matchtools Documentation

Release 0.1.2

Anton Kupenko & Dawid Kaczmarski

May 24, 2017

Contents

Contents:

CHAPTER 1

API

class `matchtools.`**`MatchBlock`**(*entry*, ***, *try_date=True*, *try_coordinates=True*, *try_str_number=True*, *try_str_custom=True*, *convert_roman=True*)

Core class that contains all methods for data extraction, processing, and comparison.

classmethod **`compare_coordinates`**(*coords1*, *coords2*, **args*, *tolerance=None*, *unit='km'*, ***kwargs*)

Check if a distance between the pairs of coordinates provided is within the specified tolerance.

Return True if yes, otherwise return False.

Use geopy (https://pypi.python.org/pypi/geopy).

Try to use Vincenty formula, if error occurs use Great Circle formula.

Parameters

- **`coords1`** – pair of coordinates - a tuple of two numbers
- **`coords2`** – pair of coordinates - a tuple of two numbers
- **`tolerance`** – number
- **`unit`** – str, one of: 'kilometers', 'km', 'meters', 'm', 'miles', 'mi', 'feet', 'ft', 'nautical', 'nm'

Return type bool

Example

```
>>> a, b = (36.1332600, -5.4505100), (35.8893300, -5.3197900)
>>> MatchBlock.compare_coordinates(a, b, tolerance=20, unit='mi')
True
```

```
>>> a, b = (36.1332600, -5.4505100), (35.8893300, -5.3197900)
>>> MatchBlock.compare_coordinates(a, b, tolerance=1, unit='mi')
False
```

classmethod compare_dates (*date1*, *date2*, *, *tolerance=None*, *pattern='%d-%b-%Y'*)

Check if the dates provided are within the specified tolerance.

Return True if yes, otherwise return False.

If lists of dates are provided check if they are of the same length.

If yes, check whether the difference between each element of list1 and the corresponding element of list2 is within the specified tolerance.

Parameters

- **date1** – datetime.datetime object or a list of such objects
- **date2** – datetime.datetime object or a list of such objects
- **tolerance** – number
- **pattern** – str

Return type bool

Example

```
>>> date1 = datetime.datetime(2005, 5, 25, 0, 0)
>>> date2 = datetime.datetime(2005, 5, 26, 0, 0)
>>> MatchBlock.compare_dates(date1, date2, tolerance=1)
True
```

```
>>> date1 = [datetime.datetime(2005, 5, 26, 0, 0)]
>>> date2 = [datetime.datetime(2006, 5, 25, 0, 0)]
>>> MatchBlock.compare_dates(date1, date2, tolerance=1)
False
```

classmethod compare_numbers (*number1*, *number2*, *, *tolerance=None*)

Check if the numbers provided are within the specified tolerance.

Return True if yes, otherwise return False.

Parameters

- **number1** – number
- **number2** – number
- **tolerance** – number

Return type bool

Example

```
>>> MatchBlock.compare_numbers(1, 10, tolerance=10)
True
```

```
>>> MatchBlock.compare_numbers(1, 10, tolerance=5)
False
```

classmethod compare_strings (*string1*, *string2*, *, *tolerance=None*, *method='uwratio'*)

Check if the strings provided have a similarity ratio within the specified tolerance.

Return True if yes, otherwise return False.

Use fuzzywuzzy (https://pypi.python.org/pypi/fuzzywuzzy).

Parameters

- **string1** – str
- **string2** – str
- **tolerance** – number
- **method** – str, one of: 'uwratio', 'partial_ratio', 'token_sort_ratio', 'token_set_ratio', 'ratio'

Return type bool

Example

```
>>> MatchBlock.compare_strings('Beatles', 'The Beatles', tolerance=10)
True
```

```
>>> MatchBlock.compare_strings('AB', 'AC', tolerance=0, method='ratio')
False
```

classmethod dict_sub(*string*, *dictionary_file=None*)

Substitute string values with values from a dictionary.

Replace part of the string, separated by non-alphanumeric characters, with a key found in a dictionary, if the string part is contained within values of the dictionary's key.

The dictionary must be stored in the JSON format. Use the file provided with the package by default.

Parameters

- **string** – str
- **dictionary_file** – str

Return type str

Example

```
>>> MatchBlock.dict_sub('S Africa')
'south Africa'
```

classmethod extract_coordinates(*string*)

Extract pair of coordinates (latitude and longitude, separated by comma) from a string.

If found, return remains of original string and tuple with coordinates, otherwise return original string and None.

Parameters **string** – str

Return type tuple

Example

```
>>> MatchBlock.extract_coordinates('Washington 38.8897, -77.0089')
('Washington', (38.8897, -77.0089))
```

```
>>> MatchBlock.extract_coordinates('55.7522200,37.6155600')
('', (55.75222, 37.61556))
```

```
>>> MatchBlock.extract_coordinates('Richmond 123.45')
('Richmond 123.45', None)
```

classmethod extract_dates (*string*)

Extract dates from text.

Use datefinder (https://pypi.python.org/pypi/datefinder).

Parameters **string** – str

Return type tuple

Example

```
>>> MatchBlock.extract_dates('Istanbul 25 May 2005 ')
('Istanbul', [datetime.datetime(2005, 5, 25, 0, 0)])
```

classmethod extract_str_custom (*string*, *dictionary_file=None*)

Extract all custom values found in a string.

Look up the values in the supplied dictionary's keys. First prepare the string by substituting values found in the dictionary's values with corresponding key using dict_sub function.

Return string and its separated custom parts.

Parameters

- **string** – str
- **dictionary_file** – str

Return type tuple

Example

```
>>> MatchBlock.extract_str_custom('East Timor')
('Timor', 'east')
```

```
>>> MatchBlock.extract_str_custom('Sud Ouest France')
('France', 'south west')
```

classmethod extract_str_number (*string*)

Extract all numeric elements found in a string. Consider an element to be a numeric if it contains at least one digit.

Return string and its separated numeric parts.

Parameters **string** – str

Return type tuple

Example

```
>>> MatchBlock.extract_str_number('Jamaica 1')
('Jamaica', '1')
```

```
>>> MatchBlock.extract_str_number('Jamaica 1X')
('Jamaica', '1X')
```

```
>>> MatchBlock.extract_str_number('Jamaica')
('Jamaica', '')
```

classmethod from_roman (*string*)

Convert the whole input string from roman to arabic numeral.

Use roman (https://pypi.python.org/pypi/roman).

Return result if conversion was successful, otherwise return input.

Parameters **string** – str

Return type str

Example

```
>>> MatchBlock.from_roman('VII')
'7'
```

```
>>> MatchBlock.from_roman('XIIIIX')
'XIIIIX'
```

```
>>> MatchBlock.from_roman('ABC')
'ABC'
```

classmethod **integers_to_roman** (*string*)

Convert all integers within the string into roman numerals.

Recognise integers separated by non-alphanumeric characters.

Parameters **string** – str

Return type str

Example

```
>>> MatchBlock.integers_to_roman('LIV 4 DOR 3')
'LIV IV DOR III'
```

classmethod **is_abbreviation** (*string1*, *string2*)

Check whether one string is an abbreviation of the other.

Parameters

- **string1** – str
- **string2** – str

Return type bool

Example

```
>>> MatchBlock.is_abbreviation('Federal Bureau of Investigation', 'FBI')
True
```

classmethod **roman_to_integers** (*string*)

Convert all roman numerals within the string into integers.

Recognise roman numerals separated by non-alphanumeric characters.

Parameters **string** – str

Return type str

Example

```
>>> MatchBlock.roman_to_integers('IV ABC II')
'4 ABC 2'
```

classmethod **split_on_nonalpha** (*string*, *return_all=True*)

Split the input string into a list of alphanumeric and non-alphanumeric components.

If return_all is False return list with alphanumeric components of the string only.

Parameters

- **string** – str
- **return_all** – bool

Return type list

Example

```
>>> MatchBlock.split_on_nonalpha('F.C. Liverpool', return_all=True)
['F', '.', 'C', '. ', 'Liverpool']
```

```
>>> MatchBlock.split_on_nonalpha('F.C. Liverpool', return_all=False)
['F', 'C', 'Liverpool']
```

classmethod strip_zeros (*string*)

Strip leading zeros in any number longer than one digit found in a string.

Parameters **string** – str

Return type str

Example

```
>>> MatchBlock.strip_zeros('Agent 007')
'Agent 7'
```

matchtools.**return_element** (*word*, *element*)

Split word and return the index of element.

If element occurs more than once in the word, an index of the first instance is returned.

Parameters

- **word** – str
- **element** – str

Return type int

Example

```
>>> return_element('South America', 'America')
1
```

matchtools.**match_rows** (*row1*, *row2*)

Compare rows by transforming each pair of values into MatchBlock objects and perform equality check on them.

The rows are considered to match if all checks result in True.

Parameters

- **row1** – list, tuple
- **row2** – list, tuple

Return type bool

Example

```
>>> row1 = ['Flight 1', 5, '1 May 2015']
>>> row2 = ['Flight 01', 5, '2015-05-01']
>>> match_rows(row1, row2)
True
```

```
>>> row1 = ['Flight 2', 5, '1 May 2015']
>>> row2 = ['Flight 02', 6, '2015-05-01']
>>> match_rows(row1, row2)
False
```

`matchtools.`**`match_find`**(*row*, *rows*)

Search list of rows and return first successful match with the input row.

Parameters

- **`row`** – list, tuple
- **`rows`** – nested list, nested tuple

Return type list

Example

```
>>> row = ['Flight 3', 100]
>>> rows = [['Flight 1', 100], ['Flight 2', 100], ['Flight 3', 100]]
>>> match_find(row, rows)
['Flight 3', 100]
```

`matchtools.`**`match_find_all`**(*row*, *rows*)

Search list of rows and return all successful matches with the input row.

Parameters

- **`row`** – list, tuple
- **`rows`** – nested list, nested tuple

Return type list

Example

```
>>> row = ['Flight 2', 100]
>>> rows = [['Flight 1', 100], ['Flight 2', 100], ['Flight 2', 100]]
>>> match_find_all(row, rows)
[['Flight 2', 100], ['Flight 2', 100]]
```

`matchtools.`**`move_element_to_front`**(*word*, *element*)

Move element of a word to front.

If a string is used as element, return_element function is triggered to determine the position of element within the word.

If an integer is used as element, it must reflect the position of element within the word split by non-alphanumeric character e.g. 'Block-A 1' -> ['Block, 'A', '1']

The function converts all sequences of non-alphanumeric characters into single whitespaces.

Parameters

- **`word`** – str
- **`element`** – str (gets converted to int by return_element()) or int

Return type str

Example

```
>>> move_element_to_front('A B C', 2)
'C A B'
```

`matchtools.move_element_to_back(word, element)`

Move element of a word to back.

If a string is used as element, return_element function is triggered to determine the position of element within the word.

If an integer is used as element, it must reflect the position of element within the word split by non-alphanumeric character e.g. 'Block-A 1' -> ['Block, 'A', '1']

The function converts all sequences of non-alphanumeric characters into single whitespaces.

Parameters

- **word** – str
- **element** – str (gets converted to int by return_element()) or int

Return type str

Example

```
>>> move_element_to_back('A B C', 0)
'B C A'
```

CHAPTER 2

Cookbook

Matchtools is a package written to streamline data matching and integration processes. This document contains an overview of the package's functionalities. It is divided into three parts:

1. Data manipulation.

2. Compare single records.

3. Compare multiple records.

The first part presents how to use the package to perform some specific data operations. Integrating data from different sources often requires similar set of normalising steps - standardising numbers, removing linguistic or conventional differences etc. Matchtools package provides functionalities to automate that.

The second part introduces the matchtools methodology of finding matching records in different sets. It includes two examples of how to use the package to compare single records - e.g. checking whether two strings or lists can be considered as equal. It also shows how to set up different types of tolerance.

The third part shows how to use matchtools to find matching records within nested lists.

```
>>> from matchtools import *
```

1. Data manipulation

Working with roman numerals

Convert roman numerals to numbers:

```
>>> caesar = 'Pro multitudine autem hominum et pro gloria belli atque fortitudinis␣
↪angustos se fines habere arbitrabantur, qui in longitudinem milia passuum CCXL, in␣
↪latitudinem CLXXX patebant.'
>>> MatchBlock.roman_to_integers(caesar)
'Pro multitudine autem hominum et pro gloria belli atque fortitudinis angustos se␣
↪fines habere arbitrabantur, qui in longitudinem milia passuum 240, in latitudinem␣
↪180 patebant.'
```

Convert numbers to roman numerals:

```
>>> caesar = 'They thought, that considering the extent of their population, and␣
↪their renown for warfare and bravery, they had but narrow limits, although they␣
↪extended in length 240, and in breadth 180 [Roman] miles.'
>>> MatchBlock.integers_to_roman(caesar)
'They thought, that considering the extent of their population, and their renown for␣
↪warfare and bravery, they had but narrow limits, although they extended in length␣
↪CCXL, and in breadth CLXXX [Roman] miles.'
```

Working with zeros

One of the most common data manipulation tasks when working on data integration. It is often the case that 'Record 1' and 'Record 0001' refer to the same object:

```
>>> record_with_zeros = 'ABC 001 DEF 2 GHI 03'
>>> MatchBlock.strip_zeros(record_with_zeros)
'ABC 1 DEF 2 GHI 3'
```

Checking if a word is an abbreviation of another word

```
>>> MatchBlock.is_abbreviation('Federal Bureau of Investigation', 'FBI')
True
```

Replacing words with their standardised forms

When integrating data coming from different sources, some linguistic, spelling or conventional inconsistencies are likely to occur. Such situation often takes place when data contain cardinal directions. For example, it probably makes sense to standardize *vest*, *w*, as well as *zapad* (rus.) or *ouest* (fr.) as *west*. The package uses a predefined set of standardised forms, see the documentation of **MatchBlock.dict_sub** to learn how to provide a user-defined one.

```
>>> MatchBlock.dict_sub("there's a feeling I get when I look to the W")
"there's a feeling I get when I look to the west"
```

Moving elements of a string

Matchtools package contains functions to move text elements to the beginning or the end of a string. You can specify the element to move by its name or position:

```
>>> move_element_to_front('London E1 United Kingdom', 1)
'E1 London United Kingdom'
```

```
>>> move_element_to_back('London E1 United Kingdom', 'E1')
'London United Kingdom E1'
```

Example: standardising names in pandas DataFrame

```
from matchtools import MatchBlock, move_element_to_back
import pandas as pd

input_data = [('nord IV N1'), ('west 3 W001'), ('e 02 E01'), ('sud 1 S1')]

df = pd.DataFrame(input_data, columns = ['Name'])

def standardize(element):
    element = move_element_to_back(element, 1)
    element = MatchBlock.dict_sub(element)
    element = MatchBlock.roman_to_integers(element)
    element = MatchBlock.strip_zeros(element)
```

```
    return element

df['Standardized'] = df.apply(lambda row: standardize(row['Name']), axis=1)

print(df)
```

```
          Name Standardized
0   nord IV N1   north N1 4
1  west 3 W001    west W1 3
2     e 02 E01    east E1 2
3     sud 1 S1   south S1 1
```

2. Compare single records

Specifying tolerance

Specifying tolerance for each data type is a crucial part of the process. This is how we define what similarity criteria two **MatchBlock** objects must fulfil in order to be considered as equal. **MatchBlock** class allows the following tolerances:

property name	description
number_tolerance	expressed in numbers. No maximum value. Default: 0.
date_tolerance	expressed in numbers (days). No maximum value. Default: 0.
coordinates_tolerance	expressed in numbers (kilometers, see the documentation MatchBlock.compare_coordinates to learn how to use different units. No maximum value. Default: 0.
string_tolerance	expressed in numbers (Levenshtein distance when calculating uwratio from fuzzywuzzy package, see the documentation of MatchBlock.compare_strings to learn how to use different algorithms). Maximum value: 100. Default: 0
str_number_tolerance	Same as string_tolerance. Used only for the numeric components of a string

Example 1: Comparing single MatchBlock objects

This is a basic example of matchtool's main functionality. It shows how to determine whether two string objects are the same, given the tolerances specified.

Note: Comparing two **MatchBlock** objects triggers the following data manipulation methods on both of them, there's no need to execute them before the comparison: **roman_to_integers**, **strip_zeros**, **is_abbreviation**, **dict_sub**.

```
>>> object1 = MatchBlock('WOJCIOW 11 DEV 07-NOV-86')
>>> object2 = MatchBlock('WOJCIOW 12 DEV 01-NOV-86')
>>> object1
<MatchBlock object at 0x105d43080: date: 1986-11-07, string: WOJCIOW DEV, string␣
↪(number part): 11>
>>> object2
<MatchBlock object at 0x1063c1e80: date: 1986-11-01, string: WOJCIOW DEV, string␣
↪(number part): 12>
```

We created two **MatchBlock** objects. We can see how the input string has been split into date, text and text-number components. Now, let's set some tolerance values and perform a comparison:

```
>>> MatchBlock.date_tolerance = 7
>>> MatchBlock.number_tolerance = 0
>>> MatchBlock.str_number_tolerance = 0
>>> object1 == object2
False
```

We can see that the objects are considered as different. While the tolerance set for dates is probably high enough it looks that there is still too much difference in the numeric components of the objects:

```
>>> MatchBlock.number_tolerance = 1
>>> object1 == object2
False
```

Still false. This is because the numeric parts of the objects come from a string, not an integer or float. Therefore we need to specify the **str_number_tolerance** appropriately. A thing to remember, **str_number_tolerance** is a Levenshtein distance tolerance. That's why setting it to 1 wouldn't be enough in this case. We use **number_tolerance** when working with numbers that are not extracted from strings and that tolerance is simply a distance between numbers in integers. The next section includes such objects.

```
>>> MatchBlock.str_number_tolerance = 50
>>> object1 == object2
True
```

Example 2: Comparing two lists

In a real work situation you will probably want to perform more complex analysis. For example, you may want to determine whether a record from Database A is equal to a record from Database B. This can be achieved with **match_rows** function.

```
>>> record_1 = ['London 1', 5, '1 May 2015']
>>> record_2 = ['London_01', 10, '2015-05-01']
>>> MatchBlock.number_tolerance = 10
>>> match_rows(record_1, record_2)
True
```

3. Compare multiple records

Matchtools include two functions to perform matching on a list of records:

- **match_find** takes an input record, compares it to a set of records and returns the first matching object
- **match_find_all** does the same but returns a list of all matching objects

```
from matchtools import MatchBlock, match_find, match_find_all

MatchBlock.number_tolerance = 10
MatchBlock.date_tolerance = 5
MatchBlock.coordinates_tolerance = 0
MatchBlock.string_tolerance = 0
MatchBlock.str_number_tolerance = 0

record_1 = ['Flight 3', 5, '1 May 2015', '52.3740300, 4.8896900']

records = [['Flight 1', 0, '3 May 2015', '52.3740300, 4.8896900'],
           ['Flight 2', 5, '4 May 2016', '52.3740300, 4.8896900'],
```

```
            ['Flight 3', 10, '5 May 2015', '52.3740300, 4.8896900'],
            ['Flight 3', 15, '6 May 2015', '52.3740300, 4.8896900']]
```

```
>>> match_find(record_1, records)
['Flight 3', 10, '5 May 2015', '52.3740300, 4.8896900']
```

```
>>> match_find_all(record_1, records)
[['Flight 3', 10, '5 May 2015', '52.3740300, 4.8896900'], ['Flight 3', 15, '6 May 2015
↪', '52.3740300, 4.8896900']]
```

CHAPTER 3

Indices and tables

- genindex

Python Module Index

m

Index