trashpanda

Release 1.2.dev1

D. Scheliga

Jan 11, 2022

CONTENTS

trashpanda is a collection of helper methods composing tasks around pandas.DataFrames and Series. Either they are to specific (or I didn't found them).

CHAPTER

ONE

INSTALLATION

Install the latest release from pip.

```
$ pip install trashpanda
```

1.1 API reference

1.1.1 DataFrame & Series related functions

trashpanda.add_blank_rows()	Adds blank rows into a Series or DataFrame with *numpy.nan* by default.
trashpanda.get_intersection()	Intersects Series or DataFrame by requested indexes.
trashpanda.get_unique_index_positions()	Determines positions of unique indexes from a Series or DataFrame.
trashpanda.cut_after()	Cuts a dataframe dropping the part after the cutting index.
trashpanda.cut_before(source_to_cut, ...)	Cuts a dataframe dropping the part after the cutting index.
trashpanda.meld_along_columns(left, right[, ...])	Melds two DataFrames of Series into a single DataFrame with a common index.
trashpanda.remove_duplicated_indexes()	Removes rows of duplicated indexes from a DataFrame.

add_blank_rows

trashpanda.**add_blank_rows**(*source: pandas.core.series.Series*, *indexes_to_add: Union[Iterable, pandas.core.indexes.base.Index]*, *fill_value: Optional[Any] = None*, *override: bool = False*) → pandas.core.series.Series

trashpanda.**add_blank_rows**(*source: pandas.core.frame.DataFrame*, *indexes_to_add: Union[Iterable, pandas.core.indexes.base.Index]*, *fill_value: Optional[Any] = None*, *override: bool = False*) → pandas.core.frame.DataFrame

Adds blank rows into a Series or DataFrame with *numpy.nan* by default. Double indexes are either overriden if argumend *override* is True or ignored.

Parameters

- **source** (*Union[Series, DataFrame]*) – Series or DataFrame in which additional 'blank' rows should be filled.
- **indexes_to_add** (*Union[Iterable, pandas.Index]*) – The targeted indexes, which

are going to be added or overriden.

- **fill_value** (*Optional[Any]*) – Default *numpy.nan*; value which is going to be used as the added rows values.
- **override** (*bool*) – States if the *indexes to add* are overriding the source or ignored.

Returns Union[Series, DataFrame]

Examples

Usage with a pandas.Series

```
>>> from pandas import Series, Index
>>> import numpy as np
>>> from doctestprinter import doctest_print
>>> from trashpanda import add_blank_rows
>>> sample_series = Series(
...     np.arange(3.0), name="a", index=Index([0.1, 0.2, 0.3], name="x")
... )
>>> prepared_frame = add_blank_rows(
...     source=sample_series, indexes_to_add=[0.15, 0.3, 0.35]
... )
>>> doctest_print(prepared_frame)
x
0.10    0.0
0.15    NaN
0.20    1.0
0.30    2.0
0.35    NaN
Name: a, dtype: float64
>>> doctest_print(
...     prepared_frame.interpolate(method="index", limit_area="inside")
... )
x
0.10    0.0
0.15    0.5
0.20    1.0
0.30    2.0
0.35    NaN
Name: a, dtype: float64
>>> doctest_print(
...     add_blank_rows(
...         source=sample_series,
...         indexes_to_add=[0.15, 0.3, 0.35],
...         override=True
...     )
... )
x
0.10    0.0
0.15    NaN
0.20    1.0
0.30    NaN
0.35    NaN
```

(continues on next page)

(continued from previous page)

```
Name: a, dtype: float64
>>> doctest_print(
...     add_blank_rows(
...         source=sample_series, indexes_to_add=[],
...     )
... )
x
0.1    0.0
0.2    1.0
0.3    2.0
Name: a, dtype: float64
```

Usage with a pandas.DataFrame

```
>>> from pandas import DataFrame, Index
>>> import numpy as np
>>> from doctestprinter import doctest_print
>>> from trashpanda import add_blank_rows
>>> sample_series = DataFrame(
...     np.arange(6.0).reshape(3, 2),
...     columns=["b", "a"],
...     index=Index([0.1, 0.2, 0.3], name="x")
... )
>>> prepared_frame = add_blank_rows(
...     source=sample_series, indexes_to_add=[0.15, 0.3, 0.35]
... )
>>> doctest_print(prepared_frame)
       b    a
x
0.10  0.0  1.0
0.15  NaN  NaN
0.20  2.0  3.0
0.30  4.0  5.0
0.35  NaN  NaN
>>> doctest_print(
...     prepared_frame.interpolate(method="index", limit_area="inside")
... )
       b    a
x
0.10  0.0  1.0
0.15  1.0  2.0
0.20  2.0  3.0
0.30  4.0  5.0
0.35  NaN  NaN
>>> doctest_print(
...     add_blank_rows(
...         source=sample_series,
...         indexes_to_add=[0.15, 0.3, 0.35],
...         override=True
...     )
... )
       b    a
```

(continues on next page)

(continued from previous page)

```
x
0.10  0.0  1.0
0.15  NaN  NaN
0.20  2.0  3.0
0.30  NaN  NaN
0.35  NaN  NaN
>>> doctest_print(
...     add_blank_rows(
...         source=sample_series, indexes_to_add=[],
...     )
... )
       b    a
x
0.1  0.0  1.0
0.2  2.0  3.0
0.3  4.0  5.0
```

get_intersection

trashpanda.**get_intersection**(*source: pandas.core.frame.DataFrame*, *targeted_indexes: Union[List, pandas.core.indexes.base.Index]*) → pandas.core.frame.DataFrame
trashpanda.**get_intersection**(*source: pandas.core.series.Series*, *targeted_indexes: Union[List, pandas.core.indexes.base.Index]*) → pandas.core.series.Series

Intersects Series or DataFrame by requested indexes. A subsection from the *source* is made for the *targeted_indexes*, which must not necessaraly be whithin the *source*.

Parameters

- **source** (*Union[DataFrame, Series]*) – Values from which an intersection will be retrieved.
- **targeted_indexes** (*Index*) – The indexes which the returned Series should contain.

Returns Union[DataFrame, Series]

Examples

Usage with pandas.Series

```
>>> from pandas import Series
>>> sample_series = Series(list(range(3)), index=list(iter("abc")), name="foo")
>>> get_intersection(sample_series, ["b", "c", "d"])
b    1
c    2
Name: foo, dtype: int64
>>> get_intersection(sample_series, ["x", "y", "z"])
Series([], Name: foo, dtype: int64)
```

Usage with pandas.DataFrame

```
>>> from pandas import DataFrame
>>> sample_series = DataFrame(
```

(continues on next page)

(continued from previous page)

```
...     list(range(3)), index=list(iter("abc")), columns=["foo"]
... )
>>> get_intersection(sample_series, ["b", "c", "d"])
   foo
b    1
c    2
>>> get_intersection(sample_series, ["x", "y", "z"])
Empty DataFrame
Columns: [foo]
Index: []
```

get_unique_index_positions

trashpanda.**get_unique_index_positions**(*source_with_duplicates: pandas.core.series.Series*, *keep: Union[str, bool] = 'first'*) → pandas.core.indexes.base.Index
trashpanda.**get_unique_index_positions**(*source_with_duplicates: pandas.core.frame.DataFrame*, *keep: Union[str, bool] = 'first'*) → pandas.core.indexes.base.Index

Determines positions of unique indexes from a Series or DataFrame.

Notes

This method assumes duplicates are within the *frame with duplicates*. Check with *index.is_unique* beforehand.

Parameters

- **source_with_duplicates** (*Union[Series, DataFrame]*) – Series or DataFrame with duplicated indexes.
- **keep** (*Union[str, bool]*) – Determines which duplicates to keep. - *first*: Default; keeps all first occurrences of duplicated indexes. - *last*: Keeps all last occurrences of duplicated indexes. - False: Drops all duplicated indexes.

Returns pandas.Index

Examples

```
>>> from pandas import DataFrame
>>> import numpy as np
>>> from doctestprinter import doctest_print
>>> sample_frame = pandas.DataFrame(
...     np.arange(5),
...     columns=["iloc"],
...     index=pandas.Index(['a', 'a', 'b', 'b', 'c'], name="index")
... )
>>> doctest_print(sample_frame)
       iloc
index
a         0
a         1
b         2
```

(continues on next page)

(continued from previous page)

```
b         3
c         4
```

Keeping the first occurrence.

```
>>> get_unique_index_positions(sample_frame, "first")
Int64Index([0, 2, 4], dtype='int64')
```

Keeping the last occurrence.

```
>>> get_unique_index_positions(sample_frame, "last")
Int64Index([1, 3, 4], dtype='int64')
```

Dropping all duplicates.

```
>>> get_unique_index_positions(sample_frame, False)
Int64Index([4], dtype='int64')
```

cut_after

trashpanda.**cut_after**(*source_to_cut: pandas.core.series.Series*, *cutting_index: float*) → pandas.core.series.Series
trashpanda.**cut_after**(*source_to_cut: pandas.core.frame.DataFrame*, *cutting_index: float*) → pandas.core.frame.DataFrame

Cuts a dataframe dropping the part after the cutting index. The cutting index will be added to the frame, which values are being interpolated, if inside.

Parameters

- **source_to_cut** (*Union[Series, DataFrame]*) – Source frame to be cut at the cutting index.
- **cutting_index** (*float*) – Cutting index at which the source frame should be cut.

Returns Union[Series, DataFrame]

Examples

Usage with pandas.Series

```
>>> import numpy
>>> from pandas import Series, Index
>>> from doctestprinter import doctest_print
>>> from trashpanda import cut_after
>>> sample_series = Series(
...     numpy.arange(3.0),
...     index=Index([0.1, 0.2, 0.3], name="x"),
...     name="y"
... )
>>> sample_series
x
0.1    0.0
0.2    1.0
```

(continues on next page)

(continued from previous page)

```
0.3    2.0
Name: y, dtype: float64
>>> cut_after(sample_series, 0.1)
x
0.1    0.0
Name: y, dtype: float64
>>> cut_after(sample_series, 0.14)
x
0.10    0.0
0.14    0.4
Name: y, dtype: float64
>>> cut_after(sample_series, 0.2)
x
0.1    0.0
0.2    1.0
Name: y, dtype: float64
>>> cut_after(sample_series, 0.31)
x
0.10    0.0
0.20    1.0
0.30    2.0
0.31    NaN
Name: y, dtype: float64
>>> cut_after(sample_series, 0.0)
Series([], Name: y, dtype: float64)
```

Usage with a pandas.DataFrame

```
>>> import numpy
>>> from pandas import DataFrame, Index
>>> from doctestprinter import doctest_print
>>> from trashpanda import cut_after
>>> sample_frame = DataFrame(
...     numpy.arange(6.0).reshape(3, 2),
...     columns=["b", "a"],
...     index=Index([0.1, 0.2, 0.3], name="x")
... )
>>> doctest_print(sample_frame)
       b    a
x
0.1  0.0  1.0
0.2  2.0  3.0
0.3  4.0  5.0
>>> doctest_print(cut_after(sample_frame, 0.1))
       b    a
x
0.1  0.0  1.0
>>> doctest_print(cut_after(sample_frame, 0.14))
        b    a
x
0.10  0.0  1.0
0.14  0.8  1.8
```

(continues on next page)

(continued from previous page)

```
>>> doctest_print(cut_after(sample_frame, 0.2))
       b    a
x
0.1  0.0  1.0
0.2  2.0  3.0
>>> doctest_print(cut_after(sample_frame, 0.31))
        b    a
x
0.10  0.0  1.0
0.20  2.0  3.0
0.30  4.0  5.0
0.31  NaN  NaN
>>> cut_after(sample_frame, 0.0)
Empty DataFrame
Columns: [b, a]
Index: []
```

cut_before

trashpanda.**cut_before**(*source_to_cut: Union[pandas.core.series.Series, pandas.core.frame.DataFrame]*, *cutting_index: float*) → Union[pandas.core.series.Series, pandas.core.frame.DataFrame]

Cuts a dataframe dropping the part after the cutting index. The cutting index will be added to the frame, which values are being interpolated, if inside.

Parameters

- **source_to_cut** (*Union[Series, DataFrame]*) – Source frame to be cut at the cutting index.
- **cutting_index** (*float*) – Cutting index at which the source frame should be cut.

Returns Union[Series, DataFrame]

Examples

Usage with pandas.Series

```
>>> import numpy
>>> from pandas import Series, Index
>>> from doctestprinter import doctest_print
>>> from trashpanda import cut_after
>>> sample_series = Series(
...     numpy.arange(3.0),
...     index=Index([0.1, 0.2, 0.3], name="x"),
...     name="y"
... )
>>> sample_series
x
0.1    0.0
0.2    1.0
0.3    2.0
```

(continues on next page)

(continued from previous page)

```
Name: y, dtype: float64
>>> cut_before(sample_series, 0.3)
x
0.3    2.0
Name: y, dtype: float64
>>> cut_before(sample_series, 0.14)
x
0.14    0.4
0.20    1.0
0.30    2.0
Name: y, dtype: float64
>>> cut_before(sample_series, 0.2)
x
0.2    1.0
0.3    2.0
Name: y, dtype: float64
>>> cut_before(sample_series, 0.09)
x
0.09    NaN
0.10    0.0
0.20    1.0
0.30    2.0
Name: y, dtype: float64
>>> cut_before(sample_series, 0.31)
Series([], Name: y, dtype: float64)
```

Usage with a pandas.DataFrame

```
>>> import numpy
>>> from pandas import DataFrame, Index
>>> from doctestprinter import doctest_print
>>> from trashpanda import cut_before
>>> sample_frame = DataFrame(
...     numpy.arange(6.0).reshape(3, 2),
...     columns=["b", "a"],
...     index=Index([0.1, 0.2, 0.3], name="x")
... )
>>> doctest_print(sample_frame)
       b    a
x
0.1  0.0  1.0
0.2  2.0  3.0
0.3  4.0  5.0
>>> doctest_print(cut_before(sample_frame, 0.3))
       b    a
x
0.3  4.0  5.0
>>> doctest_print(cut_before(sample_frame, 0.14))
        b    a
x
0.14  0.8  1.8
0.20  2.0  3.0
```

(continues on next page)

(continued from previous page)

```
0.30  4.0  5.0
>>> doctest_print(cut_before(sample_frame, 0.2))
       b    a
x
0.2  2.0  3.0
0.3  4.0  5.0
>>> doctest_print(cut_before(sample_frame, 0.09))
        b    a
x
0.09  NaN  NaN
0.10  0.0  1.0
0.20  2.0  3.0
0.30  4.0  5.0
>>> cut_before(sample_frame, 0.31)
Empty DataFrame
Columns: [b, a]
Index: []
```

meld_along_columns

trashpanda.**meld_along_columns**(*left: Union[pandas.core.series.Series, pandas.core.frame.DataFrame]*, *right: Union[pandas.core.series.Series, pandas.core.frame.DataFrame]*, *copy_at_meld: bool = True*, *keep: Union[str, bool] = 'raise'*) → pandas.core.frame.DataFrame

Melds two DataFrames of Series into a single DataFrame with a common index.

Notes

This method is called meld because it doesn't fit into existing categories of pandas join, merge or concat.

> **Warning:** This method will override values of equal named columns in *left* with *right*.

Parameters

- **left** (*DataFrame*) – Left curve to be merged with the right one.
- **right** (*DataFrame*) – Right curve, which will merge to the left one.
- **copy_at_meld** (*bool*) – Makes a copy during the concat process, creating a new DataFrame instead of overriding the source.
- **keep** (*Union[str, bool]*) – Determines which duplicates to keep. - *first*: Default; keeps all first occurrences of duplicated indexes. - *last*: Keeps all last occurrences of duplicated indexes. - False: Drops all duplicated indexes.

Returns DataFrame

Examples

Melding of two DataFrame.

```
>>> from pathlib import Path
>>> from pandas import DataFrame, Series
>>> import numpy as np
>>> import examplecurves
>>> left_curve, right_curve = examplecurves.Static.create(
...     family_name="nonlinear0", cut_curves_at=3, curve_selection=[1, 2]
... )
>>> left_curve.columns = ["left"]
>>> right_curve.columns = ["right"]
>>> from doctestprinter import doctest_print
>>> doctest_print(meld_along_columns(left_curve, right_curve))
         left     right
x
0.000  0.0000  0.000000
0.100  1.5625       NaN
0.111     NaN  1.607654
0.200  3.0000       NaN
0.222     NaN  3.085479
```

Melding of 2 Series.

```
>>> left_series = left_curve["left"]
>>> right_series = right_curve["right"]
>>> doctest_print(meld_along_columns(left_curve, right_curve))
         left     right
x
0.000  0.0000  0.000000
0.100  1.5625       NaN
0.111     NaN  1.607654
0.200  3.0000       NaN
0.222     NaN  3.085479
```

Example of melding two DataFrames with duplicated indexes and one duplicated column. Left values of the intersecting columns are being overriden with the right values.

```
>>> left_sample = pandas.DataFrame(
...     np.arange(10).reshape(5, 2),
...     columns=["b", "a"],
...     index=pandas.Index(np.linspace(0.1, 0.5, num=5), name="x")
... )
>>> doctest_print(left_sample)
     b  a
x
0.1  0  1
0.2  2  3
0.3  4  5
0.4  6  7
0.5  8  9
>>> right_sample = pandas.DataFrame(
...     np.linspace(0.5, 9.5, num=10).reshape(5, 2),
```

(continues on next page)

(continued from previous page)

```
...     columns=["a", "c"],
...     index=pandas.Index([0.1, 0.1, 0.15, 0.15, 0.2], name="x")
... )
>>> doctest_print(right_sample)
        a    c
x
0.10  0.5  1.5
0.10  2.5  3.5
0.15  4.5  5.5
0.15  6.5  7.5
0.20  8.5  9.5
>>> merged_frame = meld_along_columns(left_sample, right_sample, keep='first')
>>> doctest_print(merged_frame)
        b    a    c
x
0.10  0.0  0.5  1.5
0.15  NaN  4.5  5.5
0.20  2.0  8.5  9.5
0.30  4.0  5.0  NaN
0.40  6.0  7.0  NaN
0.50  8.0  9.0  NaN
```

remove_duplicated_indexes

trashpanda.**remove_duplicated_indexes**(*source_with_duplicates: pandas.core.series.Series*, *keep: Union[str, bool] = 'first'*) → pandas.core.series.Series

trashpanda.**remove_duplicated_indexes**(*source_with_duplicates: pandas.core.frame.DataFrame*, *keep: Union[str, bool] = 'first'*) → pandas.core.frame.DataFrame

Removes rows of duplicated indexes from a DataFrame. Keeps by default all first occurrences.

Notes

This method assumes duplicates are within the *frame with duplicates*. Check with *index.is_unique* beforehand.

Parameters

- **source_with_duplicates** (*Union[DataFrame, Series]*) – Removes existing duplicates
- **keep** (*Union[str, bool]*) – Determines which duplicates to keep. - *first*: Default; keeps all first occurrences of duplicated indexes. - *last*: Keeps all last occurrences of duplicated indexes. - False: Drops all duplicated indexes.

Returns Union[DataFrame, Series]

Examples

```
>>> from pandas import DataFrame
>>> import numpy as np
>>> from doctestprinter import doctest_print
>>> sample_frame = pandas.DataFrame(
...     np.arange(5),
...     columns=["location"],
...     index=pandas.Index(['a', 'a', 'b', 'b', 'c'], name="index")
... )
>>> doctest_print(sample_frame)
       location
index
a             0
a             1
b             2
b             3
c             4
```

Keeping the first occurrence.

```
>>> first_kept = remove_duplicated_indexes(
...     source_with_duplicates=sample_frame, keep="first"
... )
>>> doctest_print(first_kept)
       location
index
a             0
b             2
c             4
```

Keeping the last occurrence.

```
>>> last_kept = remove_duplicated_indexes(
...     source_with_duplicates=sample_frame, keep="last"
... )
>>> doctest_print(last_kept)
       location
index
a             1
b             3
c             4
```

Dropping all duplicates.

```
>>> dropped_duplicates = remove_duplicated_indexes(
...     source_with_duplicates=sample_frame, keep=False
... )
>>> doctest_print(dropped_duplicates)
       location
index
c             4
```

1.1.2 Series related functions

trashpanda.add_missing_indexes_to_series(...)	Adds different (missing) indexes to series.
trashpanda.cut_series_after_max(series_to_cut)	Cuts a Series after its maximum value.
trashpanda.find_consecutive_value_blocks_in_series(...)	Creates a MultiIndex for consecutive values within a *Series*.
trashpanda.find_index_of_value_in_series(...)	Finds the index of an value within a Series.
trashpanda.keep_consecutive_values_of_series(...)	Keeps consecutive values of a Series.
trashpanda.override_left_with_right_series(...)	Overrides overlapping items of left with right.

add_missing_indexes_to_series

trashpanda.**add_missing_indexes_to_series**(*target_series: pandas.core.series.Series*, *new_indexes: pandas.core.indexes.base.Index*, *fill_value: Optional[Any] = None*) → pandas.core.series.Series

Adds different (missing) indexes to series.

Notes

If no explicit fill value is defined, trashpanda.DEFAULT_NA will be used, which is currently numpy.NaN. Be aware that integer arrays cannot contain NaN values as these are special values for float arrays only. To preserve integer arrays the fill value must be an integer.

Parameters

- **target_series** (*Series*) – Series in which missing indexes should be added.
- **new_indexes** (*pandas.Index*) – Indexes, which should be in the *target series*.
- **fill_value** (*Optional[Any]*) – An optional fill value for the freshly added items.

Returns Series

Examples

```
>>> from pandas import Series, Index, Int16Dtype
>>> from doctestprinter import print_pandas
>>> import numpy as np
>>> target = Series(
...     np.full(3, 1), index=list(iter("abc")), name="foo", dtype=Int16Dtype()
... )
>>> target
a    1
b    1
c    1
Name: foo, dtype: Int16
```

Because new indexes are represented as numpy.NaN the resulting datatype cannot be integer. In dependendy of the current python version either a object or float type is returned.

```
>>> new_indexes_to_add = Index(list(iter("ad")))
>>> float_sample = add_missing_indexes_to_series(target, new_indexes_to_add)
```

(continues on next page)

(continued from previous page)

```
>>> str(float_sample.dtype) == str(target.dtype)
False
>>> print_pandas(float_sample)
   foo
a    1
b    1
c    1
d  nan
```

```
>>> obj_sample = add_missing_indexes_to_series(target, new_indexes_to_add, "X")
>>> obj_sample
a    1
b    1
c    1
d    X
Name: foo, dtype: object
```

cut_series_after_max

trashpanda.**cut_series_after_max**(*series_to_cut: pandas.core.series.Series*) → pandas.core.series.Series

Cuts a Series after its maximum value.

> **Warning:** This method is being replaced by *:func:trashpanda.cut_after_max* in the next release.

Parameters **series_to_cut** (*Series*) – Source frame to be cut at the cutting index.

Returns Series

Examples

```
>>> from pandas import Series, Index
>>> import numpy as np
>>> from doctestprinter import doctest_print
>>> from trashpanda import cut_series_after_max
>>> sample_data = np.sin(np.arange(0.0, np.pi, np.pi/4.0))
>>> sample_series = Series(
...     data=sample_data,
...     name="a",
...     index=Index(numpy.arange(0.0, 0.4, 0.1), name="x")
... )
>>> doctest_print(sample_series)
x
0.0    0.000000
0.1    0.707107
0.2    1.000000
0.3    0.707107
Name: a, dtype: float64
>>> cut_sample = cut_series_after_max(sample_series)
```

(continues on next page)

(continued from previous page)

```
>>> doctest_print(cut_sample)
x
0.0    0.000000
0.1    0.707107
0.2    1.000000
Name: a, dtype: float64
>>> doctest_print(cut_series_after_max(cut_sample))
x
0.0    0.000000
0.1    0.707107
0.2    1.000000
Name: a, dtype: float64
```

find_consecutive_value_blocks_in_series

trashpanda.**find_consecutive_value_blocks_in_series**(*source: pandas.core.series.Series*) → pandas.core.series.Series

Creates a MultiIndex for consecutive values within a *Series*.

> **Warning:** In development and not fully tested.

Parameters **source** – Series with potential consecutive values.

Returns Series with MultiIndex stating the block and origin index.

Return type Series

Notes

Added in 1.2

Examples

Adding the MultiIndex with a leading *block integer* enables the easy selection/iteration of the single blocks.

```
>>> from pandas import Series, Index
>>> from doctestprinter import print_pandas
>>> sample_series = Series(
...     list(iter("A--AA-AAA")),
...     index=Index(list(iter("abcdefghi")), name="old_index"),
...     name="sample"
... )
>>> block_sample = find_consecutive_value_blocks_in_series(source=sample_series)
>>> print_pandas(block_sample)
block  old_index  sample
    0          a       A
    1          b       -
               c       -
    2          d       A
```

(continues on next page)

(continued from previous page)

```
              e       A
    3         f       -
    4         g       A
              h       A
              i       A
```

```
>>> block_sample.loc[1]
old_index
b    -
c    -
Name: sample, dtype: object
```

This function also works with MultiIndexes.

```
>>> from pandas import MultiIndex
>>> from doctestprinter import print_pandas
>>> sample_multi_index = MultiIndex.from_arrays(
...     [[0, 0, 0, 1, 1, 1, 2, 2, 2], list(iter("abcdefghi"))],
...     names=["old-index-1", "old-index-2"]
... )
>>> sample_series = Series(
...     list(iter("A--AA-AAA")), index=sample_multi_index, name="sample"
... )
>>> block_sample = find_consecutive_value_blocks_in_series(source=sample_series)
>>> print_pandas(block_sample)
block  old-index-1  old-index-2  sample
    0            0            a       A
    1                         b       -
                              c       -
    2            1            d       A
                              e       A
    3                         f       -
    4            2            g       A
                              h       A
                              i       A
```

find_index_of_value_in_series

trashpanda.**find_index_of_value_in_series**(*source_series: pandas.core.series.Series*, *search_value: float*) → float

Finds the index of an value within a Series. Only float values as *search value* are supported.

Notes

This method returns the first nearest hit towards the *search value* within the *source series*. This method doesn't detext multiple entries within the *source series*.

Parameters

- **source_series** (*Series*) – Source series in which the nearest index for the *search value* should be found.
- **search_value** (*float*) – The *search value* for which the nearest value's index should be returned.

Returns float

Examples

```
>>> from pandas import Series, Index
>>> import numpy as np
>>> sample_series = Series(
...     data=[1.0, 2.0, 2.0, 3.0, 4.0],
...     index=Index([0.1, 0.2, 0.3, 0.4, 0.5])
... )
>>> sample_series
0.1    1.0
0.2    2.0
0.3    2.0
0.4    3.0
0.5    4.0
dtype: float64
```

In general the first nearest hit toward the search value ist returned.

```
>>> find_index_of_value_in_series(sample_series, -1.0)
0.1
>>> find_index_of_value_in_series(sample_series, 5.0)
0.5
```

Search values within the range returns the first occurance. In this case 0.3 will never be returned.

```
>>> find_index_of_value_in_series(sample_series, 2.49)
0.2
>>> find_index_of_value_in_series(sample_series, 2.5)
0.2
>>> find_index_of_value_in_series(sample_series, 2.51)
0.4
>>> find_index_of_value_in_series(sample_series, 3.0)
0.4
```

keep_consecutive_values_of_series

trashpanda.**keep_consecutive_values_of_series**(*source: pandas.core.series.Series*, *keep: str = 'first'*) → pandas.core.series.Series

Keeps consecutive values of a Series.

> **Warning:** In development and not fully tested.

Parameters

- **source** – Series from which consecutive values should be kept.
- **keep** – Keep the 'first' or 'last' occurrence.

Returns Series

Notes

Added in 1.2

Examples

Keeping all first occurances compared to *pandas.Series.drop_duplicates*.

```
>>> from pandas import Series
>>> sample_series = Series(list(iter("A--AA-AAA")))
>>> consecutive_first = keep_consecutive_values_of_series(
...     source=sample_series, keep="first"
... )
>>> drop_dup_first = sample_series.drop_duplicates(keep="first")
>>> print_compare(
...     sample_series, consecutive_first, drop_dup_first,
...     labels=["sample", "consec-first", "consec-first"]
... )
   sample  consec-first  consec-first
0       A             x             x
1       -             x             x
2       -
3       A             x
4       A
5       -             x
6       A             x
7       A
8       A
```

Keeping all last occurances compared to *pandas.Series.drop_duplicates*.

```
>>> consecutive_last = keep_consecutive_values_of_series(
...     source=sample_series, keep="last"
... )
>>> drop_dup_last = sample_series.drop_duplicates(keep="last")
>>> print_compare(
```

(continues on next page)

(continued from previous page)

```
...     sample_series, consecutive_last, drop_dup_last,
...     labels=["sample", "consec-last", "consec-last"]
... )
   sample  consec-last  consec-last
0       A            x
1       -
2       -            x
3       A
4       A            x
5       -            x            x
6       A
7       A
8       A            x            x
```

override_left_with_right_series

trashpanda.**override_left_with_right_series**(*left_target: pandas.core.series.Series*, *overriding_right: pandas.core.series.Series*) → pandas.core.series.Series

Overrides overlapping items of left with right.

Parameters

- **left_target** (*DataFrame*) – Series which should be overridden.
- **overriding_right** (*DataFrame*) – The new values as Series, which overrides the *left target*.

Returns Series

Examples

```
>>> from pandas import Series, Int16Dtype
>>> import numpy as np
>>> left = Series(np.full(3, 1), index=list(iter("abc")), dtype=Int16Dtype())
>>> left
a    1
b    1
c    1
dtype: Int16
>>> right = Series(np.full(2, 2), index=list(iter("ad")), dtype=Int16Dtype())
>>> right
a    2
d    2
dtype: Int16
>>> override_left_with_right_series(left, right)
a    2
b    1
c    1
d    2
dtype: Int16
```

1.1.3 DataFrame related functions

trashpanda.add_columns_to_dataframe(...[, ...])	Adds columns to a dataframe.
trashpanda.cut_dataframe_after_max(frame_to_cut, ...)	Cuts a DataFrame after the maximum value of its *condition column*.
trashpanda.override_left_with_right_dataframe(...)	Overrides overlapping items of left with right.

add_columns_to_dataframe

trashpanda.**add_columns_to_dataframe**(*frame_to_enlarge: pandas.core.frame.DataFrame*, *column_names: List[str]*, *fill_value: Optional[Any] = None*) → pandas.core.frame.DataFrame

Adds columns to a dataframe. By default the columns are filled with pandas.NA values if no *fill_value* is explizitly given.

Parameters

- **frame_to_enlarge** (*DataFrame*) – pandas.DataFrame which gets additional columns.
- **column_names** (*List[str]*) – Names of additional columns to create.
- **fill_value** (*Optional[Any]*) – Value which will fill the newly created columns. Default pandas.NA

Returns DataFrame

Examples

```
>>> from pandas import DataFrame
>>> import numpy
>>> sample_series = DataFrame(numpy.arange(4).reshape((2,2)), columns=["b", "a"])
>>> sample_series
   b  a
0  0  1
1  2  3
>>> add_columns_to_dataframe(sample_series, ["d", "c"], "+")
   b  a  d  c
0  0  1  +  +
1  2  3  +  +
>>> add_columns_to_dataframe(sample_series, ["d", "c"])
   b  a   d   c
0  0  1 NaN NaN
1  2  3 NaN NaN
```

cut_dataframe_after_max

trashpanda.**cut_dataframe_after_max**(*frame_to_cut: pandas.core.frame.DataFrame*, *condition_column: Optional[Union[str, int]] = None*) → pandas.core.frame.DataFrame

Cuts a DataFrame after the maximum value of its *condition column*. If the DataFrame contains only 1 column, then this column is the *condition column*.

Warning: This method is being replaced by *:func:trashpanda.cut_after_max* in the next release.

Parameters

- **frame_to_cut** (*DataFrame*) – Source frame to be cut at the cutting index.
- **condition_column** (*Union[str, int]*) – Column or its integer position, which contains the conditional maximum value.

Returns DataFrame

Examples

```
>>> from pandas import DataFrame, Index
>>> import numpy as np
>>> from doctestprinter import doctest_print
>>> v1, v2 = np.arange(0.0, np.pi, np.pi/4.0), np.arange(4)
>>> test_data = np.stack((np.sin(v1), v2), axis=1)
>>> test_frame = DataFrame(
...     data=test_data,
...     columns=["b", "a"],
...     index=Index(numpy.arange(0.0, 0.4, 0.1), name="x")
... )
>>> doctest_print(test_frame)
          b    a
x
0.0  0.000000  0.0
0.1  0.707107  1.0
0.2  1.000000  2.0
0.3  0.707107  3.0
>>> doctest_print(cut_dataframe_after_max(test_frame, "b"))
          b    a
x
0.0  0.000000  0.0
0.1  0.707107  1.0
0.2  1.000000  2.0
>>> doctest_print(cut_dataframe_after_max(test_frame, 0))
          b    a
x
0.0  0.000000  0.0
0.1  0.707107  1.0
0.2  1.000000  2.0
```

override_left_with_right_dataframe

trashpanda.**override_left_with_right_dataframe**(*left_target: pandas.core.frame.DataFrame*, *overriding_right: pandas.core.frame.DataFrame*) → pandas.core.frame.DataFrame

Overrides overlapping items of left with right.

Parameters

- **left_target** (*DataFrame*) – Dataframe which should be overridden.
- **overriding_right** (*DataFrame*) – The new values as frame, which overrides the *left target*.

Returns DataFrame

Examples

```
>>> from pandas import DataFrame
>>> import numpy as np
>>> left = DataFrame(np.full(3, 1), index=list(iter("abc")), columns=["v"])
>>> left
   v
a  1
b  1
c  1
>>> right = DataFrame(np.full(2, 2), index=list(iter("ad")), columns=["v"])
>>> right
   v
a  2
d  2
>>> override_left_with_right_dataframe(left, right)
   v
a  2
b  1
c  1
d  2
>>> double_data = [list(range(1, 3)) for i in range(3)]
>>> left = DataFrame(double_data, index=list(iter("abc")), columns=["m", "x"])
>>> left
   m  x
a  1  2
b  1  2
c  1  2
>>> double_data = [list(range(3, 5)) for i in range(2)]
>>> right = DataFrame(double_data, index=list(iter("ad")), columns=["x", "m"])
>>> right
   x  m
a  3  4
d  3  4
>>> override_left_with_right_dataframe(left, right)
   m  x
a  4  3
b  1  2
```

(continues on next page)

(continued from previous page)

```
c  1  2
d  4  3
>>> right = DataFrame(double_data, index=list(iter("ad")), columns=["z", "m"])
>>> right
   z  m
a  3  4
d  3  4
>>> override_left_with_right_dataframe(left, right)
   m    x    z
a  4  2.0  3.0
b  1  2.0  NaN
c  1  2.0  NaN
d  4  NaN  3.0
```

CHAPTER

TWO

INDICES AND TABLES

- genindex

INDEX