

# Comparing strings

CLEANING DATA IN PYTHON



**Adel Nehme**

Content Developer @ DataCamp

# In this chapter

## Chapter 4 - Record linkage

# Minimum edit distance

I	N	T	E	N	T	I	O	N
---	---	---	---	---	---	---	---	---

E	X	E	C	U	T	I	O	N
---	---	---	---	---	---	---	---	---

Least possible amount of steps needed to transition from one string to another

# Minimum edit distance

I	N	T	E	N	T	I	O	N
---	---	---	---	---	---	---	---	---

E	X	E	C	U	T	I	O	N
---	---	---	---	---	---	---	---	---

+ Insertion

- Deletion

↔ Substitution

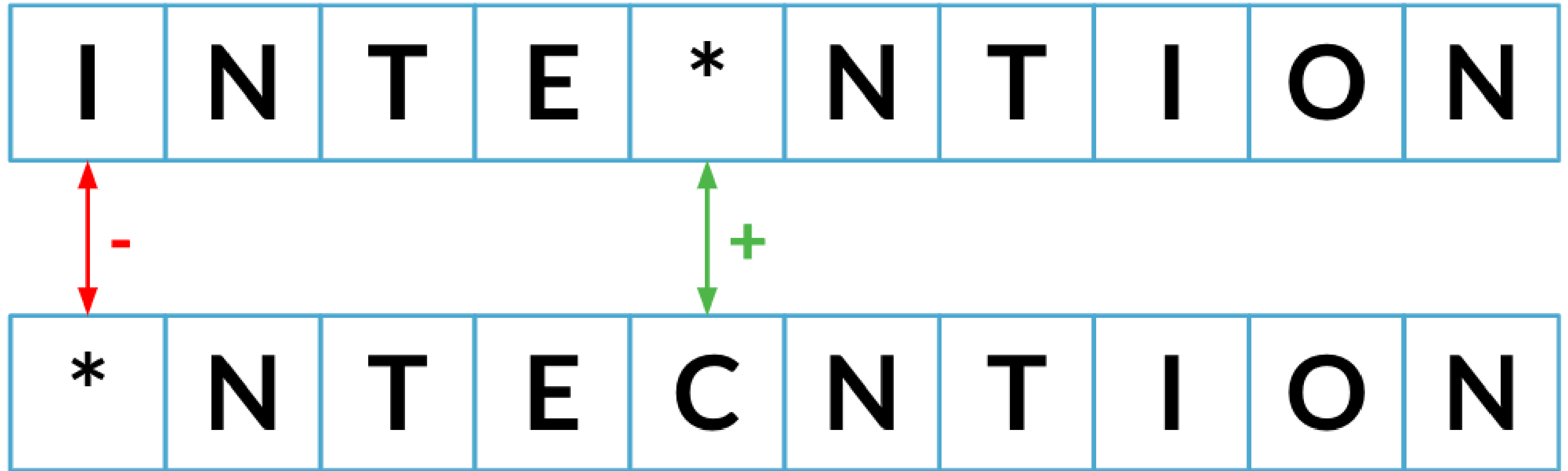
↔ Transposition

Least possible amount of steps needed to transition from one string to another

# Minimum edit distance

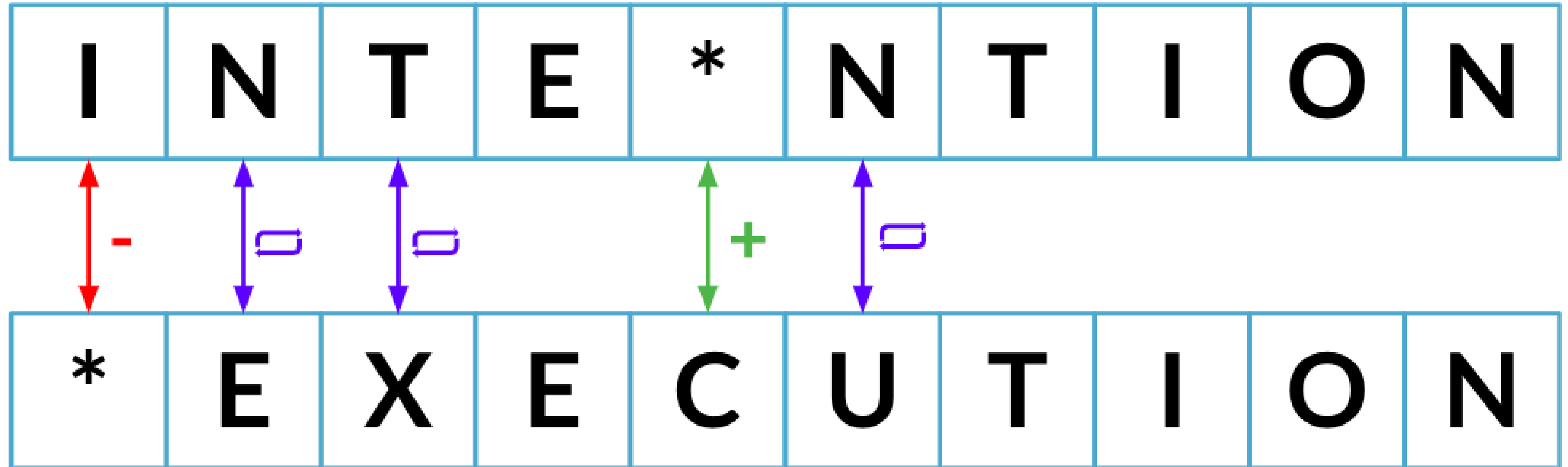
I	N	T	E	N	T	I	O	N
---	---	---	---	---	---	---	---	---

# Minimum edit distance



Minimum edit distance so far: 2

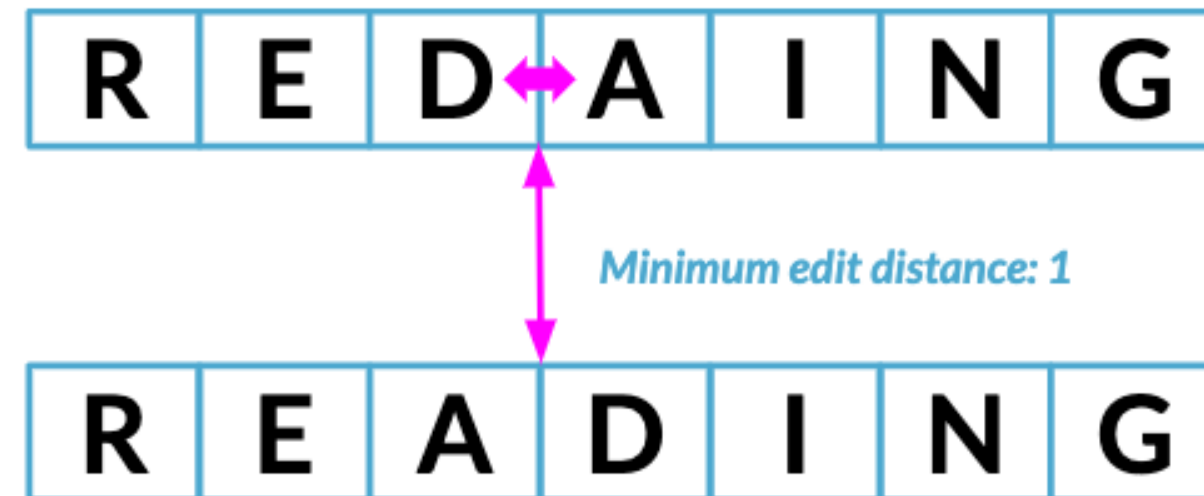
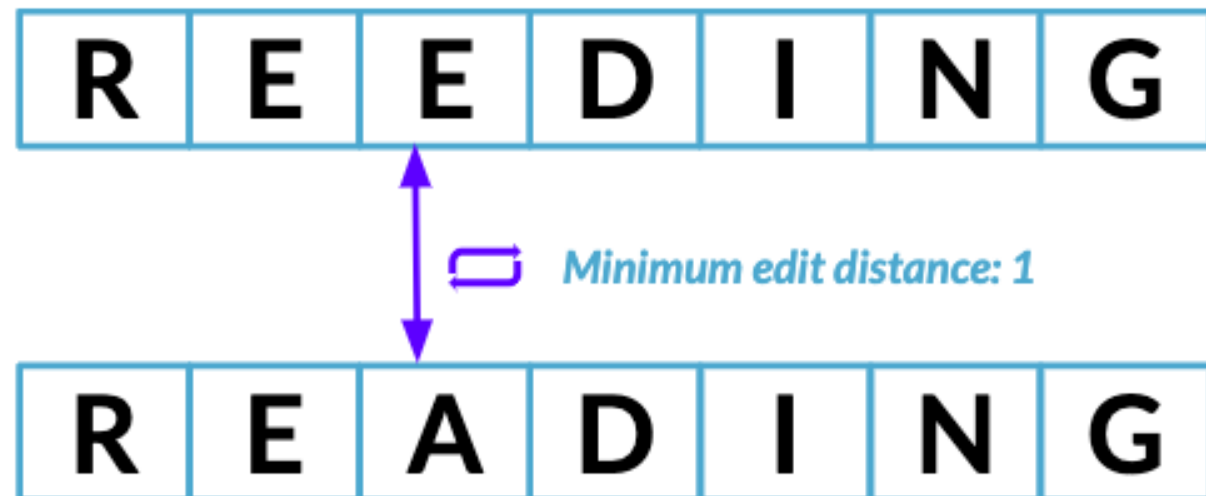
# Minimum edit distance



Minimum edit distance: 5

# Minimum edit distance

*Typos for the word:* READING





# Minimum edit distance algorithms

Algorithm	Operations
Damerau-Levenshtein	insertion, substitution, deletion, transposition
Levenshtein	insertion, substitution, deletion
Hamming	substitution only
Jaro distance	transposition only
...	...

**Possible packages:** `nltk` , `fuzzywuzzy` , `textdistance` ..

# Minimum edit distance algorithms

Algorithm	Operations
Damerau-Levenshtein	insertion, substitution, deletion, transposition
<i>Levenshtein</i>	<i>insertion, substitution, deletion</i>
Hamming	substitution only
Jaro distance	transposition only
...	...

Possible packages: `fuzzywuzzy`

# Simple string comparison

```
# Lets us compare between two strings
from fuzzywuzzy import fuzz

# Compare reeding vs reading
fuzz.WRatio('Reeding', 'Reading')
```

86

# Partial strings and different orderings

```
# Partial string comparison  
fuzz.WRatio('Houston Rockets', 'Rockets')
```

90

```
# Partial string comparison with different order  
fuzz.WRatio('Houston Rockets vs Los Angeles Lakers', 'Lakers vs Rockets')
```

86

# Comparison with arrays

```
# Import process
from fuzzywuzzy import process

# Define string and array of possible matches
string = "Houston Rockets vs Los Angeles Lakers"
choices = pd.Series(['Rockets vs Lakers', 'Lakers vs Rockets',
                    'Houson vs Los Angeles', 'Heat vs Bulls'])

process.extract(string, choices, limit = 2)
```

```
[('Rockets vs Lakers', 86, 0), ('Lakers vs Rockets', 86, 1)]
```

# Collapsing categories with string similarity

## Chapter 2

Use `.replace()` to collapse `"eur"` into `"Europe"`

*What if there are too many variations?*

`"EU"` , `"eur"` , `"Europ"` , `"Europa"` , `"Erope"` , `"Evropa"` ...

**String similarity!**

# Collapsing categories with string matching

```
print(survey['state'].unique())
```

```
categories
```

```
id      state
0    California
1         Cali
2    California
3    Californie
4    Californie
5    California
6    Calefernia
7     New York
8 New York City
...
```

```
state
0 California
1 New York
```

# Collapsing all of the state

```
# For each correct category
for state in categories['state']:
    # Find potential matches in states with typos
    matches = process.extract(state, survey['state'], limit = survey.shape[0])
    # For each potential match match
    for potential_match in matches:
        # If high similarity score
        if potential_match[1] >= 80:
            # Replace typo with correct category
            survey.loc[survey['state'] == potential_match[0], 'state'] = state
```



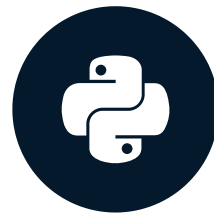
# Record linkage

Event	Time	Event	Time
Houston Rockets vs Chicago Bulls	19:00	NBA: Nets vs Magic	8pm
Miami Heat vs Los Angeles Lakers	19:00	NBA: Bulls vs Rockets	9pm
Brooklyn Nets vs Orlando Magic	20:00	NBA: Heat vs Lakers	7pm
Denver Nuggets vs Miami Heat	21:00	NBA: Grizzlies vs Heat	10pm
San Antonio Spurs vs Atlanta Hawks	21:00	NBA: Heat vs Cavaliers	9pm

**Let's practice!**  
CLEANING DATA IN PYTHON

# Generating pairs

CLEANING DATA IN PYTHON



**Adel Nehme**

Content Developer @ DataCamp

# Motivation

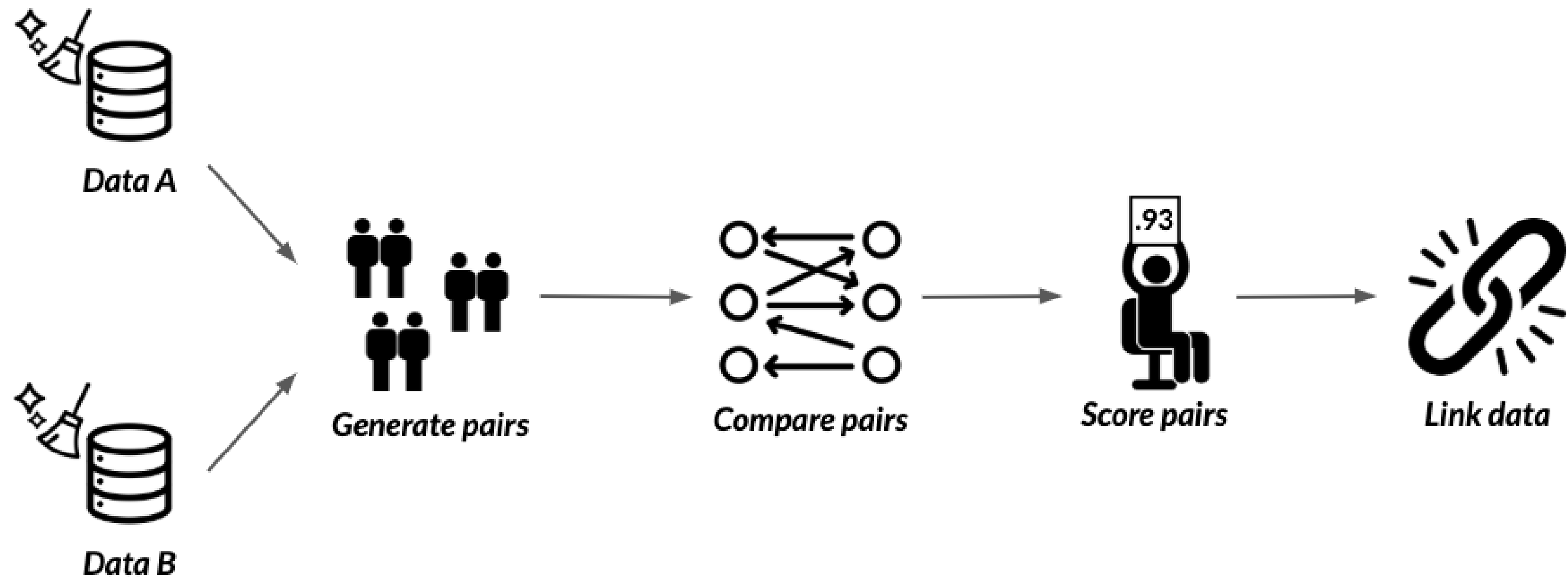
Event	Time
Houston Rockets vs Chicago Bulls	19:00
Miami Heat vs Los Angeles Lakers	19:00
Brooklyn Nets vs Orlando Magic	20:00
Denver Nuggets vs Miami Heat	21:00
San Antonio Spurs vs Atlanta Hawks	21:00

Event	Time
NBA: Nets vs Magic	8pm
NBA: Bulls vs Rockets	9pm
NBA: Heat vs Lakers	7pm
NBA: Grizzlies vs Heat	10pm
NBA: Heat vs Cavaliers	9pm

# When joins won't work

Event	Time	Event	Time
Houston Rockets vs Chicago Bulls	19:00	NBA: Nets vs Magic	8pm
Miami Heat vs Los Angeles Lakers	19:00	NBA: Bulls vs Rockets	9pm
Brooklyn Nets vs Orlando Magic	20:00	NBA: Heat vs Lakers	7pm
Denver Nuggets vs Miami Heat	21:00	NBA: Grizzlies vs Heat	10pm
San Antonio Spurs vs Atlanta Hawks	21:00	NBA: Heat vs Cavaliers	9pm

# Record linkage



The `recordlinkage` package

# Our DataFrames

census\_A

```
      given_name  surname date_of_birth      suburb state address_1
rec_id
rec-1070-org    michaela  neumann    19151111  winston hills    cal  stanley street
rec-1016-org    courtney  painter    19161214    richlands    txs  pinkerton circuit
...
```

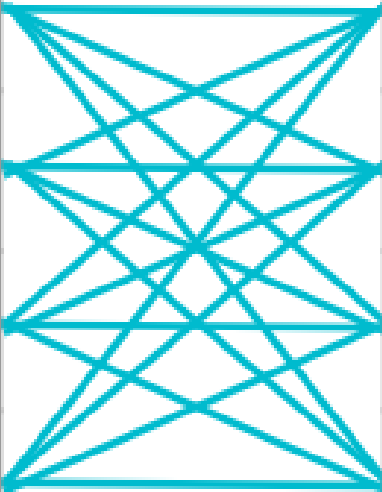
census\_B

```
      given_name  surname date_of_birth      suburb state address_1
rec_id
rec-561-dup-0      elton     NaN    19651013  windermere    ny  light setreet
rec-2642-dup-0  mitchell  maxon    19390212  north ryde    cal  edkins street
...
```

# Generating pairs

census\_A

rec_id	given_name	...	state
...	...	...	...
...	...	...	...
...	...	...	...
...	...	...	...

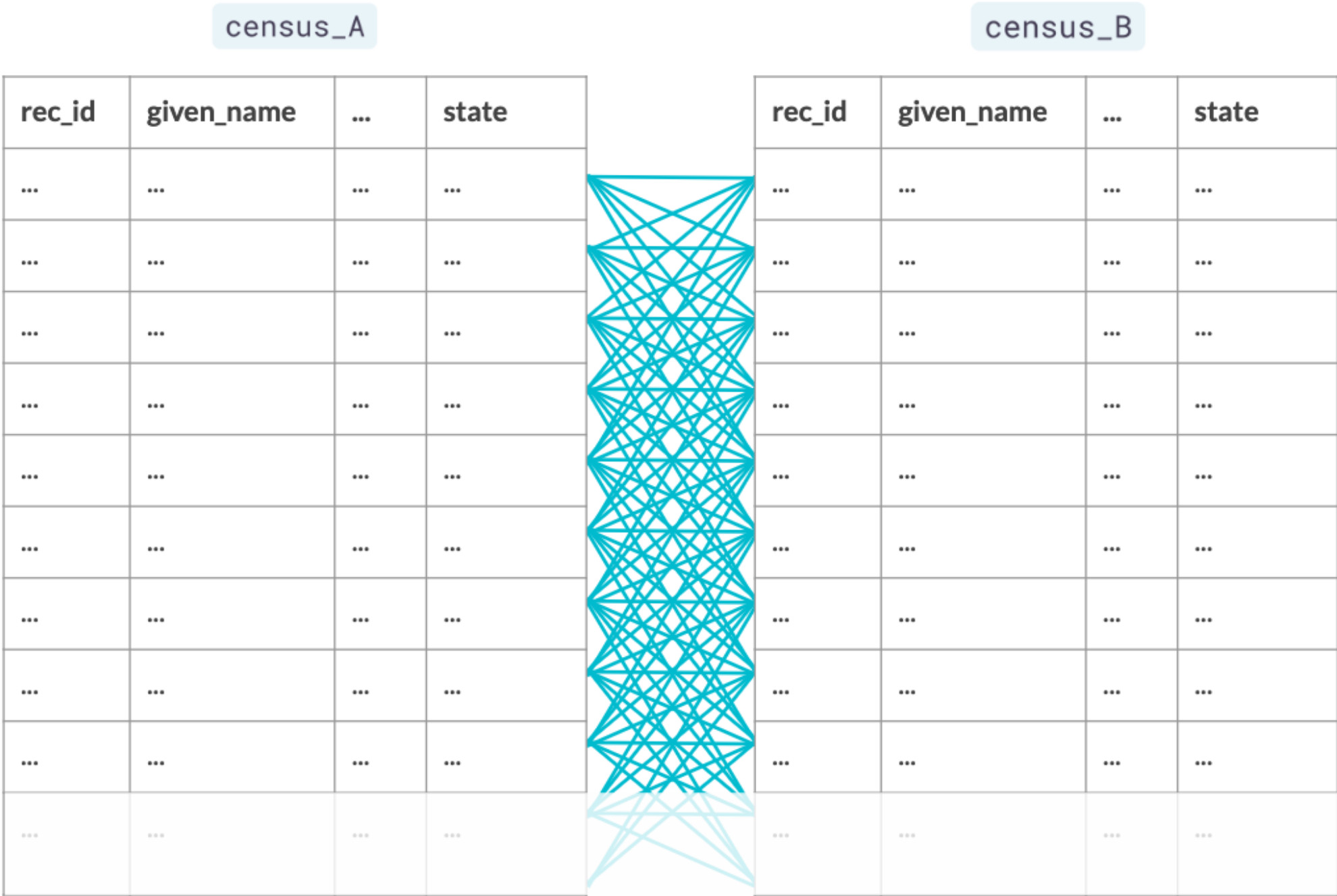


census\_B

rec_id	given_name	...	state
...	...	...	...
...	...	...	...
...	...	...	...
...	...	...	...



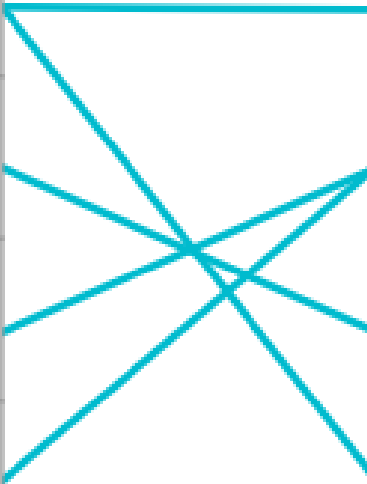
# Generating pairs



# Blocking

census\_A

rec_id	given_name	...	state
...	...	...	cal
...	...	...	ny
...	...	...	txs
...	...	...	txs



census\_A

rec_id	given_name	...	state
...	...	...	cal
...	...	...	txs
...	...	...	ny
...	...	...	cal

# Generating pairs

```
# Import recordlinkage
import recordlinkage

# Create indexing object
indexer = recordlinkage.Index()

# Generate pairs blocked on state
indexer.block('state')
pairs = indexer.index(census_A, census_B)
```

# Generating pairs

```
print(pairs)
```

```
MultiIndex(levels=[['rec-1007-org', 'rec-1016-org', 'rec-1054-org', 'rec-1066-org',  
'rec-1070-org', 'rec-1075-org', 'rec-1080-org', 'rec-110-org', 'rec-1146-org',  
'rec-1157-org', 'rec-1165-org', 'rec-1185-org', 'rec-1234-org', 'rec-1271-org',  
'rec-1280-org',.....  
66, 14, 13, 18, 34, 39, 0, 16, 80, 50, 20, 69, 28, 25, 49, 77, 51, 85, 52, 63, 74, 61,  
83, 91, 22, 26, 55, 84, 11, 81, 97, 56, 27, 48, 2, 64, 5, 17, 29, 60, 72, 47, 92, 12,  
95, 15, 19, 57, 37, 70, 94]], names=['rec_id_1', 'rec_id_2'])
```

# Comparing the DataFrames

```
# Generate the pairs
pairs = indexer.index(census_A, census_B)

# Create a Compare object
compare_cl = recordlinkage.Compare()

# Find exact matches for pairs of date_of_birth and state
compare_cl.exact('date_of_birth', 'date_of_birth', label='date_of_birth')
compare_cl.exact('state', 'state', label='state')

# Find similar matches for pairs of surname and address_1 using string similarity
compare_cl.string('surname', 'surname', threshold=0.85, label='surname')
compare_cl.string('address_1', 'address_1', threshold=0.85, label='address_1')

# Find matches
potential_matches = compare_cl.compute(pairs, census_A, census_B)
```

# Finding matching pairs

```
print(potential_matches)
```

rec_id_1	rec_id_2	date_of_birth	state	surname	address_1
rec-1070-org	rec-561-dup-0	0	1	0.0	0.0
	rec-2642-dup-0	0	1	0.0	0.0
	rec-608-dup-0	0	1	0.0	0.0
...					
rec-1631-org	rec-4070-dup-0	0	1	0.0	0.0
	rec-4862-dup-0	0	1	0.0	0.0
	rec-629-dup-0	0	1	0.0	0.0
...					

# Finding the only pairs we want

```
potential_matches[potential_matches.sum(axis = 1) == 2]
```

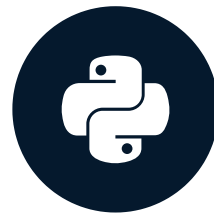
rec_id_1	rec_id_2	date_of_birth	state	surname	address_1
rec-4878-org	rec-4878-dup-0	1	1	1.0	0.0
rec-417-org	rec-2867-dup-0	0	1	0.0	1.0
rec-3964-org	rec-394-dup-0	0	1	1.0	0.0
rec-1373-org	rec-4051-dup-0	0	1	1.0	0.0
	rec-802-dup-0	0	1	1.0	0.0
rec-3540-org	rec-470-dup-0	0	1	1.0	0.0

**Let's practice!**  
CLEANING DATA IN PYTHON



# Linking DataFrames

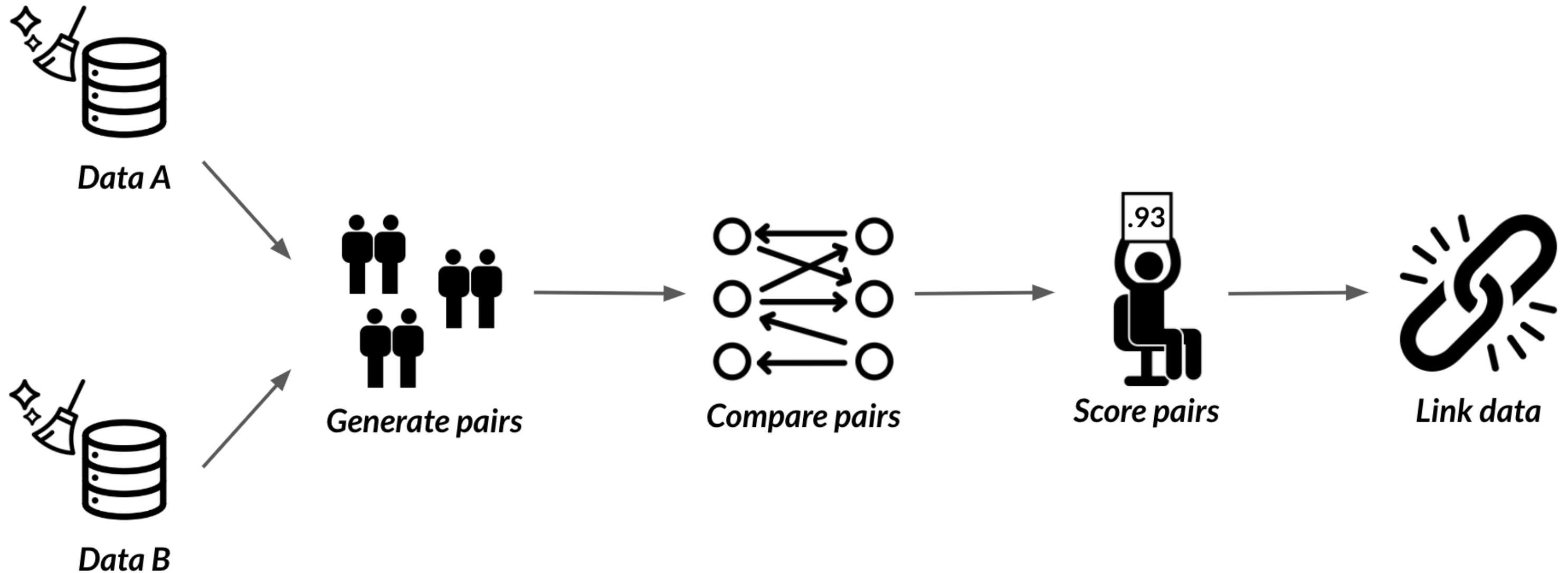
CLEANING DATA IN PYTHON



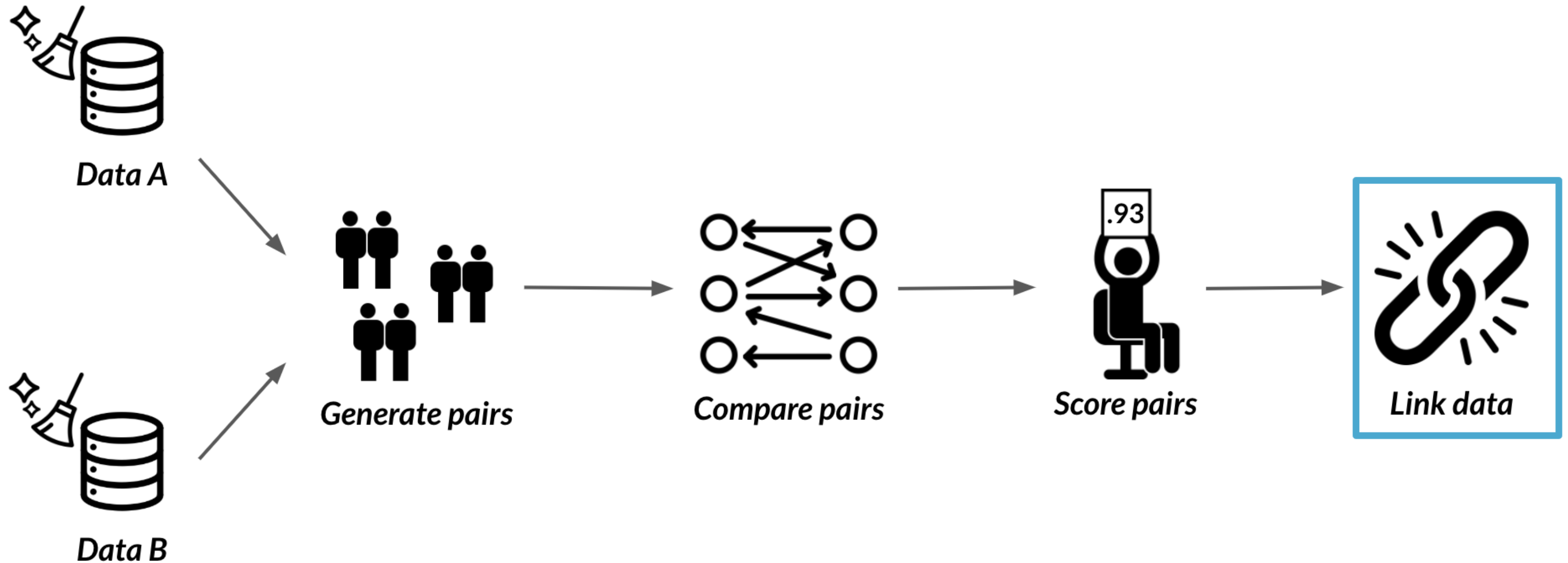
**Adel Nehme**

Content Developer @ DataCamp

# Record linkage



# Record linkage



# Our DataFrames

census\_A

```
      given_name  surname date_of_birth      suburb state address_1
rec_id
rec-1070-org    michaela  neumann    19151111  winston hills    nsw  stanley street
rec-1016-org    courtney  painter    19161214    richlands    vic  pinkerton circuit
...
```

census\_B

```
      given_name  surname date_of_birth      suburb state address_1
rec_id
rec-561-dup-0      elton      NaN    19651013  windermere    vic  light setreet
rec-2642-dup-0  mitchell    maxon    19390212  north ryde    nsw  edkins street
...
```

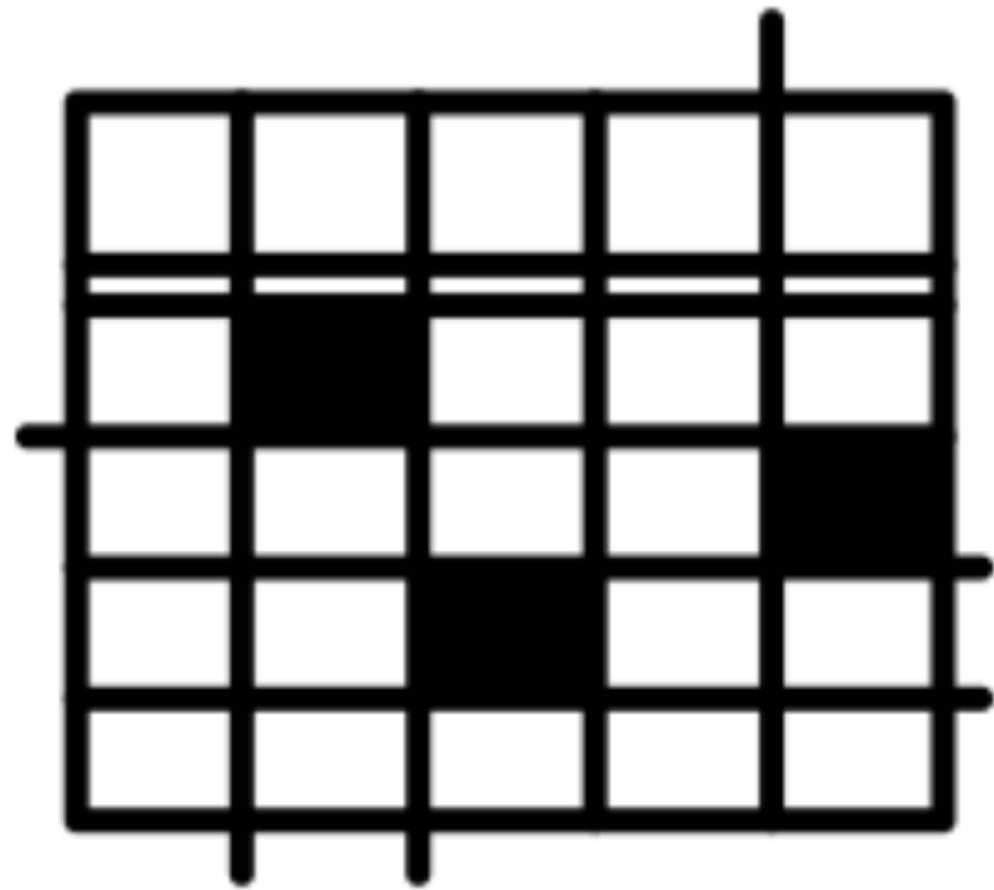
# What we've already done

```
# Import recordlinkage and generate full pairs
import recordlinkage
indexer = recordlinkage.Index()
indexer.block('state')
full_pairs = indexer.index(census_A, census_B)

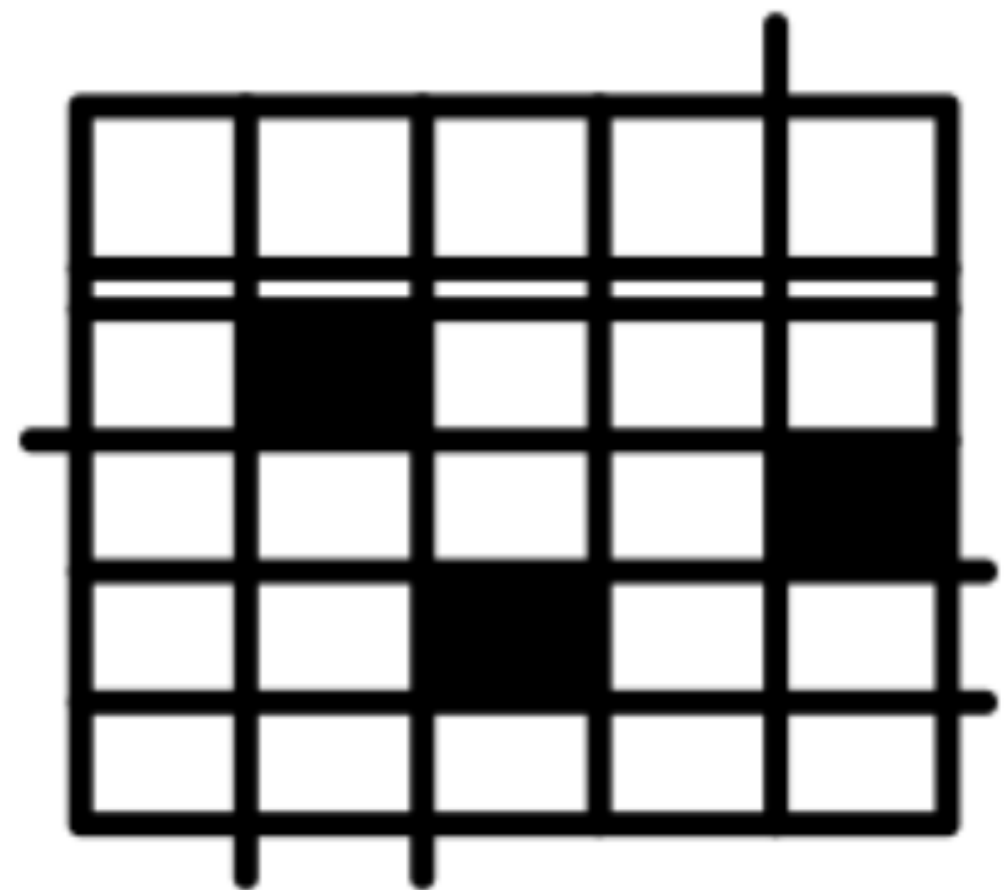
# Comparison step
compare_cl = recordlinkage.Compare()
compare_cl.exact('date_of_birth', 'date_of_birth', label='date_of_birth')
compare_cl.exact('state', 'state', label='state')
compare_cl.string('surname', 'surname', threshold=0.85, label='surname')
compare_cl.string('address_1', 'address_1', threshold=0.85, label='address_1')

potential_matches = compare_cl.compute(full_pairs, census_A, census_B)
```

# What we're doing now



census\_A



census\_B

# Our potential matches

potential\_matches

rec_id_1	rec_id_2	date_of_birth	state	surname	address_1
rec-1070-org	rec-561-dup-0	0	1	0.0	0.0
	rec-2642-dup-0	0	1	0.0	0.0
	rec-608-dup-0	0	1	0.0	0.0
...	...	...	...	...	...
rec-1631-org	rec-1697-dup-0	0	1	0.0	0.0
	rec-4404-dup-0	0	1	0.0	0.0
	rec-3780-dup-0	0	1	0.0	0.0
...	...	...	...	...	...

# Our potential matches

```
potential_matches
```

```
census_A      date_of_birth  state  surname  address_1
rec_id_1  rec_id_2
rec-1070-org  rec-561-dup-0      0      1      0.0      0.0
              rec-2642-dup-0      0      1      0.0      0.0
              rec-608-dup-0      0      1      0.0      0.0
...
rec-1631-org  rec-1697-dup-0      0      1      0.0      0.0
              rec-4404-dup-0      0      1      0.0      0.0
              rec-3780-dup-0      0      1      0.0      0.0
...
...      ...      ...      ...      ...
```



# Our potential matches

```
potential_matches
```

<b>census_A</b>	<b>census_B</b>	date_of_birth	state	surname	address_1
rec_id_1	rec_id_2				
rec-1070-org	rec-561-dup-0	0	1	0.0	0.0
	rec-2642-dup-0	0	1	0.0	0.0
	rec-608-dup-0	0	1	0.0	0.0
...		...	...	...	...
rec-1631-org	rec-1697-dup-0	0	1	0.0	0.0
	rec-4404-dup-0	0	1	0.0	0.0
	rec-3780-dup-0	0	1	0.0	0.0
...		...	...	...	...

# Our potential matches

potential\_matches

census_A	census_B	date_of_birth	state	surname	address_1
rec_id_1	rec_id_2				
rec-1070-org	rec-561-dup-0	0	1	0.0	0.0
	rec-2642-dup-0	<u>0</u>	<u>1</u>	<u>0.0</u>	<u>0.0</u>
	rec-608-dup-0	0	1	0.0	0.0
...		...	...	...	...
rec-1631-org	rec-1697-dup-0	0	1	0.0	0.0
	rec-4404-dup-0	0	1	0.0	0.0
	rec-3780-dup-0	0	1	0.0	0.0
...		...	...	...	...

# Probable matches

```
matches = potential_matches[potential_matches.sum(axis = 1) >= 3]
print(matches)
```

		date_of_birth	state	surname	address_1
rec_id_1	rec_id_2				
rec-2404-org	rec-2404-dup-0	1	1	1.0	1.0
rec-4178-org	rec-4178-dup-0	1	1	1.0	1.0
rec-1054-org	rec-1054-dup-0	1	1	1.0	1.0
...	...	...	...	...	...
rec-1234-org	rec-1234-dup-0	1	1	1.0	1.0
rec-1271-org	rec-1271-dup-0	1	1	1.0	1.0

# Probable matches

```
matches = potential_matches[potential_matches.sum(axis = 1) >= 3]
print(matches)
```

	<b>census_B</b>	date_of_birth	state	surname	address_1
rec_id_1	rec_id_2				
rec-2404-org	rec-2404-dup-0	1	1	1.0	1.0
rec-4178-org	rec-4178-dup-0	1	1	1.0	1.0
rec-1054-org	rec-1054-dup-0	1	1	1.0	1.0
...	...	...	...	...	...
rec-1234-org	rec-1234-dup-0	1	1	1.0	1.0
rec-1271-org	rec-1271-dup-0	1	1	1.0	1.0

# Get the indices

```
matches.index
```

```
MultiIndex(levels=[['rec-1007-org', 'rec-1016-org', 'rec-1054-org', 'rec-1066-org',  
'rec-1070-org', 'rec-1075-org', 'rec-1080-org', 'rec-110-org', ...
```

```
# Get indices from census_B only  
duplicate_rows = matches.index.get_level_values(1)  
print(census_B_index)
```

```
Index(['rec-2404-dup-0', 'rec-4178-dup-0', 'rec-1054-dup-0', 'rec-4663-dup-0',  
      'rec-485-dup-0', 'rec-2950-dup-0', 'rec-1234-dup-0', ... , 'rec-299-dup-0'])
```

# Linking DataFrames

```
# Finding duplicates in census_B
census_B_duplicates = census_B[census_B.index.isin(duplicate_rows)]
```

```
# Finding new rows in census_B
census_B_new = census_B[~census_B.index.isin(duplicate_rows)]
```

```
# Link the DataFrames!
full_census = census_A.append(census_B_new)
```

```
# Import recordlinkage and generate pairs and compare across columns
...
# Generate potential matches
potential_matches = compare_cl.compute(full_pairs, census_A, census_B)

# Isolate matches with matching values for 3 or more columns
matches = potential_matches[potential_matches.sum(axis = 1) >= 3]

# Get index for matching census_B rows only
duplicate_rows = matches.index.get_level_values(1)

# Finding new rows in census_B
census_B_new = census_B[~census_B.index.isin(duplicate_rows)]

# Link the DataFrames!
full_census = census_A.append(census_B_new)
```

**Let's practice!**  
CLEANING DATA IN PYTHON



# Congratulations!

CLEANING DATA IN PYTHON



**Adel Nehme**

Content Developer @ DataCamp

# What we've learned



Diagnose dirty  
data



Side effects of  
dirty data



Clean data

# What we've learned



**Data Type  
Constraints**

*Strings  
Numeric data*

...



**Data Range  
Constraints**

*Out of range data  
Out of range dates*

...



**Uniqueness  
Constraints**

*Finding duplicates  
Treating them*

...

## Chapter 1 - Common data problems

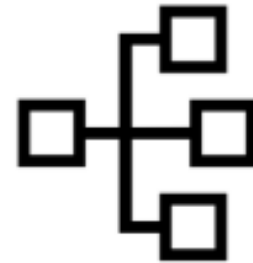
# What we've learned



## Membership Constraints

*Finding inconsistent categories  
Treating them with joins*

...



## Categorical Variables

*Finding inconsistent categories  
Collapsing them into less*

...



## Cleaning Text Data

*Unifying formats  
Finding lengths*

...

## Chapter 2 - Text and categorical data problems

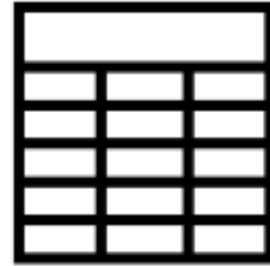
# What we've learned



## *Uniformity*

*Unifying currency formats*  
*Unifying date formats*

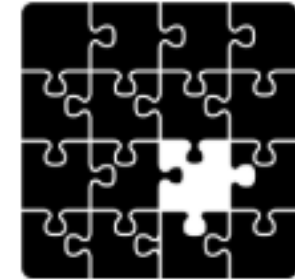
...



## *Cross field validation*

*Summing across rows*  
*Building assert functions*

...



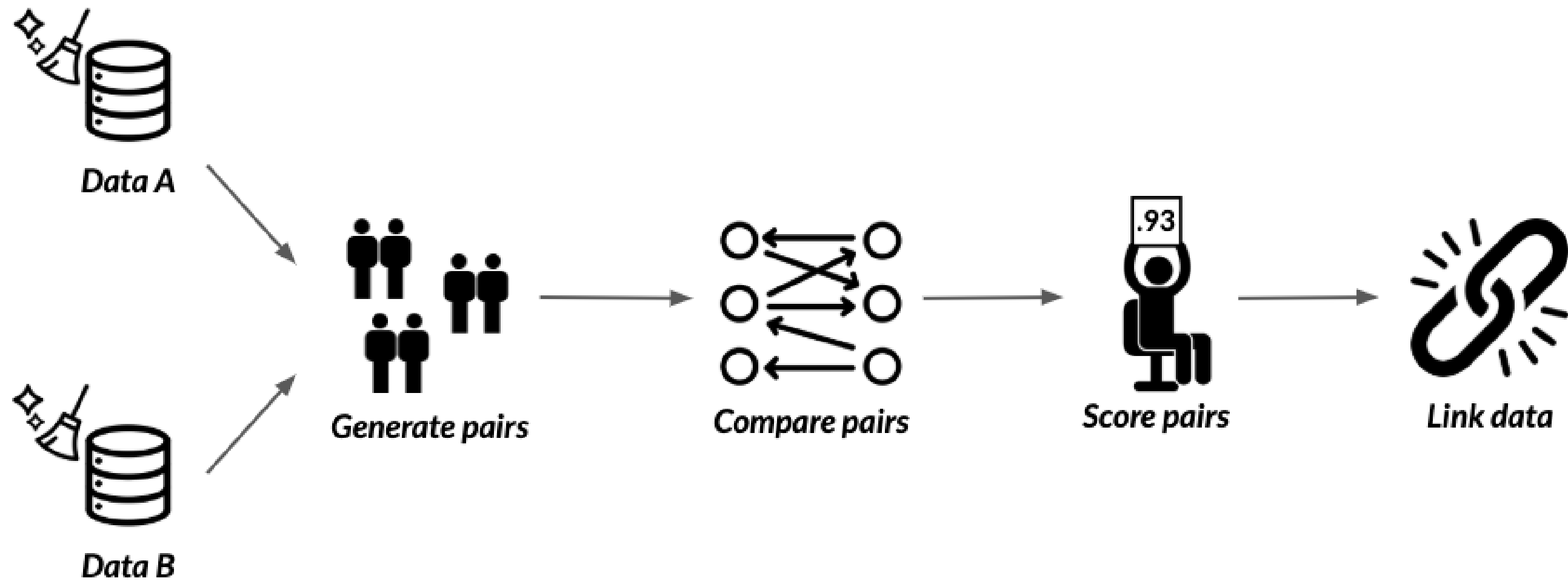
## *Completeness*

*Finding missing data*  
*Treating them*

...

## Chapter 3 - Advanced data problems

# What we've learned

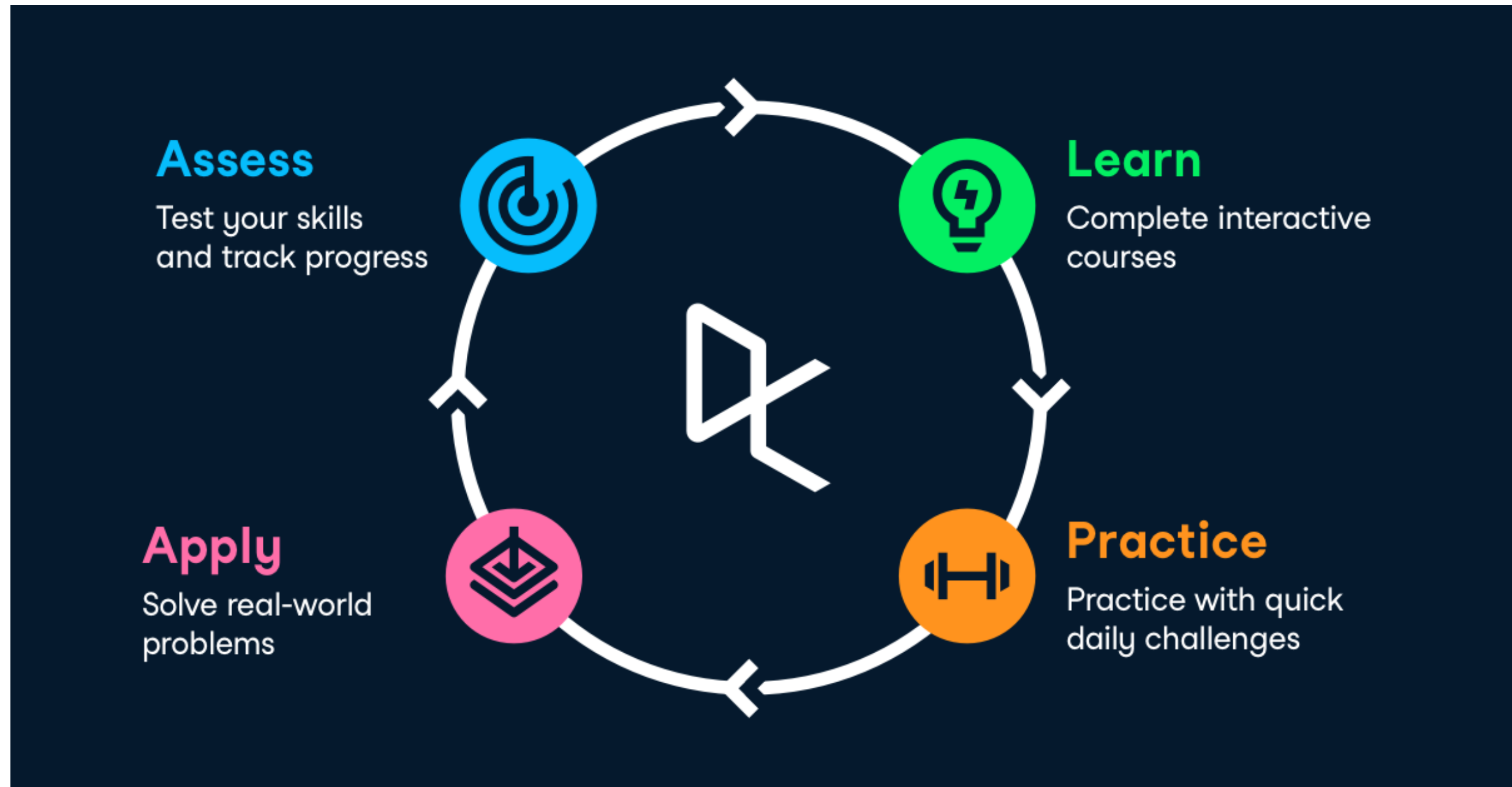


## Chapter 4 - Record linkage

# More to learn on DataCamp!

- [Working with Dates and Times in Python](#)
- [Regular Expressions in Python](#)
- [Dealing with Missing Data in Python](#)
- And more!

# More to learn!





# More to learn!



**Thank you!**  
CLEANING DATA IN PYTHON