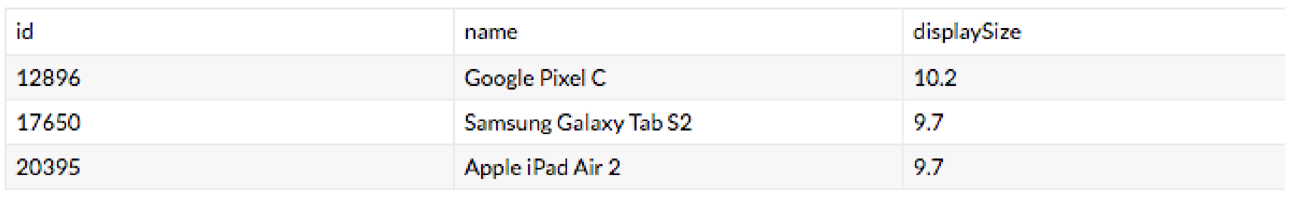
How do I use Data Sets for deduplication/record linkage?

Data sets provide means to combine and deduplicate rows.

Written by Morten Franck
Updated over a week ago

A dexi.io data set can be seen as a table in a relational/SQL database, a collection in a NoSQL database or a single sheet in a spreadsheet. A data set has a data type defining the fields for each row in the data set. Rows can be added (inserted), viewed, modified (updated) and deleted (CRUD).



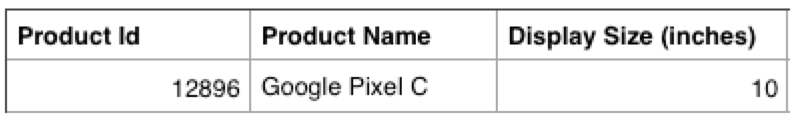
id	name	displaySize
12896	Google Pixel C	10.2
17650	Samsung Galaxy Tab S2	9.7
20395	Apple iPad Air 2	9.7

Data sets in Dexi have a feature, that in one sense, makes them more advanced than their SQL counterparts: the **key**, or *primary key* using SQL terminology, used to determine duplicate/matching rows is **dynamically configurable** - even after data has been created. Data sets with this feature can be used for ***data deduplication*** (duplicate detection) and ***record linkage***.

Example

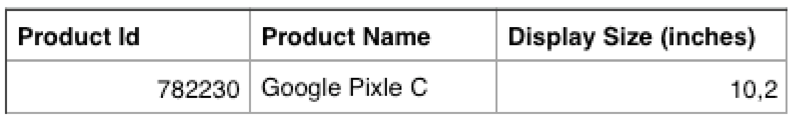
The dynamic key configuration can be used to build and maintain a "truth" table where rows with similar values are matched, or "merged". Example:

Row A:



Product Id	Product Name	Display Size (inches)
12896	Google Pixel C	10

Row B:



Product Id	Product Name	Display Size (inches)
782230	Google Pixle C	10,2

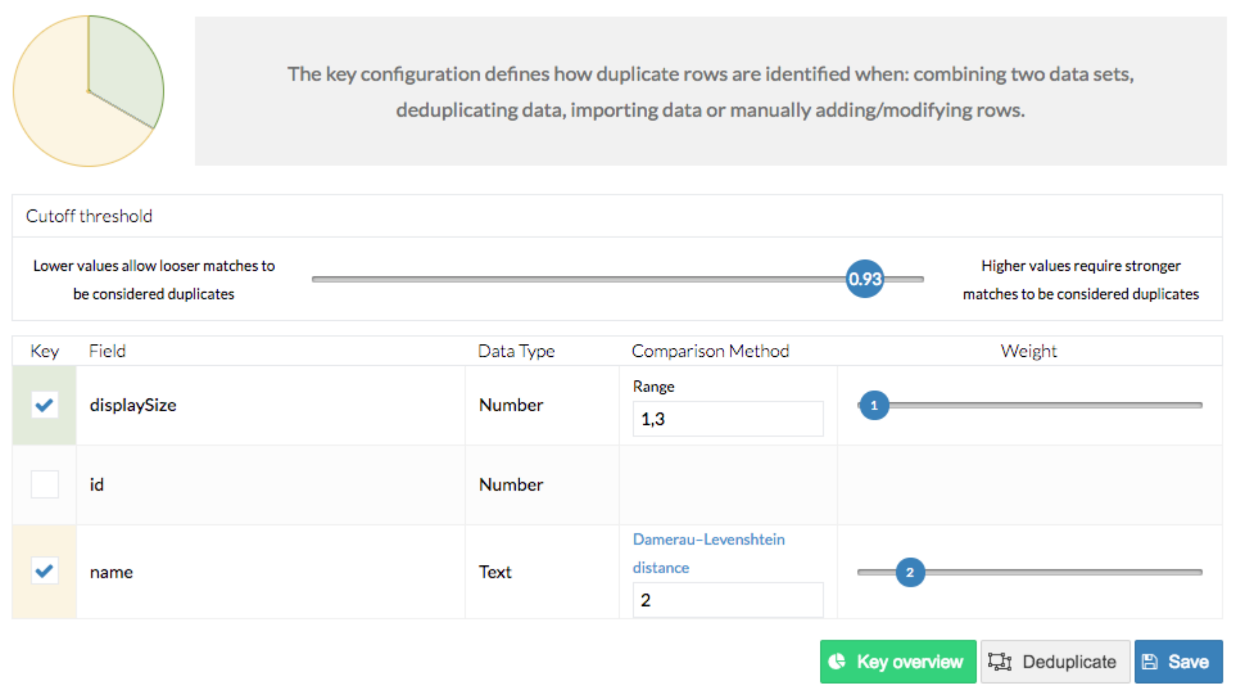
Inserting row B into a data set containing row A using the appropriate key configuration defined on the data set (explained below) can automatically identify row B as a ***duplicate***, or ***match***, of row A, even though the values for *Product Name* and *Display Size* and not equal, e.g. the value for *Product Name* in row B contains a spelling error.

We define *deduplication* to mean an operation that matches rows and removes duplicates **within one data set** and *record linkage* as an operation that **combines two data sets** (see sections *Deduplication* and *Combine (Record Linkage)* below, respectively).

The details and the different variations are explained the sections below.

Key Configuration

The key configuration defines how the system ***matches*** rows, ie. determines **duplicates**. A data set key can thus be seen as *primary key* in SQL terminology.



The key configuration defines how duplicate rows are identified when: combining two data sets, deduplicating data, importing data or manually adding/modifying rows.

Cutoff threshold

Lower values allow looser matches to be considered duplicates — 0.93 — Higher values require stronger matches to be considered duplicates

Key	Field	Data Type	Comparison Method	Weight
✓	displaySize	Number	Range 1,3	1
	id	Number		
✓	name	Text	Damerau-Levenshtein distance 2	2

For each field of the data type of the data set, the user selects the fields he/she wants to be part of the key and for each key field define:

- The ***comparison method*** - see *Comparison Methods*.
- The ***weight***: a value from 1-10 indicating how important the value of this field is relative to the values of the other fields.

When a row is compared to existing rows in a data set operation (see *Operations*) a **similarity score**, or just ***score***, is calculated for each new row using the weights:

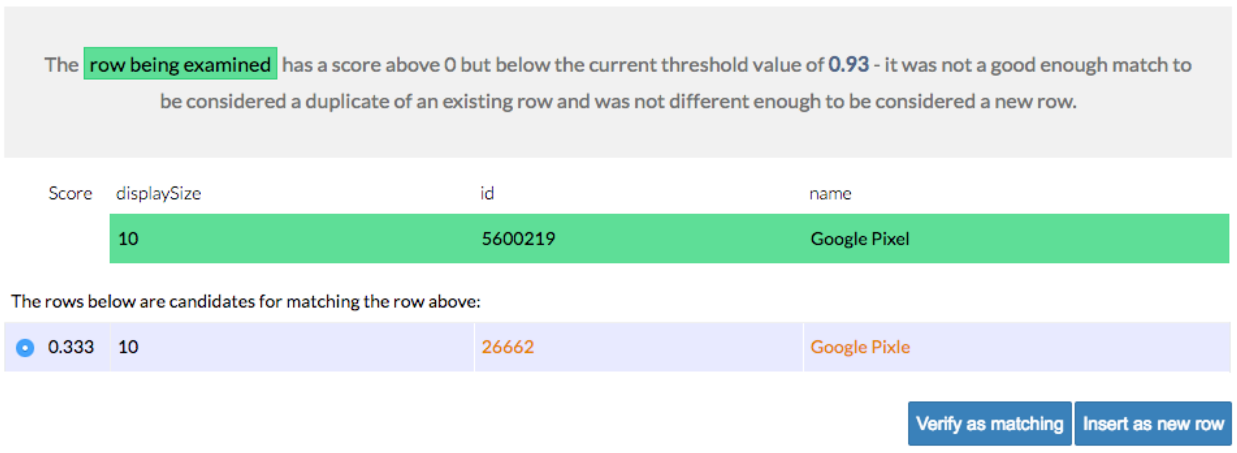
*similarity score = sum(key field score * key field weight) / sum(key field weight)*

The ***cutoff threshold***, or just ***threshold***, defines the value which the row score must be equal to or greater than to automatically be identified by the system as a duplicate and is used in combination with the score as follows:

- **score == 0**: no match, or *unknown* row. The row is added directly to the data set as a new row.
- **0 < score < threshold**: *partial* match. The row has the top x (currently, x=10) existing rows (if such rows exist) added to it as match candidates and the new row must be manually verified/matched to one of these existing rows.
- **score >= threshold**: *perfect* match. The row is added as a duplicate row with the perfectly matched row as its parent row.
- **score == 1.0**: *identical* row. The row is skipped (but it is recorded on the existing identical row that it was "touched").

Manual Verification (Matching)

A new row that does not have a high enough score (cf. explanation above) to be automatically matched by the system must be manually verified as a duplicate row.


The row being examined has a score above 0 but below the current threshold value of 0.93 - it was not a good enough match to be considered a duplicate of an existing row and was not different enough to be considered a new row.

Score	displaySize	id	name
	10	5600219	Google Pixel

The rows below are candidates for matching the row above:

● 0.333	10	26662	Google Pixle

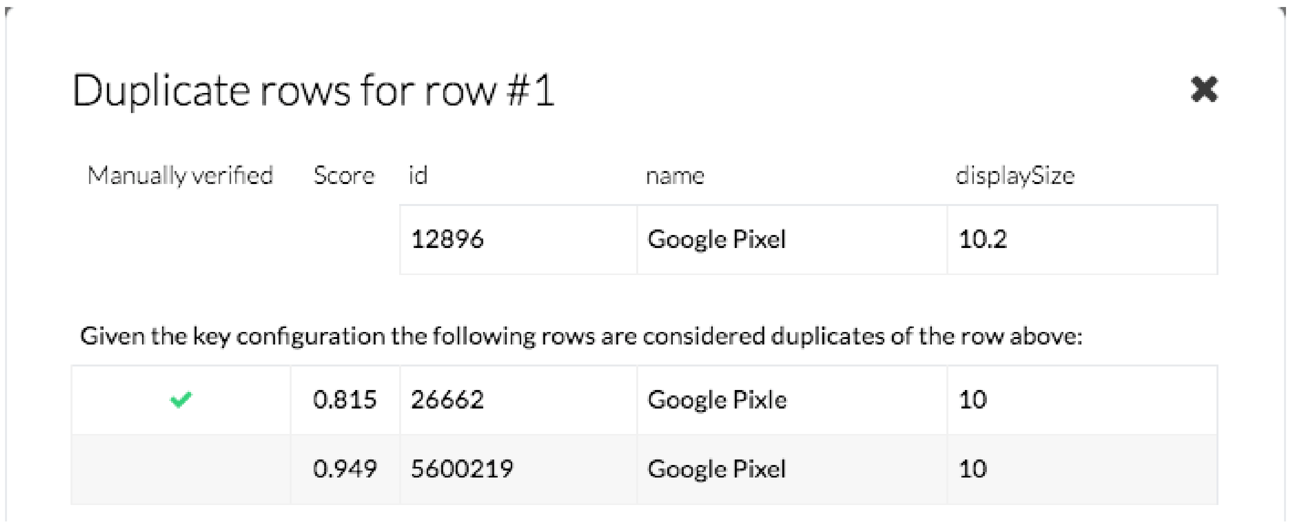
The match candidates are presented to the user and he/she selects, i.e. *verifies*, which match candidate best matches the new row. If one such row does not exist (or for some other reason), the row can instead be inserted as a **new row**.

Manually verified rows are skipped when comparing rows. The motivation for this is to avoid the system automatically changing (merging/splitting) rows on which a lot of time and effort have potentially been put into.

Any change to the key configuration and consequent operation can result in new rows that must be manually matched. See also *Deduplication*.

"Real" (Master) Data & Duplicates (Parent/Child Rows)

No matter if a row was automatically matched by the system or manually by the user, such a duplicate row can be viewed as a *child* row to the existing row that was chosen as its match.


Duplicate rows for row #1

Manually verified	Score	id	name	displaySize
		12896	Google Pixel	10.2

Given the key configuration the following rows are considered duplicates of the row above:

✓	0.815	26662	Google Pixle	10
	0.949	5600219	Google Pixel	10

Only ***parent*** rows should be seen as the data set's **real data** or *master* data.

Transitive Matches

Child rows should not be seen as actual data but *can* serve as match candidates for new rows, resulting in ***transitive*** matches:

- Row A <-> Row B: Score 80%
- Row B <-> New row C: Score 50%
- Row A <-> New row C: 0,8 * 0,5 = 0,4 = Score 40%

Only one level of matches are stored so in the example above the new C row will get A as its parent.

Comparison Methods

As mentioned above, the individual values of the fields that make up the key are compared to produce a score for the row. Each comparison method can be configured to tell the system how dissimilar, or how "far apart", two values are allowed to be while still being considered duplicates/matching.

Currently the following comparison methods exist:

- ***String/text***: used with a range to specify the allowed edit distance (Damerau–Levenshtein distance).
- ***Number***: used with a range to specify the allowed distance between two numbers. Works both with integers and floating point numbers.
- ***Image***: used with a mode to specify the allowed "difference" between two images. Currently the following modes exist:
- - ***Exact***: compares images pixel-by-pixel
- - ***Similar***: compares images using OpenCV's L^2-Norm algorithm
- - ***Contains***: one image contains the other image.

More comparison methods will probably be implemented in the future.

Deduplication

When the key configuration changes the data set must be ***deduplicated***. Changes in a comparison method, a weight, the threshold or which fields make up the key can e.g. result in rows previously being automatically matched now being seen as distinct rows or rows requiring manual matching.

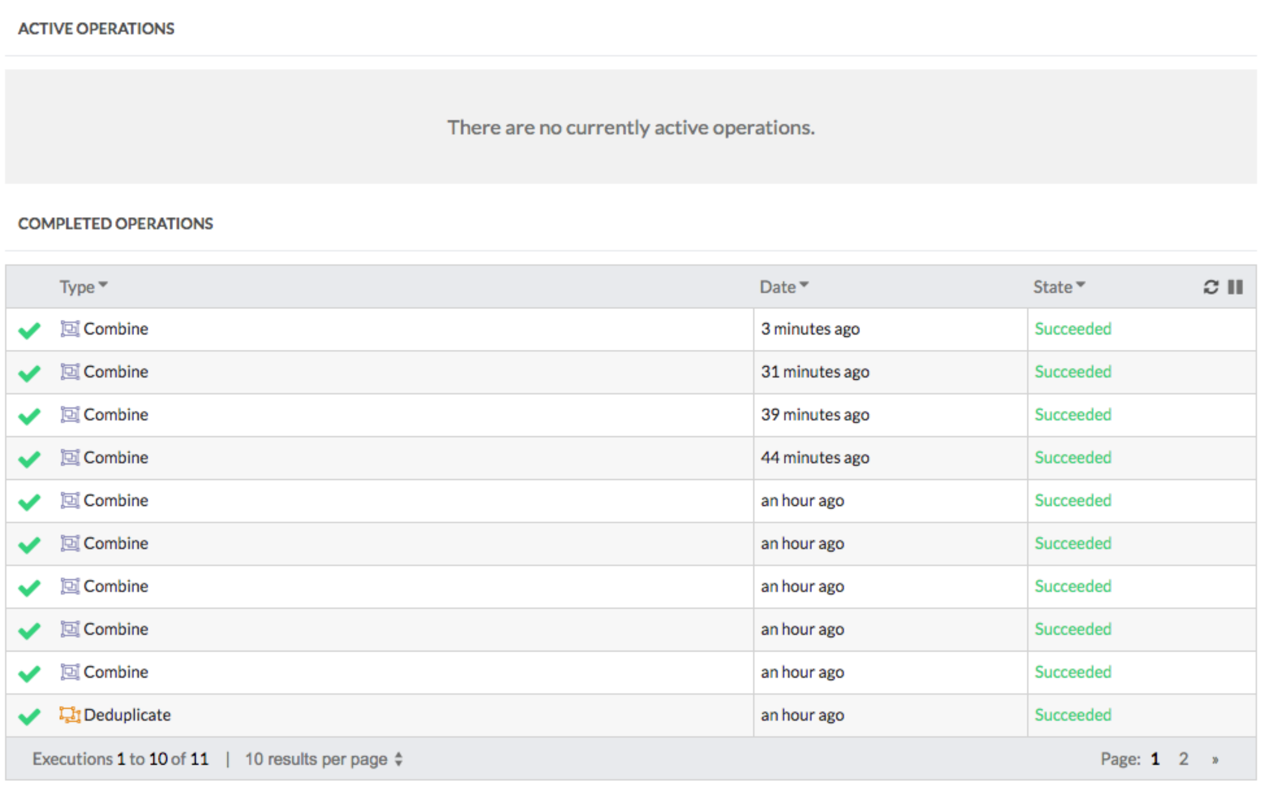
Because a deduplication operation can take a while for large data sets, the user will be asked if he/she wants to run deduplication immediately after the change to the key configuration.
NB! Not running the operation immediately will cause the data in the data set to become inconsistent with the current key configuration and such a data set should not be used for purposes other than test/debug.

Combine (Record Linkage)

The rows of one data set can be combined (merged) into another data set using the ***combine*** functionality. The operation happens one row at a time and the key configuration is used as described in *Key Configuration*.

Operations

On large data sets some operations can take a while. Currently this covers deduplication and combine.


ACTIVE OPERATIONS

There are no currently active operations.

COMPLETED OPERATIONS

	Type	Date	State
✓	Combine	3 minutes ago	Succeeded
✓	Combine	31 minutes ago	Succeeded
✓	Combine	39 minutes ago	Succeeded
✓	Combine	44 minutes ago	Succeeded
✓	Combine	an hour ago	Succeeded
✓	Combine	an hour ago	Succeeded
✓	Combine	an hour ago	Succeeded
✓	Combine	an hour ago	Succeeded
✓	Combine	an hour ago	Succeeded
✓	Deduplicate	an hour ago	Succeeded

Executions 1 to 10 of 11 | 10 results per page Page: 1 2 »

Importing a CSV and manually adding/modifying rows are not considered operations and will thus not appear on the *Operations* tab.

Data Sources

The sources of rows for data sets can be pretty much anything, e.g.:

- Importing a CSV
- Manually adding/modifying rows
- Performing data set operations, specifically:
- - Deduplication
- - Combine
- Executing a Pipe robot containing one or more of the following actions:
- - "Save row to <data set>"
- - "Lookup value in <data set>" with the "Insert unknown entries?" option enabled