dios

Bert Palm

Apr 21, 2020

CONTENTS

1 DictOfSeries **3**

2 example_DictOfSeries **19**

3 Pandas-like indexing **21**

4 Special indexer `.aloc` **23**
- 4.1 Aloc usage . . . 23
- 4.2 The return type . . . 24
- 4.3 Indexer types . . . 24
- 4.4 Aloc overiew table . . . 25
- 4.5 Example dios . . . 25
- 4.6 Select columns, gracefully . . . 26
- 4.7 Selecting Rows a smart way . . . 27
- 4.8 Boolean array-likes as row indexer . . . 27
- 4.9 pandas.Series and boolean pandas.Series as row indexer . . . 28
- 4.10 Nested-lists as row indexer . . . 29
- 4.11 The power of 2D-indexer . . . 30

5 Cookbook **31**
- 5.1 Recipes . . . 31
- 5.2 Broadcast array-likes to multiple columns . . . 31

6 Itype **33**

7 API **35**
- 7.1 Functions . . . 35
- 7.2 Classes . . . 37
- 7.3 Variables . . . 42

8 Index **45**

Python Module Index **47**

Index **49**

The whole package `dios` is mainly a container for the class `dios.DictOfSeries`. See

CHAPTER

ONE

DICTOFSERIES

class dios.**DictOfSeries**(*data=None*, *columns=None*, *index=None*, *itype=None*, *cast_policy='save'*, *fastpath=False*)

Bases: `dios.base._DiosBase`

A data frame where every column has its own index.

DictOfSeries is a collection of pd.Series's which aim to be as close as possible similar to pd.DataFrame. The advantage over pd.DataFrame is, that every *column* has its own row-index, unlike the former, which provide a single row-index for all columns. This solves problems with unaligned data and data which varies widely in length.

Indexing with `di[]`, `di.loc[]` and `di.iloc[]` should work analogous to these methods from pd.DataFrame. The indexer can be a single label, a slice, a list-like, a boolean list-like, or a boolean DictOfSeries/pd.DataFrame and can be used to selectively get or set data.

Parameters

- **data** (*array-like, Iterable, dict, or scalar value*) – Contains data stored in Series.
- **columns** (*array-like*) – Column labels to use for resulting frame. Will default to RangeIndex(0, 1, 2, . . . , n) if no column labels are provided.
- **index** (*Index or array-like*) – Index to use to reindex every given series during init. Ignored if omitted.
- **itype** (*Itype, pd.Index, Itype-string-repr or type*) – Every series that is inserted, must have an index of this type or any of this types subtypes. If None, the itype is inferred as soon as the first non-empty series is inserted.
- **cast_policy** (*{'save', 'force', 'never'}, default 'save'*) – Policy used for (down-)casting the index of a series if its type does not match the `itype`.

Attributes Summary

aloc	Access a group of rows and columns by label(s) or a boolean array with automatic alignment of indexers.
at	Access a single value for a row/column label pair.
cast_policy	The policy to use for casting new columns if its initial itype does not fit.
columns	The column labels of the DictOfSeries
debugDf	Alias for `to_df()` as property, for debugging purpose.

continues on next page

Table 1 – continued from previous page

dtypes	Return pandas.Series with the dtypes of all columns.
empty	Indicator whether DictOfSeries is empty.
iat	Access a single value for a row/column pair by integer position.
iloc	Purely integer-location based indexing for selection by position.
indexes	Return pandas.Series with the indexes of all columns.
itype	The `Itype` of the DictOfSeries.
lengths	Return pandas.Series with the lenght of all columns.
loc	Access a group of rows and columns by label(s) or a boolean array.
size	
values	Return a numpy.array of numpy.arrays with the values of all columns.

Methods Summary

all([axis])	Return whether all elements are True, potentially over an axis.
any([axis])	Return whether any element is True, potentially over an axis.
apply(func[, axis, raw, args])	Apply a function along an axis of the DictOfSeries.
astype(dtype[, copy, errors])	Cast the data to the given data type.
clear()	
copy([deep])	Make a copy of this DictOfSeries' indices and data.
copy_empty([columns])	Return a new DictOfSeries object, with same properties than the original.
dropempty()	Drop empty columns.
dropna([inplace])	Return a bolean array that is *True* if the value is a Nan-value
equals(other)	
for_each(attr_or_callable, **kwds)	Apply a callable or a pandas.Series method or property on each column.
get(key[, default])	
hasnans([axis, drop_empty])	Returns a boolean Series along an axis, which indicates if it contains NA-entries.
index_of([method])	Return an single index with indices from all columns.
isdata()	Alias for `notna(drop_empty=True)`.
isempty()	Returns a boolean Series, which indicates if an column is empty
isin(values)	Return a boolean dios, that indicates if the corresponding value is in the given array-like.
isna([drop_empty])	Return a boolean DictOfSeries which indicates NA positions.
isnull([drop_empty])	Alias for `isna()`
items()	
iteritems()	

continues on next page

Table 2 – continued from previous page

iterrows([fill_value, squeeze])	Iterate over DictOfSeries rows as (index, pandas.Series/DictOfSeries) pairs.
keys()	
max([axis, skipna])	
memory_usage([index, deep])	
min([axis, skipna])	
notempty()	Returns a boolean Series, which indicates if an column is not empty
notna([drop_empty])	Return a boolean DictOfSeries which indicates non-NA positions.
notnull([drop_empty])	Alias, see `notna()`.
pop(*args)	
popitem()	
reduce_columns(func[, initial, skipna])	Reduce all columns to a single pandas.Series by a given function.
setdefault(key[, default])	
squeeze([axis])	Squeeze a 1-dimensional axis objects into scalars.
to_csv(*args, **kwargs)	Write object to a comma-separated values (csv) file.
to_df()	Transform DictOfSeries to a pandas.DataFrame.
to_string([max_rows, min_rows, max_cols, ...])	Pretty print a dios.
update(other)	

Attributes Documentation

aloc

Access a group of rows and columns by label(s) or a boolean array with automatic alignment of indexers.

See *indexing docs*

at

Access a single value for a row/column label pair.

See *indexing docs*

cast_policy

The policy to use for casting new columns if its initial itype does not fit.

See *Itype documentation* for more info.

columns

The column labels of the DictOfSeries

debugDf

Alias for `to_df()` as property, for debugging purpose.

dtypes

Return pandas.Series with the dtypes of all columns.

empty

Indicator whether DictOfSeries is empty.

Returns If DictOfSeries is empty, return True, if not return False.

Return type bool

See also:

DictOfSeries.dropempty drop empty columns

DictOfSeries.dropna drop NAN's from a DictOfSeries

pandas.Series.dropna drop NAN's from a Series

Notes

If DictOfSeries contains only NaNs, it is still not considered empty. See the example below.

Examples

An example of an actual empty DictOfSeries.

```
>>> di_empty = DictOfSeries(columns=['A'])
>>> di_empty
Empty DictOfSeries
Columns: ['A']
>>> di_empty.empty
True
```

If we only have NaNs in our DictOfSeries, it is not considered empty! We will need to drop the NaNs to make the DictOfSeries empty:

```
>>> di = pd.DictOfSeries({'A' : [np.nan]})
>>> di
    A |
===== |
0 NaN |
>>> di.empty
False
>>> di.dropna().empty
True
```

iat

Access a single value for a row/column pair by integer position.

See *indexing docs*

iloc

Purely integer-location based indexing for selection by position.

See *indexing docs*

indexes

Return pandas.Series with the indexes of all columns.

itype

The `Itype` of the DictOfSeries.

See *Itype documentation* for more info.

lengths

Return pandas.Series with the lenght of all columns.

loc

Access a group of rows and columns by label(s) or a boolean array.

See *indexing docs*

size

values

Return a numpy.array of numpy.arrays with the values of all columns.

The outer has the length of columns, the inner holds the values of the column.

Methods Documentation

all(*axis=0*)

Return whether all elements are True, potentially over an axis.

Returns True unless there at least one element within a series or along a DictOfSeries axis that is False or equivalent (e.g. zero or empty).

Parameters **axis**(*{0 or 'index', 1 or 'columns', None}, default 0*) –

Indicate which axis or axes should be reduced.

- 0 / 'index' : reduce the index, return a Series whose index is the original column labels.
- 1 / 'columns' : reduce the columns, return a Series whose index is the union of all columns indexes.
- None : reduce all axes, return a scalar.

Returns

Return type pandas.Series

See also:

pandas.Series.all() Return True if all elements are True.

any() Return True if one (or more) elements are True.

any(*axis=0*)

Return whether any element is True, potentially over an axis.

Returns False unless there at least one element within a series or along a DictOfSeries axis that is True or equivalent (e.g. non-zero or non-empty).

Parameters **axis**(*{0 or 'index', 1 or 'columns', None}, default 0*) –

Indicate which axis or axes should be reduced.

- 0 / 'index' : reduce the index, return a Series whose index is the original column labels.
- 1 / 'columns' : reduce the columns, return a Series whose index is the union of all columns indexes.
- None : reduce all axes, return a scalar.

Returns

Return type pandas.Series

See also:

pandas.Series.any() Return whether any element is True.

all() Return True if all elements are True.

apply (*func*, *axis=0*, *raw=False*, *args=()*, ***kwds*)

Apply a function along an axis of the DictOfSeries.

Parameters

- **func** (*callable*) – Function to apply on each column.
- **axis** (*{0 or 'index', 1 or 'columns'}, default 0*) – Axis along which the function is applied:
  - 0 or 'index': apply function to each column.
  - 1 or 'columns': NOT IMPLEMENTED
- **raw** (*bool, default False*) – Determines if row or column is passed as a Series or ndarray object:
  - `False` : passes each row or column as a Series to the function.
  - `True` : the passed function will receive ndarray objects instead. If you are just applying a NumPy reduction function this will achieve much better performance.
- **args** (*tuple*) – Positional arguments to pass to *func* in addition to the array/series.
- ****kwds** – Additional keyword arguments to pass as keywords arguments to *func*.

Returns Result of applying `func` along the given axis of the DataFrame.

Return type Series or DataFrame

Raises **NotImplementedError** –

- if axis is 'columns' or 1

See also:

DictOfSeries.for_each() apply pd.Series methods or properties to each column

Examples

We use the example DictOfSeries from *indexing*.

```
>>> di = di[:5]
    a |    b |     c |     d |
===== | ==== | ===== | ===== |
0   0 | 2  5 | 4   7 | 6   0 |
1   7 | 3  6 | 5  17 | 7   1 |
2  14 | 4  7 | 6  27 | 8   2 |
3  21 | 5  8 | 7  37 | 9   3 |
4  28 | 6  9 | 8  47 | 10  4 |
```

```
>>> di.apply(max)
columns
a    28
b     9
c    47
d     4
dtype: int64
```

```
>>> di.apply(pd.Series.count)
columns
a    5
b    5
c    5
d    5
dtype: int64
```

One can pass keyword arguments directly..

```
>>> di.apply(pd.Series.value_counts, normalize=True)
      a |       b |        c |       d |
======= | ====== | ======= | ====== |
7   0.2 | 7  0.2 | 7   0.2 | 4  0.2 |
14  0.2 | 6  0.2 | 37  0.2 | 3  0.2 |
21  0.2 | 5  0.2 | 47  0.2 | 2  0.2 |
28  0.2 | 9  0.2 | 27  0.2 | 1  0.2 |
0   0.2 | 8  0.2 | 17  0.2 | 0  0.2 |
```

Or define a own funtion..

```
>>> di.apply(lambda s : 'high' if max(s) > 10 else 'low')
columns
a    high
b     low
c    high
d     low
dtype: object
```

And also more advanced functions that return a list-like can be given. Note that the returned lists not necessarily must have the same length.

```
>>> func = lambda s : ('high', max(s), min(s)) if min(s) > (max(s)//2) else (
↪'low',max(s))
>>> di.apply(func)
     a |       b |       c |       d |
====== | ======= | ====== | ====== |
0  low | 0  high | 0  low | 0  low |
1   28 | 1     9 | 1   47 | 1    4 |
       | 2     5 |        |        |
```

astype(*dtype*, *copy=True*, *errors='raise'*)

Cast the data to the given data type.

clear()

copy(*deep=True*)

Make a copy of this DictOfSeries' indices and data.

Parameters **deep** (*bool, default True*) – Make a deep copy, including a copy of the data and the indices. With deep=False neither the indices nor the data are copied.

Returns **copy**

Return type *DictOfSeries*

See also:

`pandas.DataFrame.copy()`

copy_empty(*columns=True*)

Return a new DictOfSeries object, with same properties than the original. :param columns: If `True`, the copy will have the same, but empty columns like the original. :type columns: bool, default True

Returns **DictOfSeries**

Return type empty copy

Examples

```
>>> di = DictOfSeries({'A': range(2), 'B': range(3)})
>>> di
   A |    B |
==== | ==== |
0  0 | 0  0 |
1  1 | 1  1 |
     | 2  2 |
```

```
>>> empty = di.copy_empty()
>>> empty
Empty DictOfSeries
Columns: ['A', 'B']
```

The properties are the same, eg.

```
>>> empty.itype == di.itype
True
>>> empty.cast_policy == di.cast_policy
True
>>> empty.dtypes == di.dtypes
columns
A    True
B    True
dtype: bool
```

dropempty()

Drop empty columns. Return copy.

dropna(*inplace=False*)

Return a bolean array that is *True* if the value is a Nan-value

equals(*other*)

for_each(*attr_or_callable*, ***kwds*)

Apply a callable or a pandas.Series method or property on each column.

Parameters

- **attr_or_callable** (*Any*) – A pandas.Series attribute or any callable, to apply on each column. A series attribute can be any property, field or method and also could be specified as string. If a callable is given it must take pandas.Series as the only positional argument.
- ****kwds** (*any*) – kwargs to passed to callable

Returns A series with the results, indexed by the column labels.

Return type pandas.Series

See also:

DictOfSeries.apply() Apply functions to columns and convert result to DictOfSeries.

Examples

```
>>> d = DictOfSeries([range(3), range(4)], columns=['a', 'b'])
>>> d
   a |    b |
==== | ==== |
0  0 | 0  0 |
1  1 | 1  1 |
2  2 | 2  2 |
     | 3  3 |
```

Use with a callable..

```
>>> d.for_each(max)
columns
a    2
b    3
dtype: object
```

..or with a string, denoting a pd.Series attribute and therefor is the same as giving the latter.

```
>>> d.for_each('max')
columns
a    2
b    3
dtype: object
```

```
>>> d.for_each(pd.Series.max)
columns
a    2
b    3
dtype: object
```

Both also works with properties:

```
>>> d.for_each('dtype')
columns
a    int64
b    int64
dtype: object
```

get (*key*, *default=None*)

hasnans (*axis=0*, *drop_empty=False*)

Returns a boolean Series along an axis, which indicates if it contains NA-entries.

index_of (*method='all'*)

Return an single index with indices from all columns.

Parameters **method** (*string, default 'all'*) –

- 'all' : get all indices from all columns
- 'union' : alias for 'all'
- 'shared' : get indices that are present in every columns

- 'intersection' : alias for 'shared'
- 'uniques' : get indices that are only present in a single column
- 'non-uniques' : get indices that are present in more than one column

Returns A single duplicate-free index, somehow representing indices of all columns.

Return type pd.Index

Examples

We use the example DictOfSeries from *indexing*.

```
>>> di
    a |       b |       c |      d |
===== | ====== | ====== | ===== |
0   0 | 2    5 | 4    7 | 6   0 |
1   7 | 3    6 | 5   17 | 7   1 |
2  14 | 4    7 | 6   27 | 8   2 |
3  21 | 5    8 | 7   37 | 9   3 |
4  28 | 6    9 | 8   47 | 10  4 |
5  35 | 7   10 | 9   57 | 11  5 |
6  42 | 8   11 | 10  67 | 12  6 |
7  49 | 9   12 | 11  77 | 13  7 |
8  56 | 10  13 | 12  87 | 14  8 |
9  63 | 11  14 | 13  97 | 15  9 |
```

```
>>> di.index_of()
RangeIndex(start=0, stop=16, step=1)
```

```
>>> di.index_of("shared")
Int64Index([6, 7, 8, 9], dtype='int64')
```

```
>>> di.index_of("uniques")
Int64Index([0, 1, 14, 15], dtype='int64')
```

isdata()

Alias for `notna(drop_empty=True)`.

isempty()

Returns a boolean Series, which indicates if an column is empty

isin(*values*)

Return a boolean dios, that indicates if the corresponding value is in the given array-like.

isna(*drop_empty=False*)

Return a boolean DictOfSeries which indicates NA positions.

isnull(*drop_empty=False*)

Alias for `isna()`

items()

iteritems()

iterrows(*fill_value=nan*, *squeeze=True*)

Iterate over DictOfSeries rows as (index, pandas.Series/DictOfSeries) pairs. **MAY BE VERY PERFORMANCE AND/OR MEMORY EXPENSIVE**

Parameters

- **fill_value** (*scalar, default numpy.nan*) – Fill value for row entry, if the column does not have an entry at the current index location. This ensures that the returned Row always contain all columns. If `None` is given no value is filled.

  If `fill_value=None` and `squeeze=True` the resulting Row (a pandas.Series) may differ in length between iterator calls. That's because an entry, that is not present in a column, will also not be present in the resulting Row.

- **squeeze** (*bool, default False*) –
  - `True` : A pandas.Series is returned for each row.
  - `False` : A single-rowed DictOfSeries is returned for each row.

Yields

- **index** (*label*) – The index of the row.
- **data** (*Series or DictOfSeries*) – The data of the row as a Series if squeeze is True, as a DictOfSeries otherwise.

See also:

DictOfSeries.iteritems() Iterate over (column name, Series) pairs.

keys()

max(*axis=None*, *skipna=None*)

memory_usage(*index=True*, *deep=False*)

min(*axis=0*, *skipna=True*)

notempty()
Returns a boolean Series, which indicates if an column is not empty

notna(*drop_empty=False*)
Return a boolean DictOfSeries which indicates non-NA positions.

notnull(*drop_empty=False*)
Alias, see `notna()`.

pop(**args*)

popitem()

reduce_columns(*func*, *initial=None*, *skipna=False*)
Reduce all columns to a single pandas.Series by a given function.

Apply a function of two pandas.Series as arguments, cumulatively to all columns, from left to right, so as to reduce the columns to a single pandas.Series. If initial is present, it is placed before the columns in the calculation, and serves as a default when the columns are empty.

Parameters

- **func** (*function*) – The function must take two identically indexed pandas.Series and should return a single pandas.Series with the same index.
- **initial** (*column-label or pd.Series, default None*) – The series to start with. If None a dummy series is created, with the indices of all columns and the first seen values.
- **skipna** (*bool, default False*) – If True, skip NaN values.

Returns A series with the reducing result and the index of the start series, defined by `initializer`.

Return type pandas.Series

setdefault(*key*, *default=None*)

squeeze(*axis=None*)

Squeeze a 1-dimensional axis objects into scalars.

to_csv(**args*, ***kwargs*)

Write object to a comma-separated values (csv) file.

Changed in version 0.24.0: The order of arguments for Series was changed.

Parameters

- **path_or_buf** (*str or file handle, default None*) – File path or object, if None is provided the result is returned as a string. If a file object is passed it should be opened with *newline=''*, disabling universal newlines.

  Changed in version 0.24.0: Was previously named "path" for Series.

- **sep** (*str, default ','*) – String of length 1. Field delimiter for the output file.
- **na_rep** (*str, default ''*) – Missing data representation.
- **float_format** (*str, default None*) – Format string for floating point numbers.
- **columns** (*sequence, optional*) – Columns to write.
- **header** (*bool or list of str, default True*) – Write out the column names. If a list of strings is given it is assumed to be aliases for the column names.

  Changed in version 0.24.0: Previously defaulted to False for Series.

- **index** (*bool, default True*) – Write row names (index).
- **index_label** (*str or sequence, or False, default None*) – Column label for index column(s) if desired. If None is given, and *header* and *index* are True, then the index names are used. A sequence should be given if the object uses MultiIndex. If False do not print fields for index names. Use index_label=False for easier importing in R.
- **mode** (*str*) – Python write mode, default 'w'.
- **encoding** (*str, optional*) – A string representing the encoding to use in the output file, defaults to 'utf-8'.
- **compression** (*str or dict, default 'infer'*) – If str, represents compression mode. If dict, value at 'method' is the compression mode. Compression mode may be any of the following possible values: {'infer', 'gzip', 'bz2', 'zip', 'xz', None}. If compression mode is 'infer' and *path_or_buf* is path-like, then detect compression mode from the following extensions: '.gz', '.bz2', '.zip' or '.xz'. (otherwise no compression). If dict given and mode is 'zip' or inferred as 'zip', other entries passed as additional compression options.

  Changed in version 1.0.0: May now be a dict with key 'method' as compression mode and other entries as additional compression options if compression mode is 'zip'.

- **quoting** (*optional constant from csv module*) – Defaults to csv.QUOTE_MINIMAL. If you have set a *float_format* then floats are converted to strings and thus csv.QUOTE_NONNUMERIC will treat them as non-numeric.

- **quotechar** (*str, default '"'*) – String of length 1. Character used to quote fields.
- **line_terminator** (*str, optional*) – The newline character or character sequence to use in the output file. Defaults to *os.linesep*, which depends on the OS in which this method is called ('n' for linux, 'rn' for Windows, i.e.).

  Changed in version 0.24.0.
- **chunksize** (*int or None*) – Rows to write at a time.
- **date_format** (*str, default None*) – Format string for datetime objects.
- **doublequote** (*bool, default True*) – Control quoting of *quotechar* inside a field.
- **escapechar** (*str, default None*) – String of length 1. Character used to escape *sep* and *quotechar* when appropriate.
- **decimal** (*str, default '.'*) – Character recognized as decimal separator. E.g. use ',' for European data.

Returns If path_or_buf is None, returns the resulting csv format as a string. Otherwise returns None.

Return type None or str

See also:

read_csv() Load a CSV file into a DataFrame.

to_excel() Write DataFrame to an Excel file.

Examples

```
>>> df = pd.DataFrame({'name': ['Raphael', 'Donatello'],
...                    'mask': ['red', 'purple'],
...                    'weapon': ['sai', 'bo staff']})
>>> df.to_csv(index=False)
'name,mask,weapon\nRaphael,red,sai\nDonatello,purple,bo staff\n'
```

Create 'out.zip' containing 'out.csv'

```
>>> compression_opts = dict(method='zip',
...                         archive_name='out.csv')
>>> df.to_csv('out.zip', index=False,
...           compression=compression_opts)
```

to_df()

Transform DictOfSeries to a pandas.DataFrame.

Because a pandas.DataFrame can not handle Series of different length, but DictOfSeries can, the missing data is filled with NaNs.

Returns **pandas.DataFrame**

Return type transformed data

Examples

Missing data locations are filled with NaN's

```
>>> a = pd.Series(11, index=range(2))
>>> b = pd.Series(22, index=range(3))
>>> c = pd.Series(33, index=range(1,9,3))
>>> di = DictOfSeries(dict(a=a, b=b, c=c))
>>> di
    a |     b |     c |
===== | ===== | ===== |
0  11 | 0  22 | 1  33 |
1  11 | 1  22 | 4  33 |
      | 2  22 | 7  33 |
>>> di.to_df()
columns     a     b     c
0        11.0  22.0   NaN
1        11.0  22.0  33.0
2         NaN  22.0   NaN
4         NaN   NaN  33.0
7         NaN   NaN  33.0
```

to_string(*max_rows=None*, *min_rows=None*, *max_cols=None*, *na_rep='NaN'*, *show_dimensions=False*, *method='indexed'*, *no_value=' '*, *empty_series_rep='no data'*, *col_delim='|'*, *header_delim='='*, *col_space=None*)

Pretty print a dios.

if *method* == *indexed* (default): every column is represented by a own index and corresponding values

if *method* == *aligned* [2]: one(!) global index is generated and values from a column appear at the corresponding index-location.

Parameters

- **max_cols** – not more column than *max_cols* are printed [1]
- **max_rows** – see *min_rows* [1]
- **min_rows** – not more rows than *min_rows* are printed, if rows of any series exceed *max_rows* [1]
- **na_rep** – all NaN-values are replaced by *na_rep*. Default *NaN*
- **empty_series_rep** – Ignored if not *method='indexed'*. Empty series are represented by the string in *empty_series_rep*
- **col_delim** (*str*) – Ignored if not *method='indexed'*. between all columns *col_delim* is inserted.
- **header_delim** – Ignored if not *method='indexed'*. between the column names (header) and the data, *header_delim* is inserted, if not None. The string is repeated, up to the width of the column. (str or None).
- **no_value** – Ignored if not *method='aligned'*. value that indicates, that no entry in the underling series is present. Bear in mind that this should differ from *na_rep*, otherwise you cannot differ missing- from NaN- values.

Notes

[1]: defaults to the corresponding value in *dios_options* [2]: the common-params are directly passed to pd.DataFrame.to_string(..) under the hood, if method is *aligned*

update(*other*)

CHAPTER

TWO

EXAMPLE_DICTOFSERIES

`dios.example_DictOfSeries()`

Return a example dios.

Returns **DictOfSeries**

Return type an example

Examples

```
>>> from dios import example_DictOfSeries
>>> di = example_DictOfSeries()
>>> di
    a |      b |      c |     d |
===== | ====== | ====== | ===== |
0   0 | 2    5 | 4    7 | 6   0 |
1   7 | 3    6 | 5   17 | 7   1 |
2  14 | 4    7 | 6   27 | 8   2 |
3  21 | 5    8 | 7   37 | 9   3 |
4  28 | 6    9 | 8   47 | 10  4 |
5  35 | 7   10 | 9   57 | 11  5 |
6  42 | 8   11 | 10  67 | 12  6 |
7  49 | 9   12 | 11  77 | 13  7 |
8  56 | 10  13 | 12  87 | 14  8 |
9  63 | 11  14 | 13  97 | 15  9 |
```

Most magic happen in getting and setting elements. To select any combination from columns and rows, read the documentation about indexing:

CHAPTER

THREE

PANDAS-LIKE INDEXING

`[]` and `.loc[]`, `.iloc[]` and `.at[]`, `.iat[]` - should behave exactly like their counter-parts from pandas.DataFrame. They can take as indexer

- lists, array-like objects and in general all iterables
- boolean lists and iterables
- slices
- scalars and any hashable object

Most indexers are directly passed to the underling columns-series or row-series depending on the position of the indexer and the complexity of the operation. For `.loc`, `.iloc`, `.at` and `iat` the first position is the *row indexer*, the second the *column indexer*. The second can be omitted and will default to `slice(None)`. Examples:

- `di.loc[[1,2,3], ['a']]` : select labels 1,2,3 from column a
- `di.iloc[[1,2,3], [0,3]]` : select positions 1,2,3 from the columns 0 and 3
- `di.loc[:, 'a':'c']` : select all rows from columns a to d
- `di.at[4,'c']` : select the elements with label 4 in column c
- `di.loc[:]` -> `di.loc[:,:]` : select everything.

Scalar indexing always return a pandas Series if the other indexer is a non-scalar. If both indexer are scalars, the element itself is returned. In all other cases a dios is returned. For more pandas-like indexing magic and the differences between the indexers, see the pandas documentation.

> **Note:**
>
> In contrast to pandas.DataFrame, `.loc[:]` and `.loc[:, :]` always behaves identical. Same apply for `iloc` and *`aloc`*. For example, two pandas.DataFrames `df1` and `df2` with different columns, does align columns with `df1.loc[:, :] = df2` , but does **not** with `df1.loc[:] = df2`.
>
> If this is the desired behavior or a bug, i couldn't verify so far. – Bert Palm

2D-indexer

`dios[boolean dios-like]` (as single key) - dios accept boolean 2D-indexer (boolean pandas.Dataframe or boolean Dios).

Columns and rows from the indexer align with the dios. This means that only matching columns selected and in this columns rows are selected where i) indices are match and ii) the value is True in the indexer-bool-dios. There is no difference between missing indices and present indices, but False values.

Values from unselected rows and columns are dropped, but empty columns are still preserved, with the effect that the resulting Dios always have the same column dimension than the initial dios.

> **Note:** This is the exact same behavior like pandas.DataFrame's handling of 2D-indexer, despite that pandas.DataFrame fill numpy.nan's at missing locations and therefore also fill-up, whole missing columns with numpy.nan's.

setting values

Setting values with `[]` and `.loc[]`, `.iloc[]` and `.at[]`, `.iat[]` works like in pandas. With `.at`/`.iat` only single items can be set, for the other the right hand side values can be:

- *scalars*: these are broadcasted to the selected positions
- *lists*: the length the list must match the number of indexed columns. The items can be everything that can applied to a series, with the respective indexing method (`loc`, `iloc`, `[]`).
- *dios*: the length of the columns must match the number of indexed columns - columns does *not* align, they are just iterated. Rows do align. Rows that are present on the right but not on the left are ignored. Rows that are present on the left (bear in mind: these rows was explicitly chosen for write!), but not present on the right, are filled with `NaN`s, like in pandas.
- *pandas.Series*: column indexer must be a scalar(!), the series is passed down, and set with `loc`, `iloc` or `[]` by pandas Series, where it maybe align, depending on the method.

Examples:

- `dios.loc[2:5, 'a'] = [1,2,3]` is the same as `a=dios['a']; a.loc[2:5]=[1,2,3]; dios['a']=a`
- `dios.loc[2:5, :]  = 99` : set 99 on rows 2 to 5 on all columns

CHAPTER

FOUR

SPECIAL INDEXER .ALOC

Additional to the pandas like indexers we have a `.aloc[..]` (align locator) indexing method. Unlike `.iloc` and `.loc` indexers fully align if possible and 1D-array-likes can be broadcast to multiple columns at once. This method also handle missing indexer-items gracefully. It is used like `.loc`, so a single indexer (`.aloc[indexer]`) or a tuple of row-indexer and column-indexer (`.aloc[row-indexer, column-indexer]`) can be given. Also it can handle boolean and *non-bolean* 2D-Indexer.

The main **purpose** of `.aloc` is:

- to select gracefully, so rows or columns, that was given as indexer, but doesn't exist, not raise an error
- align series/dios-indexer
- vertically broadcasting aka. setting multiple columns at once with a list-like value

4.1 Aloc usage

`aloc` is *called* like `loc`, with a single key, that act as row indexer `aloc[rowkey]` or with a tuple of row indexer and column indexer `aloc[rowkey, columnkey]`. Also 2D-indexer (like dios or df) can be given, but only as a single key, like `.aloc[2D-indexer]` or with the special column key `...`, the ellipsis (`.aloc[2D-indexer, ...]`). The ellipsis may change, how the 2D-indexer is interpreted, but this will explained *later* in detail.

If a normal (non 2D-dimensional) row indexer is given, but no column indexer, the latter defaults to `:` aka. `slice(None)`, so `.aloc[row-indexer]` becomes `.aloc[row-indexer, :]`, which means, that all columns are used. In general, a normal row-indexer is applied to every column, that was chosen by the column indexer, but for each column separately.

So maybe a first example gives an rough idea:

```
>>> s = pd.Series([11] * 4 )
>>> di = DictOfSeries(dict(a=s[:2]*6, b=s[2:4]*7, c=s[:2]*8, d=s[1:3]*9))
>>> di
    a |     b |     c |     d |
===== | ===== | ===== | ===== |
0  66 | 2  77 | 0  88 | 1  99 |
1  66 | 3  77 | 1  88 | 2  99 |

>>> di.aloc[[1,2], ['a', 'b', 'd', 'x']]
    a |     b |     d |
===== | ===== | ===== |
1  66 | 2  77 | 1  99 |
      |       | 2  99 |
```

4.2 The return type

Unlike the other two indexer methods `loc` and `iloc`, it is not possible to get a single item returned; the return type is either a pandas.Series, iff the column-indexer is a single key (eg. `'a'`) or a dios, iff not. The row-indexer does not play any role in the return type choice.

> **Note for the curios:**
>
> This is because a scalar (`.aloc[key]`) is translates to `.loc[key:key]` under the hood.

4.3 Indexer types

Following the `.aloc` specific indexer are listed. Any indexer that is not listed below (slice, boolean lists, ...), but are known to work with `.loc`, are treated as they would passed to `.loc`, as they actually do under the hood.

Some indexer are linked to later sections, where a more detailed explanation and examples are given.

special Column indexer *are :*

- *list / array-like* (or any iterable object): Only labels that are present in the columns are used, others are ignored.
- *pd.Series* : `.values` are taken from series and handled like a *list*.
- *scalar* (or any hashable obj) : Select a single column, if label is present, otherwise nothing.

special Row indexer *are :*

- *list / array-like* (or any iterable object): Only rows, which indices are present in the index of the column are used, others are ignored. A dios is returned.
- *scalar* (or any hashable obj) : Select a single row from a column, if the value is present in the index of the column, otherwise nothing is selected. [1]
- *pd.Series* : align the index from the given Series with the column, what means only common indices are used. The actual values of the series are ignored(!).
- *boolean pd.Series* : like *pd.Series* but only True values are evaluated. False values are equivalent to missing indices. To treat a boolean series as a *normal* indexer series, as decribed above, one can use `.aloc(usebool=False)[boolean pd.Series]`.

special 2D-indexer *are :*

- `.aloc[boolean dios-like]` : work same like `di[boolean dios-like]` (see there). Brief: full align, select items, where the index is present and the value is True.
- `.aloc[dios-like, ...]` (with Ellipsis) : Align in columns and rows, ignore its values. Per common column, the common indices are selected. The ellipsis forces `aloc`, to ignore the values, so a boolean dios could be treated as a non-boolean. Alternatively `.aloc(usebool=False)[boolean dios-like]` could be used.[2]
- `.aloc[nested list-like]` : The inner lists are used as `aloc`-*list*-row-indexer (see there) on all columns. One list for one column, which implies, that the outer list has the same length as the number of columns.

*special handling of 1D-**values***

Values that are list- or array-like, which includes pd.Series, are set on all selected columns. pd.Series align like `s1.loc[:] = s2` do. See also the cookbook.

4.4 Aloc overiew table

4.5 Example dios

The Dios used in the examples, unless stated otherwise, looks like so:

```
>>> dictofser
    a |      b |      c |     d |
===== | ====== | ====== | ===== |
0   0 | 2    5 | 4    7 | 6   0 |
1   7 | 3    6 | 5   17 | 7   1 |
2  14 | 4    7 | 6   27 | 8   2 |
3  21 | 5    8 | 7   37 | 9   3 |
4  28 | 6    9 | 8   47 | 10  4 |
5  35 | 7   10 | 9   57 | 11  5 |
6  42 | 8   11 | 10  67 | 12  6 |
7  49 | 9   12 | 11  77 | 13  7 |
8  56 | 10  13 | 12  87 | 14  8 |
```

or the short version:

```
>>> di
    a |    b |     c |     d |
===== | ==== | ===== | ===== |
0   0 | 2  5 | 4   7 | 6   0 |
1   7 | 3  6 | 5  17 | 7   1 |
2  14 | 4  7 | 6  27 | 8   2 |
3  21 | 5  8 | 7  37 | 9   3 |
4  28 | 6  9 | 8  47 | 10  4 |
```

The example Dios can get via a function:

```
from dios import example_DictOfSeries()
mydios = example_DictOfSeries()
```

or generated manually like so:

```
>>> a = pd.Series(range(0, 70, 7))
>>> b = pd.Series(range(5, 15, 1))
>>> c = pd.Series(range(7, 107, 10))
>>> d = pd.Series(range(0, 10, 1))
>>> for i, s in enumerate([a,b,c,d]): s.index += i*2
>>> dictofser = DictOfSeries(dict(a=a, b=b, c=c, d=d))
>>> di = dictofser[:5]
```

4.6 Select columns, gracefully

One can use `.aloc[:, key]` to select **single columns** gracefully. The underling pandas.Series is returned, if the key exist. Otherwise a empty pandas.Series with `dtype=object` is returned.

```
>>> di.aloc[:, 'a']
0     0
1     7
2    14
3    21
4    28
Name: a, dtype: int64

>>> di.aloc[:, 'x']
Series([], dtype: object)
```

Multiple columns

Just like selecting *single columns gracefully*, but with a array-like indexer. A dios is returned, with a subset of the existing columns. If no key is present a empty dios is returned.

```
>>> di.aloc[:, ['c', 99, None, 'a', 'x', 'y']]
    a |     c |
===== | ===== |
0   0 | 4   7 |
1   7 | 5  17 |
2  14 | 6  27 |
3  21 | 7  37 |
4  28 | 8  47 |

>>> di.aloc[:, ['x', 'y']]
Empty DictOfSeries
Columns: []

s = pd.Series(dict(a='a', b='x', c='c', foo='d'))
d.aloc[:, s]
    a |     c |     d |
===== | ===== | ===== |
0   0 | 4   7 | 6   0 |
1   7 | 5  17 | 7   1 |
2  14 | 6  27 | 8   2 |
3  21 | 7  37 | 9   3 |
4  28 | 8  47 | 10  4 |
```

Boolean indexing, indexing with pd.Series and slice indexer

Boolean indexer, for example `[True, 'False', 'True', 'False']`, must have the same length than the number of columns, then only columns, where the indexer has a `True` value are selected.

If the key is a **pandas.Series**, its *values* are used for indexing, especially the Series's index is ignored. If a series has boolean values its treated like a boolean indexer, otherwise its treated as a array-like indexer.

A easy way to select all columns, is, to use null-**slice**es, like `.aloc[:,:]` or even simpler `.aloc[:]`. This is just like one would do, with `loc` or `iloc`. Of course slicing with boundaries also work, eg `.loc[:, 'a':'f']`.

See also

- pandas slicing ranges
- pandas boolean indexing

4.7 Selecting Rows a smart way

For scalar and array-like indexer with label values, the keys are handled gracefully, just like with array-like column indexers.

```
>>> di.aloc[1]
   a |       b |       c |       d |
==== | ======= | ======= | ======= |
1  7 | no data | no data | no data |

>>> di.aloc[99]
Empty DictOfSeries
Columns: ['a', 'b', 'c', 'd']

>>> di.aloc[[3,6,7,18]]
    a |    b |     c |    d |
===== | ==== | ===== | ==== |
3  21 | 3  6 | 6  27 | 6  0 |
      | 6  9 | 7  37 | 7  1 |
```

The length of columns can differ:

```
>>> di.aloc[[3,6,7,18]].aloc[[3,6]]
    a |    b |     c |    d |
===== | ==== | ===== | ==== |
3  21 | 3  6 | 6  27 | 6  0 |
      | 6  9 |       |      |
```

4.8 Boolean array-likes as row indexer

For array-like indexer that hold boolean values, the length of the indexer and the length of all column(s) to index must match.

```
>>> di.aloc[[True,False,False,True,False]]
    a |    b |     c |    d |
===== | ==== | ===== | ==== |
0   0 | 2  5 | 4   7 | 6  0 |
3  21 | 5  8 | 7  37 | 9  3 |
```

If the length does not match a `IndexError` is raised:

```
>>> di.aloc[[True,False,False]]
Traceback (most recent call last):
  ...
  IndexError: failed for column a: Boolean index has wrong length: 3 instead of 5
```

This can be tricky, especially if columns have different length:

```
>>> difflen
    a |    b |     c |    d |
===== | ==== | ===== | ==== |
0   0 | 2  5 | 4   7 | 6  0 |
1   7 | 3  6 | 6  27 | 7  1 |
2  14 | 4  7 |       | 8  2 |
```

(continues on next page)

(continued from previous page)

```
>>> difflen.aloc[[False,True,False]]
Traceback (most recent call last):
  ...
  IndexError: Boolean index has wrong length: 3 instead of 2
```

4.9 pandas.Series and boolean pandas.Series as row indexer

When using a pandas.Series as row indexer with `aloc`, all its magic comes to light. The index of the given series align itself with the index of each column separately and is this way used as a filter.

```
>>> s = di['b'] + 100
>>> s
2    105
3    106
4    107
5    108
6    109
Name: b, dtype: int64

>>> di.aloc[s]
    a |    b |     c |    d |
===== | ==== | ===== | ==== |
2  14 | 2  5 | 4   7 | 6  0 |
3  21 | 3  6 | 5  17 |      |
4  28 | 4  7 | 6  27 |      |
      | 5  8 |       |      |
      | 6  9 |       |      |
```

As seen in the example above the series' values are ignored completely. The functionalityis similar to `s1.loc[s2.index]`, with `s1` and `s2` are pandas.Series's, and s2 is the indexer and s1 is one column after the other.

If the indexer series holds boolean values, these are **not** ignored. The series align the same way as explained above, but additional only the `True` values are evaluated. Thus `False`-values are treated like missing indices. The behavior here is analogous to `s1.loc[s2[s2].index]`.

```
>>> boolseries = di['b'] > 6
>>> boolseries
2    False
3    False
4     True
5     True
6     True
Name: b, dtype: bool

>>> di.aloc[boolseries]
    a |    b |     c |    d |
===== | ==== | ===== | ==== |
4  28 | 4  7 | 4   7 | 6  0 |
      | 5  8 | 5  17 |      |
      | 6  9 | 6  27 |      |
```

To evaluate boolean values is a very handy feature, as it can easily used with multiple conditions and also fits nicely with writing those as one-liner:

```
>>> di.aloc[d['b'] > 6]
    a |    b |     c |    d |
===== | ==== | ===== | ==== |
4  28 | 4  7 | 4   7 | 6  0 |
      | 5  8 | 5  17 |      |
      | 6  9 | 6  27 |      |

>>> di.aloc[(d['a'] > 6) & (d['b'] > 6)]
    a |    b |    c |       d |
===== | ==== | ==== | ======= |
4  28 | 4  7 | 4  7 | no data |
```

Note:

Nevertheless, something like `di.aloc[di['a'] > di['b']]` do not work, because the comparison fails, as long as the two series objects not have the same index. But maybe one want to checkout DictOfSeries.index_of().

4.10 Nested-lists as row indexer

It is possible to pass different array-like indexer to different columns, by using nested lists as indexer. The outer list's length must match the number of columns of the dios. The items of the outer list, all must be array-like and not further nested. For example list, pandas.Series, boolean lists or pandas.Series, numpy.arrays... Every inner list-like item is applied as row indexer to the according column.

```
>>> d
    a |    b |     c |     d |
===== | ==== | ===== | ===== |
0   0 | 2  5 | 4   7 | 6   0 |
1   7 | 3  6 | 5  17 | 7   1 |
2  14 | 4  7 | 6  27 | 8   2 |
3  21 | 5  8 | 7  37 | 9   3 |
4  28 | 6  9 | 8  47 | 10  4 |

>>> di.aloc[ [d['a'], [True,False,True,False,False], [], [7,8,10]] ]
    a |    b |       c |     d |
===== | ==== | ======= | ===== |
0   0 | 2  5 | no data | 7   1 |
1   7 | 4  7 |         | 8   2 |
2  14 |      |         | 10  4 |
3  21 |      |         |       |
4  28 |      |         |       |

>>> ar = np.array([2,3])
>>> di.aloc[[ar, ar+1, ar+2, ar+3]]
    a |    b |     c |    d |
===== | ==== | ===== | ==== |
2  14 | 3  6 | 4   7 | 6  0 |
3  21 | 4  7 | 5  17 |      |
```

Even this looks like a 2D-indexer, that are explained in the next section, it is not. In contrast to the 2D-indexer, we also can provide a column key, to pre-filter the columns.

```
>>> di.aloc[[ar, ar+1, ar+3], ['a','b','d']]
    a |    b |    d |
```

(continues on next page)

(continued from previous page)

```
===== | ==== | ==== |
2  14 | 3  6 | 6  0 |
3  21 | 4  7 |      |
```

4.11 The power of 2D-indexer

Overview:

T_O_D_O

CHAPTER

FIVE

COOKBOOK

5.1 Recipes

- select common rows from all columns
- align columns to an other column
- align columns to a given index
- align dios with dios
- get/set values by condition
- apply a value to multiple columns
- *Broadcast array-likes to multiple columns*
- apply a array-like value to multiple columns
- nan-policy - mask vs. drop values, when nan's are inserted (mv to Readme ??)
- itype - when to use, pitfalls and best-practise
- changing the index of series' in dios (one, some, all)
- changing the dtype of series' in dios (one, some, all)
- changing properties of series' in dios (one, some, all)

T_O_D_O

5.2 Broadcast array-likes to multiple columns

T_O_D_O

For the idea behind the Itype concept and its usage read:

CHAPTER

SIX

ITYPE

DictOfSeries holds multiple series, and each series can have a different index length and index type. Differing index lengths are either solved by some aligning magic, or simply fail, if aligning makes no sense (eg. assigning the very same list to series of different lengths (see `.aloc`).

A bigger challange is the type of the index. If one series has an alphabetical index, and another one a numeric index, selecting along columns can fail in every scenario. To keep track of the types of index or to prohibit the inserting of a *not fitting* index type, we introduce the `itype`. This can be set on creation of a Dios and also changed during usage. On change of the itype, all indexes of all series in the dios are casted to a new fitting type, if possible. Different cast-mechanisms are available.

If an itype prohibits some certain types of indexes and a series with a non-fitting index-type is inserted, an implicit type cast is done (with or without a warning) or an error is raised. The warning/error policy can be adjusted via global options.

For implemented methods and module functions, respectively the full module api, see:

CHAPTER

SEVEN

API

7.1 Functions

cast_to_itype(series, itype[, policy, err, ...])	Cast a series (more explicit the type of the index) to fit the itype of a dios.
example_DictOfSeries()	Return a example dios.
get_itype(obj)	Return the according Itype.
is_itype(obj, itype)	Check if obj is a instance of the given itype or its str-alias was given
is_itype_like(obj, itype)	Check if obj is a subclass or a instance of the given itype or any of its subtypes
is_itype_subtype(obj, itype)	Check if obj is a subclass or a instance of a subclass of the given itype
pprint_dios(dios[, max_rows, min_rows, ...])	
to_dios(obj)	

7.1.1 cast_to_itype

dios.**cast_to_itype**(*series*, *itype*, *policy='lossless'*, *err='raise'*, *inplace=False*)

Cast a series (more explicit the type of the index) to fit the itype of a dios.

Return the casted series if successful, None otherwise.

Note: This is very basic number-casting, so in most cases, information from the old index will be lost after the cast.

7.1.2 get_itype

dios.**get_itype**(*obj*)

Return the according Itype.

and return the according Itype :param obj: get the itype fitting for the input :type obj: {itype string, Itype, pandas.Index, instance of pd.index}

Examples

```
>>> get_itype("datetime")
<class 'dios.lib.DtItype'>
```

```
>>> s = pd.Series(index=pd.to_datetime([]))
>>> get_itype(s.index)
<class 'dios.lib.DtItype'>
```

```
>>> get_itype(DtItype)
<class 'dios.lib.DtItype'>
```

```
>>> get_itype(pd.DatetimeIndex)
<class 'dios.lib.DtItype'>
```

7.1.3 is_itype

`dios.`**`is_itype`**(*obj*, *itype*)

Check if obj is a instance of the given itype or its str-alias was given

7.1.4 is_itype_like

`dios.`**`is_itype_like`**(*obj*, *itype*)

Check if obj is a subclass or a instance of the given itype or any of its subtypes

7.1.5 is_itype_subtype

`dios.`**`is_itype_subtype`**(*obj*, *itype*)

Check if obj is a subclass or a instance of a subclass of the given itype

7.1.6 pprint_dios

`dios.`**`pprint_dios`**(*dios*, *max_rows=None*, *min_rows=None*, *max_cols=None*, *na_rep='NaN'*, *empty_series_rep='no data'*, *col_space=None*, *show_dimensions=True*, *col_delim=' | '*, *header_delim='='*)

7.1.7 to_dios

`dios.`**`to_dios`**(*obj*) → dios.dios.DictOfSeries

7.2 Classes

CastPolicy	
DictOfSeries([data, columns, index, itype, . . .])	A data frame where every column has its own index.
DtItype()	
FloatItype()	
IntItype()	
ItypeCastError	
ItypeCastWarning	
ItypeWarning	
NumItype()	
ObjItype()	
Opts	storage class for string values for *dios_options*
OptsFields	storage class for the keys in *dios_options*

7.2.1 CastPolicy

class dios.**CastPolicy**

Bases: `object`

Attributes Summary

force
never
save

Attributes Documentation

force = 'force'

never = 'never'

save = 'save'

7.2.2 DtItype

class dios.**DtItype**

Bases: dios.lib.__Itype

Attributes Summary

min_pdindex
name
subtypes
unique

Attributes Documentation

min_pdindex = DatetimeIndex([], dtype='datetime64[ns]', freq=None)

name = 'datetime'

subtypes = (<class 'pandas.core.indexes.datetimes.DatetimeIndex'>,)

unique = True

7.2.3 FloatItype

class dios.**FloatItype**

Bases: dios.lib.__Itype

Attributes Summary

min_pdindex
name
subtypes
unique

Attributes Documentation

min_pdindex = Float64Index([], dtype='float64')

name = 'float'

subtypes = (<class 'pandas.core.indexes.numeric.Float64Index'>, <class 'float'>)

unique = True

7.2.4 IntItype

class dios.**IntItype**

Bases: dios.lib.__Itype

Attributes Summary

min_pdindex
name
subtypes
unique

Attributes Documentation

min_pdindex = Int64Index([], dtype='int64')

name = 'integer'

subtypes = (<class 'pandas.core.indexes.range.RangeIndex'>, <class 'pandas.core.indexe

unique = True

7.2.5 ItypeCastError

exception dios.**ItypeCastError**

7.2.6 ItypeCastWarning

exception dios.**ItypeCastWarning**

7.2.7 ItypeWarning

exception dios.**ItypeWarning**

7.2.8 NumItype

class dios.**NumItype**

Bases: dios.lib.__Itype

Attributes Summary

min_pdindex
name
subtypes
unique

Attributes Documentation

min_pdindex = Float64Index([], dtype='float64')

name = 'numeric'

subtypes = (<class 'dios.lib.IntItype'>, <class 'dios.lib.FloatItype'>, <class 'pandas

unique = False

7.2.9 ObjItype

class dios.**ObjItype**

Bases: dios.lib.__Itype

Attributes Summary

min_pdindex
name
subtypes
unique

Attributes Documentation

min_pdindex = Index([], dtype='object')

name = 'object'

subtypes = (<class 'dios.lib.DtItype'>, <class 'dios.lib.IntItype'>, <class 'dios.lib.I

unique = False

7.2.10 Opts

class dios.**Opts**

Bases: object

storage class for string values for *dios_options*

Use like so: dios_options[OptsFields.X] = Opts.Y.

See also:

OptsFields keys for the options dict

dios_options options dict for module

Attributes Summary

itype_err
itype_ignore
itype_warn
repr_aligned
repr_indexed

Attributes Documentation

itype_err = 'err'

itype_ignore = 'ignore'

itype_warn = 'warn'

repr_aligned = 'aligned'

repr_indexed = 'indexed'

7.2.11 OptsFields

class dios.**OptsFields**

Bases: `object`

storage class for the keys in *dios_options*

Use like so: `dios_options[OptsFields.X] = Opts.Y`.

See also:

Opts values for the options dict

dios_options options dict for module

Attributes Summary

dios_repr
disp_max_cols
disp_max_rows
disp_min_rows
mixed_itype_warn_policy

Attributes Documentation

dios_repr = 'dios_repr'

disp_max_cols = 'disp_max_vars'

disp_max_rows = 'disp_max_rows '

disp_min_rows = 'disp_min_rows '

mixed_itype_warn_policy = 'mixed_itype_policy'

7.3 Variables

dios_options	Options dictionary for module *dios*.
opdoc	str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

7.3.1 dios_options

dios.**dios_options = {'dios_repr': 'indexed', 'disp_max_rows ': 60, 'disp_max_vars': 10,**

Options dictionary for module *dios*.

Use like so: `dios_options[OptsFields.X] = Opts.Y`.

Items:

- **dios_repr: {'indexed', 'aligned'} default: 'indexed'**

  dios default representation if:

  - *indexed*: show every column with its index
  - *aligned*: transform to pandas.DataFrame with indexed merged together.
- **disp_max_rows** [int] Maximum numbers of row before truncated to *disp_min_rows* in representation of DictOfSeries
- **disp_min_rows** [int] min rows to display if *max_rows* is exceeded
- **disp_max_vars** [int] Maximum numbers of columns before truncated representation
- **mixed_itype_policy** [{'warn', 'err', 'ignore'}] How to inform user about mixed Itype

See also:

OptsFields keys for the options dict

Opts values for the options dict

7.3.2 opdoc

dios.**opdoc = "Options dictionary for module `dios`.\n\nUse like so: ``dios_options[OptsFie**

str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

or browse the Index..

dummy file to be able to link to index

CHAPTER

EIGHT

INDEX

PYTHON MODULE INDEX

d

dios, 35

INDEX

A

all() (*dios.DictOfSeries method*), 7
aloc (*dios.DictOfSeries attribute*), 5
any() (*dios.DictOfSeries method*), 7
apply() (*dios.DictOfSeries method*), 7
astype() (*dios.DictOfSeries method*), 9
at (*dios.DictOfSeries attribute*), 5

C

cast_policy (*dios.DictOfSeries attribute*), 5
cast_to_itype() (*in module dios*), 35
CastPolicy (*class in dios*), 37
clear() (*dios.DictOfSeries method*), 9
columns (*dios.DictOfSeries attribute*), 5
copy() (*dios.DictOfSeries method*), 9
copy_empty() (*dios.DictOfSeries method*), 9

D

debugDf (*dios.DictOfSeries attribute*), 5
DictOfSeries (*class in dios*), 3
dios
    module, 35
dios_options (*in module dios*), 42
dios_repr (*dios.OptsFields attribute*), 42
disp_max_cols (*dios.OptsFields attribute*), 42
disp_max_rows (*dios.OptsFields attribute*), 42
disp_min_rows (*dios.OptsFields attribute*), 42
dropempty() (*dios.DictOfSeries method*), 10
dropna() (*dios.DictOfSeries method*), 10
DtItype (*class in dios*), 38
dtypes (*dios.DictOfSeries attribute*), 5

E

empty (*dios.DictOfSeries attribute*), 5
equals() (*dios.DictOfSeries method*), 10
example_DictOfSeries() (*in module dios*), 19

F

FloatItype (*class in dios*), 38
for_each() (*dios.DictOfSeries method*), 10
force (*dios.CastPolicy attribute*), 37

G

get() (*dios.DictOfSeries method*), 11
get_itype() (*in module dios*), 35

H

hasnans() (*dios.DictOfSeries method*), 11

I

iat (*dios.DictOfSeries attribute*), 6
iloc (*dios.DictOfSeries attribute*), 6
index_of() (*dios.DictOfSeries method*), 11
indexes (*dios.DictOfSeries attribute*), 6
IntItype (*class in dios*), 39
is_itype() (*in module dios*), 36
is_itype_like() (*in module dios*), 36
is_itype_subtype() (*in module dios*), 36
isdata() (*dios.DictOfSeries method*), 12
isempty() (*dios.DictOfSeries method*), 12
isin() (*dios.DictOfSeries method*), 12
isna() (*dios.DictOfSeries method*), 12
isnull() (*dios.DictOfSeries method*), 12
items() (*dios.DictOfSeries method*), 12
iteritems() (*dios.DictOfSeries method*), 12
iterrows() (*dios.DictOfSeries method*), 12
itype (*dios.DictOfSeries attribute*), 6
itype_err (*dios.Opts attribute*), 41
itype_ignore (*dios.Opts attribute*), 41
itype_warn (*dios.Opts attribute*), 41
ItypeCastError, 39
ItypeCastWarning, 39
ItypeWarning, 39

K

keys() (*dios.DictOfSeries method*), 13

L

lengths (*dios.DictOfSeries attribute*), 6
loc (*dios.DictOfSeries attribute*), 6

M

max() (*dios.DictOfSeries method*), 13

memory_usage() (*dios.DictOfSeries method*), 13
min() (*dios.DictOfSeries method*), 13
min_pdindex (*dios.DtItype attribute*), 38
min_pdindex (*dios.FloatItype attribute*), 38
min_pdindex (*dios.IntItype attribute*), 39
min_pdindex (*dios.NumItype attribute*), 40
min_pdindex (*dios.ObjItype attribute*), 40
mixed_itype_warn_policy (*dios.OptsFields attribute*), 42
module
    dios, 35

N

name (*dios.DtItype attribute*), 38
name (*dios.FloatItype attribute*), 38
name (*dios.IntItype attribute*), 39
name (*dios.NumItype attribute*), 40
name (*dios.ObjItype attribute*), 40
never (*dios.CastPolicy attribute*), 37
notempty() (*dios.DictOfSeries method*), 13
notna() (*dios.DictOfSeries method*), 13
notnull() (*dios.DictOfSeries method*), 13
NumItype (*class in dios*), 39

O

ObjItype (*class in dios*), 40
opdoc (*in module dios*), 43
Opts (*class in dios*), 40
OptsFields (*class in dios*), 41

P

pop() (*dios.DictOfSeries method*), 13
popitem() (*dios.DictOfSeries method*), 13
pprint_dios() (*in module dios*), 36

R

reduce_columns() (*dios.DictOfSeries method*), 13
repr_aligned (*dios.Opts attribute*), 41
repr_indexed (*dios.Opts attribute*), 41

S

save (*dios.CastPolicy attribute*), 37
setdefault() (*dios.DictOfSeries method*), 14
size (*dios.DictOfSeries attribute*), 6
squeeze() (*dios.DictOfSeries method*), 14
subtypes (*dios.DtItype attribute*), 38
subtypes (*dios.FloatItype attribute*), 38
subtypes (*dios.IntItype attribute*), 39
subtypes (*dios.NumItype attribute*), 40
subtypes (*dios.ObjItype attribute*), 40

T

to_csv() (*dios.DictOfSeries method*), 14
to_df() (*dios.DictOfSeries method*), 15
to_dios() (*in module dios*), 36
to_string() (*dios.DictOfSeries method*), 16

U

unique (*dios.DtItype attribute*), 38
unique (*dios.FloatItype attribute*), 38
unique (*dios.IntItype attribute*), 39
unique (*dios.NumItype attribute*), 40
unique (*dios.ObjItype attribute*), 40
update() (*dios.DictOfSeries method*), 17

V

values (*dios.DictOfSeries attribute*), 7