({'index': 'date_of_birth'}, axis=1)
                  .sort_values('date_of_birth', ascending=True))
      xs = counted['date_of_birth']
      ys = counted['counts']
      plt.plot(xs, ys, label=series_name, color=palette[series_index % len(palette)])

    fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
    df_sorted = dfA.sort_values('date_of_birth', ascending=True)
    _plot_series(df_sorted, '')
    sns.despine(fig=fig, ax=ax)
    plt.xlabel('date_of_birth')
    _ = plt.ylabel('count()')
```



- **Make record pairs - compare each record in DataFrame dfA with all records of DataFrame dfB and printing the length.**

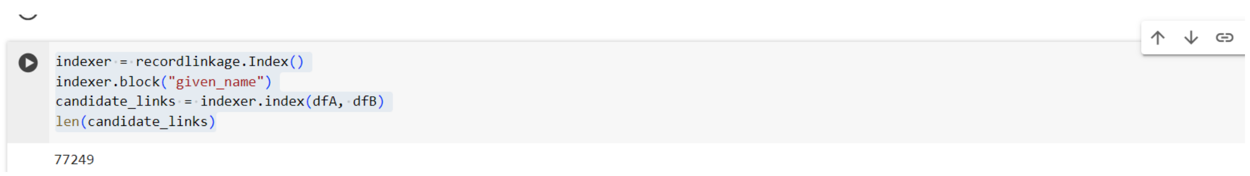
```
[ ] indexer = recordlinkage.Index()
    indexer.full()
    pairs = indexer.index(dfA, dfB)
```

```
WARNING:recordlinkage:indexing - performance warning - A full index can result in large number of record pairs.
WARNING:recordlinkage:indexing - performance warning - A full index can result in large number of record pairs.
```

```
[ ] print(len(dfA), len(dfB), len(pairs))
```

```
5000 5000 25000000
```

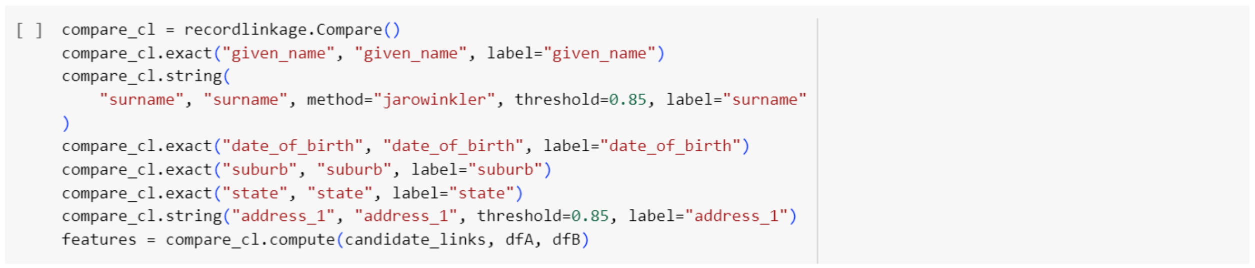
- **This code initializes an indexer to perform blocking on the "given_name" column, then generates candidate record pairs from DataFrames dfA and dfB based on the blocking configuration.**



```
indexer = recordlinkage.Index()
indexer.block("given_name")
candidate_links = indexer.index(dfA, dfB)
len(candidate_links)
```

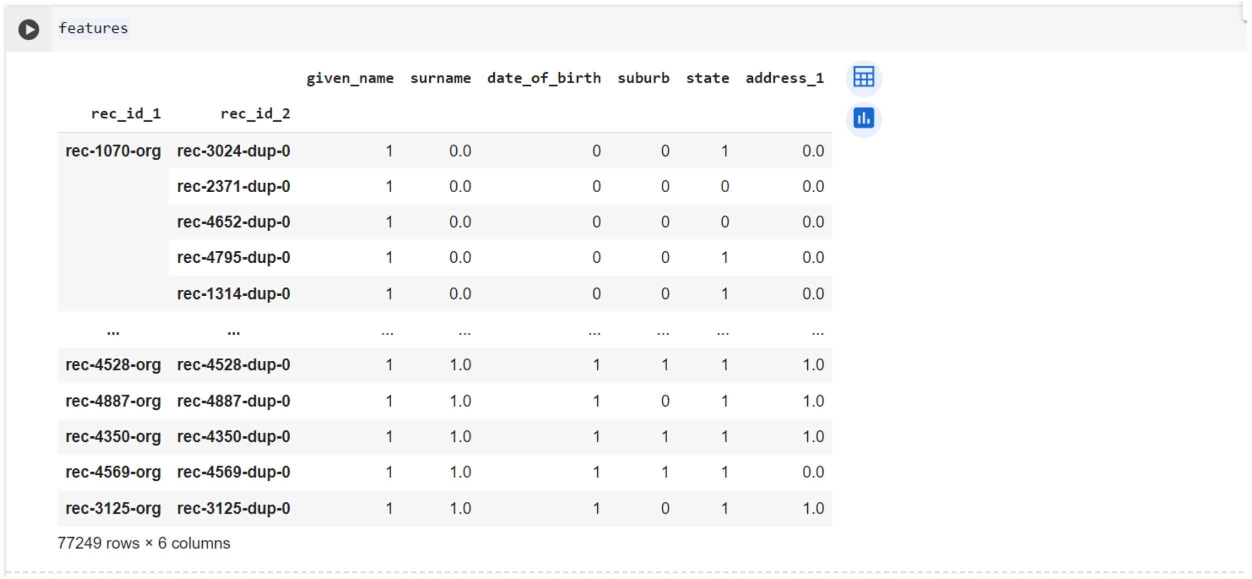
```
77249
```

- **Compare Records – Each record pair is a candidate match. To classify the candidate record pairs into matches and non-matches, compare the records on all attributes both records have in common.**



```
[ ] compare_cl = recordlinkage.Compare()
    compare_cl.exact("given_name", "given_name", label="given_name")
    compare_cl.string(
        "surname", "surname", method="jarowinkler", threshold=0.85, label="surname"
    )
    compare_cl.exact("date_of_birth", "date_of_birth", label="date_of_birth")
    compare_cl.exact("suburb", "suburb", label="suburb")
    compare_cl.exact("state", "state", label="state")
    compare_cl.string("address_1", "address_1", threshold=0.85, label="address_1")
    features = compare_cl.compute(candidate_links, dfA, dfB)
```

- **Features**

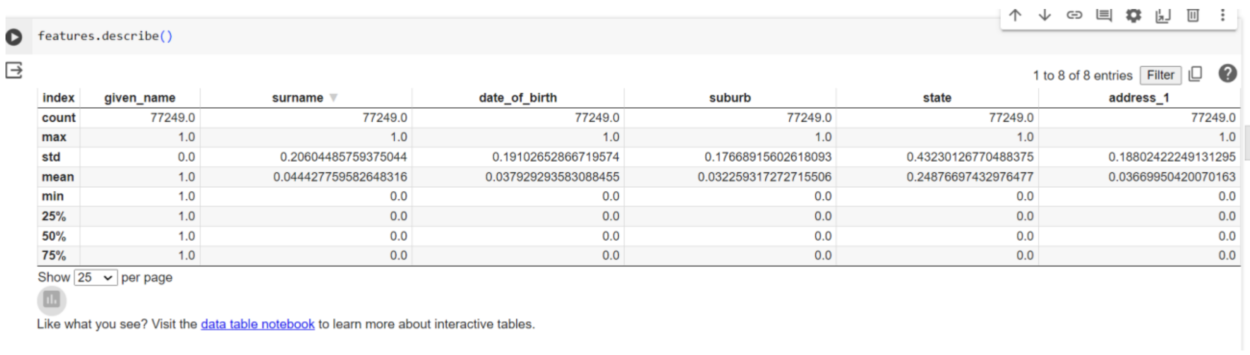


```
features
```

rec_id_1	rec_id_2	given_name	surname	date_of_birth	suburb	state	address_1
rec-1070-org	rec-3024-dup-0	1	0.0	0	0	1	0.0
	rec-2371-dup-0	1	0.0	0	0	0	0.0
	rec-4652-dup-0	1	0.0	0	0	0	0.0
	rec-4795-dup-0	1	0.0	0	0	1	0.0
	rec-1314-dup-0	1	0.0	0	0	1	0.0
...	...	...	...	...	...	...	...
rec-4528-org	rec-4528-dup-0	1	1.0	1	1	1	1.0
rec-4887-org	rec-4887-dup-0	1	1.0	1	0	1	1.0
rec-4350-org	rec-4350-dup-0	1	1.0	1	1	1	1.0
rec-4569-org	rec-4569-dup-0	1	1.0	1	1	1	0.0
rec-3125-org	rec-3125-dup-0	1	1.0	1	0	1	1.0

77249 rows × 6 columns

- **Features.describe(): summary statistics of features in a dataset**



```
features.describe()
```

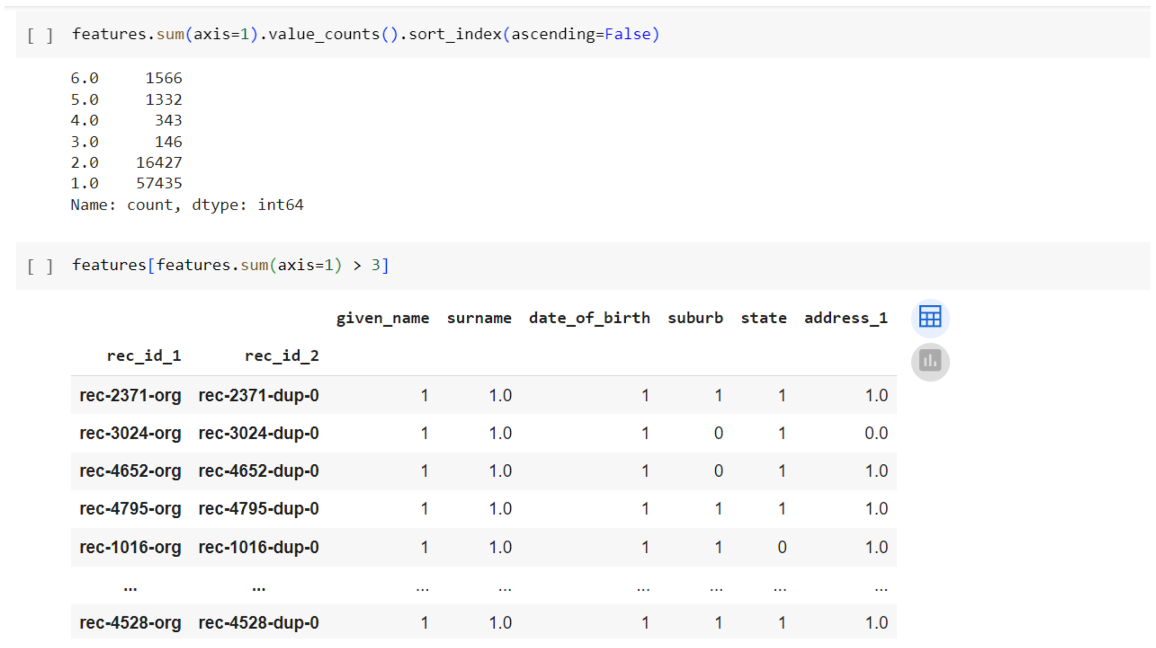
1 to 8 of 8 entries

index	given_name	surname	date_of_birth	suburb	state	address_1
count	77249.0	77249.0	77249.0	77249.0	77249.0	77249.0
max	1.0	1.0	1.0	1.0	1.0	1.0
std	0.0	0.20604485759375044	0.19102652866719574	0.17668915602618093	0.43230126770488375	0.18802422249131295
mean	1.0	0.044427759582648316	0.037929293583088455	0.032259317272715506	0.24876697432976477	0.03669950420070163
min	1.0	0.0	0.0	0.0	0.0	0.0
25%	1.0	0.0	0.0	0.0	0.0	0.0
50%	1.0	0.0	0.0	0.0	0.0	0.0
75%	1.0	0.0	0.0	0.0	0.0	0.0

Show 25 per page

Like what you see? Visit the data table notebook to learn more about interactive tables.

- **Decide which records belong to the same person**



```
[ ] features.sum(axis=1).value_counts().sort_index(ascending=False)
```

```
6.0     1566
5.0     1332
4.0      343
3.0      146
2.0    16427
1.0    57435
Name: count, dtype: int64
```

```
[ ] features[features.sum(axis=1) > 3]
```

rec_id_1	rec_id_2	given_name	surname	date_of_birth	suburb	state	address_1
rec-2371-org	rec-2371-dup-0	1	1.0	1	1	1	1.0
rec-3024-org	rec-3024-dup-0	1	1.0	1	0	1	0.0
rec-4652-org	rec-4652-dup-0	1	1.0	1	0	1	1.0
rec-4795-org	rec-4795-dup-0	1	1.0	1	1	1	1.0
rec-1016-org	rec-1016-dup-0	1	1.0	1	1	0	1.0
...	...	...	...	...	...	...	...
rec-4528-org	rec-4528-dup-0	1	1.0	1	1	1	1.0

Record Linkage for Real-World Data

Real Data Application is the application of Record Linkage on NOSQL databases.

Dataset A: Contains information about customers from a retail store, including names, addresses, and contact numbers.

Dataset B: Contains information about customers from an online platform, including names, email addresses, and purchase history.

Our goal is to link or match records from Dataset A with corresponding records in Dataset B to create a unified customer database.

Step-by-step approach to applying record linkage:

Data Preprocessing:
Clean and preprocess both datasets to handle missing values, standardize formats (e.g., convert all names to lowercase), and remove irrelevant information.

Blocking:
Divide both datasets into smaller blocks based on certain attributes that are likely to match, such as zip codes or phone numbers. This reduces the computational complexity of comparing every pair of records.

Comparison:
For each block, compare pairs of records from Dataset A and Dataset B using similarity metrics such as string similarity (e.g., Jaccard similarity for names) or numeric similarity (e.g., Levenshtein distance for addresses).

Scoring:
Calculate a similarity score for each pair of records based on the comparison results. This score represents the likelihood that the two records represent the same entity.

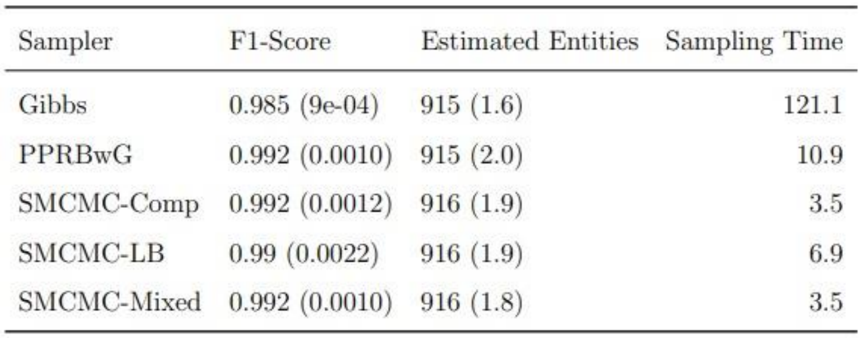
Classification:
Decide on a threshold for the similarity score above which pairs of records are considered as matches. Pairs with scores below the threshold are considered non-matches.

Evaluation:
Evaluate the performance of the record linkage process using metrics such as precision, recall, and F1-score. This helps assess the accuracy of the matched pairs compared to a ground truth if available.

Post-processing:
Resolve any potential conflicts or duplicates in the matched pairs, and consolidate the matched records into a single database.

We apply streaming record linkage to a sample of records from a longitudinal survey with a known true identity for each record. Individuals may be recorded multiple times in separate years but there is no duplication of individuals within a year. Four files of data were selected from the full dataset from the years 2007 through 2013. The four files have varying sizes, with n1 = 151, n2 = 464, n3 = 688, and n4 = 677, for a total of 1980 records. The files were created by randomly sampling, without replacement, 910 individuals from all individuals appearing in at least one of the included years. Of the 910 individuals, 306 appear in just one file, 240 appear in two files, 262 appear in three files, and 102 appear in all four files. Linkage was performed using six fields: gender, province, educational attainment, and year, month, and day of birth. All fields are categorical and were compared using binary comparisons. We chose hyperparameters to produce flat priors in m, u, and Z(ℓ) for ℓ = 1, 2, 3. We compared five samplers: a non-streaming Gibbs sampler (Gibbs), sequentially applied PPRB-within-Gibbs updates with locally balanced proposals (PPRBwG), and sequentially applied SMCMC updates with component-wise proposals (SMCMC-Comp), locally balanced proposals (SMCMC-LB) or a mix using component-wise jumping kernel proposals and locally balanced transition kernel proposals (SMCMC-Mixed).

The streaming record linkage models were able to recover the true co-referent records.



Sampler	F1-Score	Estimated Entities	Sampling Time
Gibbs	0.985 (9e-04)	915 (1.6)	121.1
PPRBwG	0.992 (0.0010)	915 (2.0)	10.9
SMCMC-Comp	0.992 (0.0012)	916 (1.9)	3.5
SMCMC-LB	0.99 (0.0022)	916 (1.9)	6.9
SMCMC-Mixed	0.992 (0.0010)	916 (1.8)	3.5

The above Table shows the posterior F1-score distribution for each of the 5 samplers, the posterior distribution of the estimated number of entities resulting from the linkage, and the time to generate the posterior samples. All samplers performed equally well at recovering the true co-referent record sets with a posterior mean F1-score between 0.985 and 0.992.

Streaming samplers were significantly faster than the non-streaming Gibbs sampler, with times given for the cumulative time required to produce both three-file and four-file inference using each sampling method. This is representative of the streaming data setting where inference is required after each new file arrives. The streaming samplers show between 11 times and 35 times speedup when compared to the non-streaming Gibbs sampler, where SMCMC time estimates are based on the assumption that enough cores are available for each ensemble to be run simultaneously in parallel.

Table: Posterior means and standard deviations of F1-score and estimated number of entities, and total sampling time, for **the four-file Poland SDS data set using five samplers.** There are 910 true entities in the four files. Sampling time is given in cumulative hours required to produce posterior samples of the parameters conditioned first on three files, then on four files using each sampling method. The SMCMC sampling time is estimated assuming 1000 available cores so that each ensemble member can be updated in parallel.

Use Cases and Applications of Record Linkage

Record linkage has numerous applications across various domains, including healthcare, finance, government, marketing, and research.

Healthcare Fraud Detection:

Incident: In 2018, the U.S. Department of Justice announced the largest healthcare fraud enforcement action in its history, charging hundreds of individuals for their involvement in healthcare fraud schemes totaling billions of dollars.

Application: Record linkage was likely a crucial tool in identifying fraudulent activities by linking patient records, billing information, and provider data to uncover patterns indicative of fraud, such as billing for services not rendered or upcoding.

Disaster Response and Recovery:

Incident: Following natural disasters such as hurricanes, floods, or earthquakes, governments and relief organizations need to quickly identify and locate affected individuals to provide assistance and support.

Application: Record linkage helps in integrating data from various sources such as emergency shelters, government databases, and social media to identify individuals in need of assistance, reunite families, and coordinate relief efforts effectively.

Financial Crime Investigation:

Incident: In recent years, several high-profile cases of money laundering, terrorist financing, and fraud have been reported globally, involving banks, financial institutions, and individuals.

Application: Record linkage enables financial institutions and regulatory agencies to link and analyze large volumes of transactional data, customer profiles, and suspicious activity reports to detect and investigate financial crimes, identify money trails, and prevent illicit activities.

Public Health Surveillance:

Incident: During disease outbreaks or public health emergencies, such as the COVID-19 pandemic, it is essential to track and monitor the spread of the disease, identify clusters of cases, and implement control measures.

Application: Record linkage facilitates the integration of data from healthcare facilities, laboratories, public health departments, and other sources to create comprehensive disease surveillance systems. By linking demographic, clinical, and geographic data, authorities can track the progression of diseases, identify at-risk populations, and allocate resources effectively.

Customer Relationship Management (CRM):

Incident: E-commerce platforms, social media companies, and online retailers collect vast amounts of customer data to personalize services, target advertisements, and improve customer experiences.

Application: Record linkage enables companies to integrate data from multiple touchpoints, such as website visits, purchase history, social media interactions, and customer support inquiries, to create unified customer profiles. By linking and analyzing this data, companies can tailor marketing campaigns, recommend products, and provide personalized services to customers.

References:

1. Record Linkage in Health Data: a simulation study https://www.cbs.nl//media/imported/documents/2014/16/2014-record-linkage-art.pdf
2. https://www.cbs.nl/-/media/imported/documents/2014/16/2014-record-linkage-art.pdf
3. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=a2c4dec86a96a99adc00cb664b703e8407216183
4. https://medium.com/data-science-business/record-linkage-merging-disparate-datasets-8aa02a2e4535
5. https://recordlinkage.readthedocs.io/en/latest/guides/link_two_dataframes.html
6. https://pub.towardsai.net/the-python-record-linkage-toolkit-9b1c59fd156a
7. https://f-tadao.medium.com/match-and-deduplicate-web-data-record-linkage-fe3fa75fef83