
miners_doc Documentation

Release latest

Dashboard Team

Sep 22, 2020

Contents

1	Setup	1
2	Contributing	3
3	Reference	5
3.1	Conversions module	5
3.2	Hierarchy module	7
3.3	Transformations module	9
3.4	Utility module	13
	Python Module Index	15
	Index	17

CHAPTER 1

Setup

To install the package, run either of the following:

```
$ pip install pandas-extras  
$ pipenv install pandas-extras
```


CHAPTER 2

Contributing

If you wish to help in developing this package further, any PRs are more that welcome!

3.1 Conversions module

Contains function that help in converting between types

```
class pandas_extras.conversions.NativeDict (*args, **kwargs)
```

Bases: dict

Helper class to ensure that only native types are in the dicts produced by `to_dict()`

```
>>> df.to_dict(orient='records', into=NativeDict)
```

Note: Needed until [#21256](#) is resolved.

```
static convert_if_needed (value)
```

Converts *value* to native python type.

Warning: Only `Timestamp` and `numpy dtypes` are converted.

```
pandas_extras.conversions.clear_nan (dataframe)
```

Change the pandas.NaT and the pandas.nan elements to None.

Parameters dataframe – The pandas.DataFrame object which should be transformed

Returns The modified *dataframe*

```
pandas_extras.conversions.convert_to_type (dataframe, mapper, *types,  
                                           kwargs_map=None)
```

Converts columns to types specified by the `mapper`. In case of integer, float, signed and unsigned typecasting, the smallest possible type will be chosen. See more details at `to_numeric()`.

```

>>> df = pd.DataFrame({
...     'date': ['05/06/2018', '05/04/2018'],
...     'datetime': [156879000, 156879650],
...     'number': ['1', '2.34'],
...     'int': [4, 8103],
...     'float': [4.0, 8103.0],
...     'object': ['just some', 'strings']
... })
>>> mapper = {
...     'number': 'number', 'integer': 'int', 'float': 'float',
...     'date': ['date', 'datetime']
... }
>>> kwargs_map = {'datetime': {'unit': 'ms'}}
>>> df.pipe(
...     convert_to_type, mapper, 'integer', 'date',
...     'number', 'float', kwargs_map=kwargs_map
... ).dtypes
date           datetime64[ns]
datetime       datetime64[ns]
number          float64
int             int64
float           float32
object          object
dtype: object

```

Parameters

- **dataframe** (`DataFrame`) – The `DataFrame` object to work on.
- **mapper** (`dict`) – Dict with column names as values and any of the following keys: `number`, `integer`, `float`, `signed`, `unsigned`, `date` and `datetime`.
- ***types** (`str`) – any number of keys from the mapper. If omitted, all keys from `mapper` will be used.
- **kwargs_map** (`dict`) – Dict of keyword arguments to apply to `to_datetime()` or `to_numeric()`. Keys must be the column names, values are the kwargs dict.

Returns The converted dataframe

Return type `DataFrame`

`pandas_extras.conversions.truncate_strings` (`dataframe`, `length_mapping`)

Truncates strings in columns to defined length.

```

>>> df = pd.DataFrame({
...     'strings': [
...         'foo',
...         'baz',
...     ],
...     'long_strings': [
...         'foofoofoofoofoo',
...         'bazbazbazbazbaz',
...     ],
...     'even_longer_strings': [
...         'foofoofoofoofoofoofoofoofoo',
...         'bazbazbazbazbazbazbazbazbaz',
...     ]
... })

```

(continues on next page)

(continued from previous page)

```

...})
>>> df.pipe(truncate_strings, {'long_strings': 6, 'even_longer_strings': 9})
  strings long_strings even_longer_strings
0      foo      foofoo      foofoofoo
1      baz      bazbaz      bazbazbaz

```

Parameters

- **dataframe** (`DataFrame`) – The DataFrame object to work on.
- **length_mapping** (`dict`) – Dict of column names and desired length

Returns The converted dataframe**Return type** `DataFrame`

3.2 Hierarchy module

Contains functions to help manage hierarchical data in pandas.

`pandas_extras.hierarchy.flatten_adjacency_list` (`dataframe`, `parent`, `right_on=None`)

Creates the flattened hierarchy out of an adjacency list.

```

>>> df = pd.DataFrame([
...     {'employee': 0, 'manager': None},
...     {'employee': 1, 'manager': 0},
...     {'employee': 2, 'manager': 0},
...     {'employee': 3, 'manager': 0},
...     {'employee': 4, 'manager': 1},
...     {'employee': 5, 'manager': 1},
...     {'employee': 6, 'manager': 2},
...     {'employee': 7, 'manager': 6},
... ])
>>> df.pipe(flatten_adjacency_list, 'manager', right_on='employee')
  employee  manager  manager_1  manager_2
0         0      NaN         NaN         NaN
1         1         0         NaN         NaN
2         2         0         NaN         NaN
3         3         0         NaN         NaN
4         4         1           0         NaN
5         5         1           0         NaN
6         6         2           0         NaN
7         7         6           2           0

>>> df.set_index('employee').pipe(flatten_adjacency_list, 'manager')
  manager  manager_1  manager_2
employee
0         NaN         NaN         NaN
1         0         NaN         NaN
2         0         NaN         NaN
3         0         NaN         NaN
4         1           0         NaN
5         1           0         NaN
6         2           0         NaN
7         6           2           0

```

Parameters

- **dataframe** (`DataFrame`) – The DataFrame object to work on.
- **parent** (`str`) – The name of the column that contains the parent id.
- **right_on** (`str`) – Name of the primary key column. If not given, the indices will be used.

Returns The flattened DataFrame

Return type `DataFrame`

`pandas_extras.hierarchy.get_adjacency_list_depth` (`dataframe`, `parent`, `right_on=None`, `new_column='depth'`)

Calculates node depth in the adjacency list hierarchy.

```
>>> df = pd.DataFrame([
...     {'employee': 0, 'manager': None},
...     {'employee': 1, 'manager': 0},
...     {'employee': 2, 'manager': 0},
...     {'employee': 3, 'manager': 0},
...     {'employee': 4, 'manager': 1},
...     {'employee': 5, 'manager': 1},
...     {'employee': 6, 'manager': 2},
...     {'employee': 7, 'manager': 6},
... ])
>>> df.pipe(get_adjacency_list_depth, 'manager', right_on='employee')
   employee  manager  depth
0         0      NaN      0
1         1         0      1
2         2         0      1
3         3         0      1
4         4         1      2
5         5         1      2
6         6         2      2
7         7         6      3

>>> df.set_index('employee').pipe(
...     get_adjacency_list_depth, 'manager', new_column='level'
... )
   employee  manager  level
0         0      NaN      0
1         1         0      1
2         2         0      1
3         3         0      1
4         4         1      2
5         5         1      2
6         6         2      2
7         7         6      3
```

Parameters

- **dataframe** (`DataFrame`) – The DataFrame object to work on.
- **parent** (`str`) – The name of the column that contains the parent id.
- **right_on** (`str`) – Name of the primary key column. If not given, the indices will be used.

- **new_column** (*str*) – Name of the new column to be created. By default *depth* will be used.

Returns The flattened DataFrame

Return type DataFrame

3.3 Transformations module

Contains functions to help transform columns data containing complex types, like lists or dictionaries.

`pandas_extras.transformations.concatenate_columns` (*dataframe*, *columns*, *new_column*, *descriptor=None*, *mapper=None*)

Concatenates *columns* together along the indices and adds a *descriptor* column, if specified, with the column name where the data originates from.

```
>>> df = pd.DataFrame([
...     {'key': 'TICKET-1', 'assignee': 'Bob', 'reporter': 'Alice'},
...     {'key': 'TICKET-2', 'assignee': 'Bob', 'reporter': 'Alice'},
...     {'key': 'TICKET-3', 'assignee': 'Bob', 'reporter': 'Alice'},
... ])
>>> df.pipe(concatenate_columns, ['assignee', 'reporter'], 'user')
   key          user      descriptor
0  'TICKET-1'  'Alice'  'reporter'
0  'TICKET-1'  'Bob'    'assignee'
1  'TICKET-2'  'Alice'  'reporter'
1  'TICKET-2'  'Bob'    'assignee'
2  'TICKET-3'  'Alice'  'reporter'
2  'TICKET-3'  'Bob'    'assignee'
```

Parameters

- **dataframe** (DataFrame) – The DataFrame object to work on.
- **columns** – The name of the columns which should be concatenated.
- **new_column** – Name of the new column.
- **descriptor** – Name of the new descriptor column.
- **mapper** – A map to apply to *descriptor* values

Returns The concatenated DataFrame

Return type DataFrame

`pandas_extras.transformations.expand_list` (*dataframe*, *column*, *new_column=None*)

Expands lists to new rows.

```
>>> df = DataFrame({
...     'trial_num': [1, 2, 3, 1, 2, 3],
...     'subject': [1, 1, 1, 2, 2, 2],
...     'samples': [
...         [1, 2, 3, 4],
...         [1, 2, 3],
...         [1, 2],
...         [1],
...         []
...     ]
... })
```

(continues on next page)

(continued from previous page)

```

...     None,
...     ]
... })
>>> df.pipe(expand_list, 'samples', new_column='sample_id').head(7)
   trial_num  subject  sample_id
0           1         1           1
0           1         1           2
0           1         1           3
0           1         1           4
1           2         1           1
1           2         1           2
1           2         1           3

```

Warning: Between calls of `expand_list` and/or `expand_lists`, the dataframe index duplications must be removed, otherwise plenty of duplications will occur.

Warning: Calling `expand_list` on multiple columns might cause data duplications, that shall be handled.

Parameters

- **dataframe** (`DataFrame`) – The DataFrame object to work on.
- **column** – The name of the column which should be extracted.
- **new_column** – Name of the new columns. If not defined, columns will not be renamed.

Returns The expanded DataFrame

Return type `DataFrame`

`pandas_extras.transformations.expand_lists` (*dataframe, columns, new_columns=None*)

Expands multiple lists to new rows. Pairs elements of lists respective to their index. Pads with `None` to the longest list.

```

>>> df = DataFrame({
...     'trial_num': [1, 2, 3, 1, 2, 3],
...     'subject': [1, 1, 1, 2, 2, 2],
...     'samples': [
...         [1, 2, 3, 4],
...         [1, 2, 3],
...         [1, 2],
...         [1],
...         [],
...         None,
...     ],
...     'samples2': [
...         [1, 2],
...         [1, 2, 3],
...         [1, 2],
...         [1],
...         [],
...         None,
...     ]
... })

```

(continues on next page)

(continued from previous page)

```

...     ]
... })
>>> df.pipe(
...     expand_lists, ['samples', 'samples'], new_column=['sample_id', 'sample_id2
→'])
... ).head(7)
   trial_num  subject  sample_id  sample_id2
0           1         1           1           1
0           1         1           2           2
0           1         1           3           NaN
0           1         1           4           NaN
1           2         1           1           1
1           2         1           2           2
1           2         1           3           3

```

Warning: Between calls of `expand_list` and/or `expand_lists`, the dataframe index duplications must be removed, otherwise plenty of duplications will occur.

Warning: Calling `expand_lists` on multiple columns might cause data duplications, that shall be handled.

Parameters

- **dataframe** (`DataFrame`) – The `DataFrame` object to work on.
- **columns** – The name of the columns which should be extracted.
- **new_columns** – Name of the new columns. If not defined, columns will not be renamed.

Returns The expanded `DataFrame`

Return type `DataFrame`

`pandas_extras.transformations.extract_dict_key` (*dataframe*, *column*, *key*,
new_column=None, *separator='.'*)

Extract values of `key` into `new_column`. If `key` is missing, `None` is added to the column.

```

>>> df = DataFrame({
...     'trial_num': [1, 2, 1, 2],
...     'subject': [1, 1, 2, 2],
...     'samples': [
...         {'A': 1, 'B': 2, 'C': None},
...         {'A': 3, 'B': 4, 'C': 5},
...         {'A': 6, 'B': 7, 'C': None},
...         None,
...     ]
... })
>>> df.pipe(extract_dict_key, 'samples', key='A')
   trial_num  subject  samples.A  samples
0           1         1           1  {'A': 1, 'B': 2, 'C': None}
1           2         1           3  {'A': 3, 'B': 4, 'C': 5}
2           1         2           6  {'A': 6, 'B': 7, 'C': None}
3           2         2           NaN  NaN

```

Parameters

- **dataframe** (`DataFrame`) – The `DataFrame` object to work on.
- **column** (`str`) – The name of the column which should be extracted.
- **key** (`str`) – Key that should be extracted.
- **new_column** (`str`) – Name of the new column. By default, `column` will be applied as prefix to `key`.
- **separator** (`str`) – The separator between `column` and `key` if `new_column` is not specified.

Returns The extracted `DataFrame`

Return type `DataFrame`

`pandas_extras.transformations.extract_dictionary` (`dataframe`, `column`, `key_list=None`, `prefix=None`, `separator='.'`)

Extract values of keys in `key_list` into separate columns.

```
>>> df = DataFrame({
...     'trial_num': [1, 2, 1, 2],
...     'subject': [1, 1, 2, 2],
...     'samples': [
...         {'A': 1, 'B': 2, 'C': None},
...         {'A': 3, 'B': 4, 'C': 5},
...         {'A': 6, 'B': 7, 'C': None},
...         None,
...     ]
... })
>>> df.pipe(extract_dictionary, 'samples', key_list=('A', 'B'))
   trial_num  subject  samples.A  samples.B
0           1         1          1          2
1           2         1          3          4
2           1         2          6          7
3           2         2         NaN         NaN
```

Warning: `column` will be dropped from the `DataFrame`.

Parameters

- **dataframe** (`DataFrame`) – The `DataFrame` object to work on.
- **column** (`str`) – The name of the column which should be extracted.
- **key_list** (`list`) – Collection of keys that should be extracted. The new column names will be created from the key names.
- **prefix** (`str`) – Prefix for new column names. By default, `column` will be applied as prefix.
- **separator** (`str`) – The separator between the prefix and the key name for new column names.

Returns The extracted `DataFrame`

Return type `DataFrame`

`pandas_extras.transformations.merge_columns` (*dataframe*, *col_header_list*,
new_column_name, *keep=None*,
aggr=None)

Add a new column or modify an existing one in *dataframe* called *new_column_name* by iterating over the rows and select the proper notnull element from the values of *col_header_list* columns in the given row if *keep* is filled OR call the *aggr* function with the values of *col_header_list*. Only one of (*keep*, *aggr*) can be filled.

Parameters

- **dataframe** – the pandas.DataFrame object to modify
- **col_header_list** – list of the names of the headers to merge
- **new_column_name** (*str*) – the name of the new column, if it already exists the operation will overwrite it
- **keep** (*str*) – Specify whether the first or the last proper value is needed. values: *first* and *last* as string.
- **aggr** – Callable function which will get the values of *col_header_list* as parameter. The return value of this function will be the value in *new_column_name*

Returns The merged DataFrame

Return type DataFrame

3.4 Utility module

Contains utility functions.

`pandas_extras.util.check_duplicated_labels` (*dataframe*)

Checks if there are duplications on column labels. Raises *ValueError* if there is any duplicated label.

Parameters **dataframe** (DataFrame) – The DataFrame object to work on.

Returns The original DataFrame

Return type DataFrame

Raises ValueError

modindex search

p

`pandas_extras.conversions`, 5
`pandas_extras.hierarchy`, 7
`pandas_extras.transformations`, 9
`pandas_extras.util`, 13

C

`check_duplicated_labels()` (in module `pandas_extras.util`), 13
`clear_nan()` (in module `pandas_extras.conversions`), 5
`concatenate_columns()` (in module `pandas_extras.transformations`), 9
`convert_if_needed()` (`pandas_extras.conversions.NativeDict` method), 5
`convert_to_type()` (in module `pandas_extras.conversions`), 5

E

`expand_list()` (in module `pandas_extras.transformations`), 9
`expand_lists()` (in module `pandas_extras.transformations`), 10
`extract_dict_key()` (in module `pandas_extras.transformations`), 11
`extract_dictionary()` (in module `pandas_extras.transformations`), 12

F

`flatten_adjacency_list()` (in module `pandas_extras.hierarchy`), 7

G

`get_adjacency_list_depth()` (in module `pandas_extras.hierarchy`), 8

M

`merge_columns()` (in module `pandas_extras.transformations`), 12

N

`NativeDict` (class in `pandas_extras.conversions`), 5

P

`pandas_extras.conversions` (module), 5
`pandas_extras.hierarchy` (module), 7
`pandas_extras.transformations` (module), 9
`pandas_extras.util` (module), 13

T

`truncate_strings()` (in module `pandas_extras.conversions`), 6