TidyFrame

Release 0.0.3

Mar 08, 2020

Contents:

TidyFrame help clean data to tidy DadaFrame quickly. TidyFrame provide some function to help you extract, transform, load data easily.

CHAPTER 1

Overview

TidyFrame help clean data to tidy DadaFrame quickly. TidyFrame provide some function to help you extract, transform, load data easily. This page help you know how to use tidyframe function.

- Make tranform nest dictionary easily

```
>>> from tidyframe import flatten_dict
>>> nest_dict = {
...     'a': 1,
...     'b': [1, 2],
...     'c': {
...         'cc1': 3,
...         'cc2': 4
...     },
...     'd': {
...         'd1': 5,
...         'd2': {
...             'dd1': 6,
...             'dd2': 7
...         }
...     }
... }
>>> flatten_dict(nest_dict)
{'a': 1, 'b': [1, 2], 'c_cc1': 3, 'c_cc2': 4, 'd_d1': 5, 'd_d2_dd1': 6, 'd_d2_dd2
↪': 7}
>>> flatten_dict(nest_dict, inner_name=True)
{'a': 1, 'b': [1, 2], 'cc1': 3, 'cc2': 4, 'd1': 5, 'dd1': 6, 'dd2': 7}
```

- Make select columns, reorder columns easily

```
>>> import numpy as np
>>> import pandas as pd
>>> from tidyframe import select, reorder_columns
>>> df = pd.DataFrame(np.array(range(10)).reshape(2, 5),
...                   columns=list('abcde'),
```

(continues on next page)

(continued from previous page)

```
...                     index=['row_1', 'row_2'])
>>> select(df, columns=['b', 'd'])
    b  d
row_1  1  3
row_2  6  8
>>> select(df, columns_minus=['b', 'd'])
    a  c  e
row_1  0  2  4
row_2  5  7  9
>>> select(df, pattern='[a|b]')
    a  b
row_1  0  1
row_2  5  6
>>> df = pd.DataFrame([{'a': 1, 'b': 1, 'c': 1, 'd': 1, 'e': 2}])
>>> df_reorder = reorder_columns(df, ['b', 'c'], last_columns=['a', 'd'])
>>> df_reorder
b  c  e  a  d
0  1  1  2  1  1
```

- Wapper SQLAlchemy function to help you create table, insert table, drop table easily
  - Help find suitable data type and string length for database table(you can change length by sqlalchemy Table object if you want)
  - Help you bulk insert records and find insert fail records quickly

```
>>> import pandas as pd
>>> from sqlalchemy import create_engine
>>> from datetime import datetime
>>> from tidyframe import (create_table, load_table_schema, bulk_insert)
>>>
>>> engine = create_engine('sqlite:///test_for_create_table.db')
>>> df = pd.DataFrame()
>>> df['a'] = list('abc')
>>> df['b'] = 1
>>> df['c'] = 1.3
>>> df['d'] = [pd.np.nan, 10, 1.4]
>>> df['e'] = ['adev', pd.NaT, '']
>>> df['f'] = [datetime.now(), None, datetime.now()]
>>> df['g'] = [True, False, True]
>>> df['h'] = 2147483647 * 2
>>> create_table(df,
...              'test_table',
...              engine,
...              primary_key=['a'],
...              nvarchar_columns=['e'],
...              non_nullable_columns=['d'],
...              create=False)
Table('test_table', MetaData(bind=Engine(sqlite:///test_for_create_table.db)),
↪ Column('a', CHAR(length=1), table=<test_table>, primary_key=True,␣
↪nullable=False), Column('b', Integer(), table=<test_table>), Column('c',␣
↪Float(), table=<test_table>), Column('d', Float(), table=<test_table>,␣
↪nullable=False), Column('e', NVARCHAR(length=8), table=<test_table>),␣
↪Column('f', DATETIME(), table=<test_table>), Column('g', BOOLEAN(), table=
↪<test_table>), Column('h', Integer(), table=<test_table>), schema=None)
>>>
>>> create_table(df,
```

(continues on next page)

(continued from previous page)

```
...                 'test_table_create',
...                 engine,
...                 primary_key=['a'],
...                 nvarchar_columns=['e'],
...                 non_nullable_columns=['d'],
...                 create=True)
True
>>>
>>> engine = create_engine("mysql://root:sdysuD4UXaynu84u@127.0.0.1/test_db")
>>> df = pd.DataFrame()
>>> df["a"] = ["a"] * 10000
>>> df["b"] = [1] * 10000
>>> df["c"] = [1.3] * 10000
>>>
>>> create_table(df, "want_insert_table", engine, create=True)
True
>>> table = load_table_schema("want_insert_table", engine)
>>>
>>> df.iloc[0,0]= "abc"
>>> df.iloc[-1,0]= "abc"
>>>
>>> insert_fail_records = bulk_insert(df.to_dict("record"),
...                                   table,
...                                   engine,
...                                   batch_size=100)
>>> len(insert_fail_records)
200
>>>
>>> insert_fail_records = bulk_insert(df.to_dict("record"),
...                                   table,
...                                   engine,
...                                   batch_size=100,
...                                   only_insert_fail=True)
>>> len(insert_fail_records)
2
```

CHAPTER 2

Dictionary Function

deal with dict object

tools.dict.**flatten_dict**(*source_dict*, *name_delimiter='_'*, *inner_name=False*)

flatten nest dict

Parameters

- **source_dict** (*nest dict*) –
- **name_delimiter** (*flatten name delimiter(non-use when inner_name is True)*) –
- **inner_name** (*False, use innermost name as retrun dict key or not*) –

Returns

Return type flatten dict

Examples

```
>>> from tidyframe import flatten_dict
>>> nest_dict = {
...     'a': 1,
...     'b': [1, 2],
...     'c': {
...         'cc1': 3,
...         'cc2': 4
...     },
...     'd': {
...         'd1': 5,
...         'd2': {
...             'dd1': 6,
...             'dd2': 7
...         }
```

(continues on next page)

(continued from previous page)

```
...     }
... }
>>> flatten_dict(nest_dict)
{'a': 1, 'b': [1, 2], 'c_cc1': 3, 'c_cc2': 4, 'd_d1': 5, 'd_d2_dd1': 6, 'd_d2_dd2
↪': 7}
>>> flatten_dict(nest_dict, inner_name=True)
{'a': 1, 'b': [1, 2], 'cc1': 3, 'cc2': 4, 'd1': 5, 'dd1': 6, 'dd2': 7}
```

CHAPTER 3

Select DataFrame

Easy Select Column Method from Pandas DataFrame

tools.select.**get_batch_dataframe**(*df*, *batch_size=100*)

split DataFrame to sub-DataDrame and each sub-DataDrame row size is batch_size

Parameters

- **df** (*Pandas DataFrame*) –
- **batch_size** (*number of records in each sub-dataframe(default: 100)*) –

Returns

Return type DataFrame generator

Examples

```
>>> import pandas as pd
>>> from tidyframe import get_batch_dataframe
>>> df = pd.DataFrame()
>>> df['col_1'] = list("abcde")
>>> df['col_2'] = [1, 2, 3, 4, 5]
>>> dfs = [ x for x in get_batch_dataframe(df,2)]
>>> dfs[-1]
    col_1  col_2
4       e      5
>>> [ x.shape[0] for x in dfs]
[2, 2, 1]
```

tools.select.**reorder_columns**(*df*, *columns=None*, *pattern=None*, *last_columns=None*)

reorder columns of pandas DataFrame

Parameters

- **df** (*Pandas DataFrame*) –

- **columns** (*list which want to head column name(non-use if pattern is not None)*) –
- **pattern** (*regular expression pattern which let selected columns be at head columns*) –
- **last_columns** (*list which want to last column name*) –

Returns

Return type Pandas DataFrame

Examples

```
>>> import pandas as pd
>>> from tidyframe import reorder_columns
>>> df = pd.DataFrame([{'a': 1, 'b': 1, 'c': 1, 'd': 1, 'e': 2}])
>>> df_reorder = reorder_columns(df, ['b', 'c'], last_columns=['a', 'd'])
>>> df_reorder
b  c  e  a  d
0  1  1  2  1  1
```

tools.select.**select**(*df*, *columns=None*, *columns_minus=None*, *columns_between=None*, *pattern=None*, *copy=False*)

Select Pandas DataFrame Columns

Parameters

- **df** (*Pandas DataFrame*) –
- **columns_minus** (*column which want to remove*) –
- **columns_between** (*list with two element, select columns bwtween two columns*) –
- **pattern** (*regular expression or list of regular expression, return match columns*) –
- **copy** (*whether return deep copy DataFrame*) –

Returns

Return type Pandas DataFrame

Examples

```
>>> import numpy as np
>>> import pandas as pd
>>> from tidyframe import select
>>> df = pd.DataFrame(np.array(range(10)).reshape(2, 5),
...                   columns=list('abcde'),
...                   index=['row_1', 'row_2'])
>>> select(df, columns=['b', 'd'])
     b  d
row_1  1  3
row_2  6  8
>>> select(df, columns_minus=['b', 'd'])
     a  c  e
row_1  0  2  4
```

(continues on next page)

(continued from previous page)

```
row_2  5  7  9
>>> select(df, pattern='[a|b]')
    a  b
row_1  0  1
row_2  5  6
```

tools.select.**select_index**(*x*, *i*, *otherwise=nan*)

Select by index and Catch all Exception

Parameters

- **x** (*array*) –
- **i** (*index*) –
- **otherwise** (*fill value if exist exception*) –

Returns

Return type x[i] if not exception happen else return otherwise

CHAPTER 4

Database Function

Wapper SQLAlchemy function to help you create table, insert table, drop table easily.

tools.database.**bulk_insert**(*records*, *table*, *con*, *batch_size=10000*, *pool_size=1*, *only_insert_fail=False*)

bulk insert records(list dict)

Parameters

- **records** (*list of dict*) –
- **table** (*sqlalchemy Table object(you can get from function load_table_schema)*) –
- **con** (*sqlalchemy.engine.Engine or sqlite3.Connection*) –
- **batch_size** (*batch size for bluk insert*) –
- **pool_size** (*Int(default: 1), number of threads for insert records*) –
- **only_insert_fail** (*Bool(default: False), only return record wihich insert fail*) –

Returns

Return type list of record which insert fail in batch records or list of record which fail to insert database

Examples

```
>>> import pandas as pd
>>> from sqlalchemy import create_engine
>>> from tidyframe import (create_table, load_table_schema, bulk_insert)
>>>
>>> engine = create_engine("mysql://root:sdysuD4UXaynu84u@127.0.0.1/test_db")
>>> df = pd.DataFrame()
```

(continues on next page)

(continued from previous page)

```
>>> df["a"] = ["a"] * 10000
>>> df["b"] = [1] * 10000
>>> df["c"] = [1.3] * 10000
>>>
>>> create_table(df, "want_insert_table", engine, create=True)
True
>>> table = load_table_schema("want_insert_table", engine)
>>>
>>> df.iloc[0,0]= "abc"
>>> df.iloc[-1,0]= "abc"
>>>
>>> insert_fail_records = bulk_insert(df.to_dict("record"),
...                                   table,
...                                   engine,
...                                   batch_size=100)
>>> len(insert_fail_records)
200
>>>
>>> insert_fail_records = bulk_insert(df.to_dict("record"),
...                                   table,
...                                   engine,
...                                   batch_size=100,
...                                   only_insert_fail=True)
>>> len(insert_fail_records)
2
```

tools.database.**copy_table_schema**(*source_table*, *target_table*, *source_con*, *target_con*, *omit_collation=False*, *create=True*, *add_columns=[]*)

Copy table schema from database to another database

Parameters

- **source_table** (*source table name in database*) –
- **target_table** (*target table name*) –
- **source_con** (*sqlalchemy.engine.Engine or sqlite3.Connection, source engine*) –
- **target_con** (*sqlalchemy.engine.Engine or sqlite3.Connection, target engine*) –
- **omit_collation** (*Bool(default: False), omit all char collation*) –
- **create** (*Bool(default: True), direct create table in database*) –
- **add_columns** (*list of column object*) –

Returns

Return type sqlalchemy Table object or True

Examples

```
>>> import pandas as pd
>>> from sqlalchemy import (create_engine, VARCHAR, Column, DateTime)
```

(continues on next page)

(continued from previous page)

```
>>> from datetime import datetime
>>> from tidyframe import copy_table_schema
>>>
>>> engine = create_engine('sqlite:///source.db')
>>> engine_target = create_engine('sqlite:///target.db')
>>> df = pd.DataFrame()
>>> df['a'] = list('abc')
>>> df['b'] = 1
>>> df['c'] = 1.3
>>> df['d'] = [pd.np.nan, 10, 1.4]
>>> df['e'] = ['adev', pd.NaT, '']
>>> df['f'] = [datetime.now(), None, datetime.now()]
>>> df['g'] = [True, False, True]
>>> df.shape
(3, 7)
>>> df.to_sql('raw_table', engine, index=False)
>>> copy_table_schema('raw_table',
...                   'target_table',
...                   source_con=engine,
...                   target_con=engine_target,
...                   add_columns=[Column('last_maintain_date', DateTime())],
...                   omit_collation=True,
...                   create=True)
True
>>> pd.read_sql_table('target_table', engine_target).shape
(0, 8)
```

tools.database.**create_table**(*df*, *name*, *con*, *primary_key=[]*, *nvarchar_columns=[]*, *non_nullable_columns=[]*, *dtype=None*, *create=True*, *all_nvarchar=False*, *base_char_type=CHAR()*, *base_nchar_type=NVARCHAR()*, *base_int_type=Integer()*, *base_bigint_type=BigInteger()*, *base_float_type=Float()*, *base_boolean_type=BOOLEAN()*)

Create sqlalchemy Table object for create table in database

Parameters

- **df** (*Pandas DataFrame*) –
- **con** (*sqlalchemy.engine.Engine or sqlite3.Connection*) –
- **name** (*string, name of SQL table*) –
- **primary_key** (*list, primary key columns*) –
- **nvarchar_columns** (*list, nvarchar columns*) –
- **non_nullable_columns** (*list, non-nullable columns*) –
- **dtype** (*dict, optional, specifying the datatype for columns. The keys should be the column names and the values should be the SQLAlchemy types or strings for the sqlite3 legacy mode.*) –
- **all_nvarchar** (*Bool, all string column use NVARCHAR or not*) –
- **create** (*Bool(default: False), direct create table in database*) –

Returns

Return type sqlalchemy Table object or True

Example

```
>>> import pandas as pd
>>> from sqlalchemy import create_engine
>>> from datetime import datetime
>>> from tidyframe import create_table
>>>
>>> engine = create_engine('sqlite:///test_for_create_table.db')
>>> df = pd.DataFrame()
>>> df['a'] = list('abc')
>>> df['b'] = 1
>>> df['c'] = 1.3
>>> df['d'] = [pd.np.nan, 10, 1.4]
>>> df['e'] = ['adev', pd.NaT, '']
>>> df['f'] = [datetime.now(), None, datetime.now()]
>>> df['g'] = [True, False, True]
>>> df['h'] = 2147483647 * 2
>>> create_table(df,
...              'test_table',
...              engine,
...              primary_key=['a'],
...              nvarchar_columns=['e'],
...              non_nullable_columns=['d'],
...              create=False)
Table('test_table', MetaData(bind=Engine(sqlite:///test_for_create_table.db)),␣
↪Column('a', CHAR(length=1), table=<test_table>, primary_key=True,␣
↪nullable=False), Column('b', Integer(), table=<test_table>), Column('c',␣
↪Float(), table=<test_table>), Column('d', Float(), table=<test_table>,␣
↪nullable=False), Column('e', NVARCHAR(length=8), table=<test_table>), Column('f
↪', DATETIME(), table=<test_table>), Column('g', BOOLEAN(), table=<test_table>),␣
↪Column('h', Integer(), table=<test_table>), schema=None)
>>>
>>> create_table(df,
...              'test_table_create',
...              engine,
...              primary_key=['a'],
...              nvarchar_columns=['e'],
...              non_nullable_columns=['d'],
...              create=True)
True
```

tools.database.**drop_table**(*name*, *con*)

drop table from database

Parameters

- **name** (*string, name of SQL table*) –
- **con** (*sqlalchemy.engine.Engine or sqlite3.Connection*) –

Returns

Return type True

Examples

```
>>> import pandas as pd
>>> from sqlalchemy import create_engine
>>> from tidyframe import drop_table
>>>
>>> engine = create_engine("sqlite:///raw_table.db")
>>> df = pd.DataFrame([{"a": 1, "b": 2}, {"a": 1, "b": 2}])
>>> df.to_sql("raw_table", engine)
>>> drop_table("raw_table", engine)
True
```

tools.database.**fit_table_schema_type**(*df*, *table*)

Fit DataFrame to table schema type, let you can use DataFrame.to_sql directly if table is exist. Limit: Not tranform column dtype if python_type is str and column dtype is object

Parameters

- **df** (*Pandas DataFrame*) –
- **table** (*Table object*) –

Returns

Return type None

tools.database.**get_create_table_script**(*table*)

get create table script

Parameters **table** (*sqlalchemy Table object*) –

Returns

Return type string which sqlalchemy create for create table

Examples

```
>>> import pandas as pd
>>> from sqlalchemy import create_engine
>>> from tidyframe import create_table, get_create_table_script
>>>
>>> engine = create_engine('sqlite:///testing_get_create_table_script.db')
>>> df = pd.DataFrame()
>>> df['a'] = list('abc')
>>> df['b'] = 1
>>> df['c'] = 1.3
>>> table = create_table(df,
...                      'test_table',
...                      engine,
...                      primary_key=['a'],
...                      nvarchar_columns=['e'],
...                      non_nullable_columns=['d'],
...                      create=False)
>>> create_table_script = get_create_table_script(table)
```

tools.database.**load_table_schema**(*name*, *con*)

load table schema from database

Parameters

- **name** (*string, name of SQL table*) –
- **con** (*sqlalchemy.engine.Engine or sqlite3.Connection*) –

Returns

Return type sqlalchemy Table object

Example

```
>>> import pandas as pd
>>> from sqlalchemy import (create_engine, Table, MetaData)
>>> from tidyframe import (load_table_schema, create_table)
>>>
>>> engine = create_engine('sqlite:///load_table_schema.db')
>>> num_row = 100000
>>> df = pd.DataFrame()
>>> df['a'] = ['a'] * num_row
>>> df['b'] = ['b'] * num_row
>>> df['c'] = ['c'] * num_row
>>> create_table(df, 'test_table', engine, create=True)
True
>>> records = df.to_dict('record')
>>> table_b = load_table_schema('test_table', engine)
>>> table_b
Table('test_table', MetaData(bind=Engine(sqlite:///load_table_schema.db)), Column(
↪'a', CHAR(length=1), table=<test_table>), Column('b', CHAR(length=1), table=
↪<test_table>), Column('c', CHAR(length=1), table=<test_table>), schema=None
```

CHAPTER 5

DataFrame Combination

All combination rows from list of DataFrame

tools.combination.**combination**(*dfs*)

All combination rows from list of DataFrame

Parameters **dfs** (*list of Pandas DataFrame*) –

Returns

Return type Pandas DataFrame

Examples

```
>>> import pandas as pd
>>> from tidyframe import combination
>>> df_a = pd.DataFrame({'a1': list('ABC'), 'a2': list('CDE')})
>>> df_b = pd.DataFrame({'b1': list('01234'), 'b2': list('56789')})
>>> df_c = pd.DataFrame({'c1': list('pq'), 'c2': list('rs')})
>>> combination([df_a, df_b, df_c])
    index_0 a1 a2  index_1 b1 b2  index_2 c1 c2
0         0  A  C        0  0  5        0  p  r
1         0  A  C        0  0  5        1  q  s
2         0  A  C        1  1  6        0  p  r
3         0  A  C        1  1  6        1  q  s
4         0  A  C        2  2  7        0  p  r
5         0  A  C        2  2  7        1  q  s
6         0  A  C        3  3  8        0  p  r
7         0  A  C        3  3  8        1  q  s
8         0  A  C        4  4  9        0  p  r
9         0  A  C        4  4  9        1  q  s
10        1  B  D        0  0  5        0  p  r
11        1  B  D        0  0  5        1  q  s
12        1  B  D        1  1  6        0  p  r
13        1  B  D        1  1  6        1  q  s
```

(continues on next page)

(continued from previous page)

```
14      1  B  D      2  2  7      0  p  r
15      1  B  D      2  2  7      1  q  s
16      1  B  D      3  3  8      0  p  r
17      1  B  D      3  3  8      1  q  s
18      1  B  D      4  4  9      0  p  r
19      1  B  D      4  4  9      1  q  s
20      2  C  E      0  0  5      0  p  r
21      2  C  E      0  0  5      1  q  s
22      2  C  E      1  1  6      0  p  r
23      2  C  E      1  1  6      1  q  s
24      2  C  E      2  2  7      0  p  r
25      2  C  E      2  2  7      1  q  s
26      2  C  E      3  3  8      0  p  r
27      2  C  E      3  3  8      1  q  s
28      2  C  E      4  4  9      0  p  r
29      2  C  E      4  4  9      1  q  s
```

CHAPTER 6

Separate Function

Separate string list to Pandas DataFrame

tools.separate.**separate**(*series*, *index=None*, *columns=None*, *otherwise=nan*)

Separate string list to Pandas DataFrame

Parameters

- **series** (*list of list or Series of list*) –
- **index** (*filter return index*) –
- **columns** (*return column name of DataFrame*) –
- **otherwise** (*numpy.NaN, fill value of not exist value*) –

Returns

Return type Pandas DataFrame with split each element of series to column

Examples

```
>>> from tidyframe.tools import separate
>>> df = pd.DataFrame({'full_string': ['a b c d e z', 'f g h i']},
...                   index=['row_1', 'row_2'])
>>> series = df.full_string.str.split(' ')
>>> separate(series)
    col_0 col_1 col_2 col_3 col_4 col_5
row_1     a     b     c     d     e     z
row_2     f     g     h     i   NaN   NaN
```

CHAPTER 7

String Function

string function

tools.string.**replace_by_dict**(*source_string*, *mapping*)
replace string by dictionary

Parameters

- **source_string**(*string*) –
- **mapping**(*dict*) –

Returns **return_string**

Return type string

tools.string.**strip_whitespace**(*source_string*)
replace all space character in source_string

Parameters **source_string**(*string*) –

Returns **return_string**

Return type string

tools.string.**strip_whitespace_include_newline**(*source_string*)
replace all space character in source_string include n r and t

Parameters **source_string**(*string*) –

Returns **return_string**

Return type string

CHAPTER 8

Windows Function

tools.window.**apply_cum**(*series*, *cum_func=<function <lambda>>*, *judge_func=<function <lambda>>*, *init_value=0*)

Apply Cumulative Function on Series or list

Parameters

- **series** (*list or series*) –
- **cum_func** (*cumulative function with two parameters*) –
- **judge_func** (*judge function which return value is True or False for reset cumulative value*) –
- **init_value** (*reset value if judge function result is True*) –

Returns

Return type DataFrame with three columns(cum_value, index_first, index_last)

Example

```
>>> import numpy as np
>>> from tidyframe import apply_cum
>>> series = np.random.randint(1, 6, 10)
>>> cum_func = lambda x, y: x * y
>>> judge_func = lambda x: x > 10
>>> apply_cum(series, cum_func, init_value=1)
cum_value  index_first  index_last
0          4          True       False
1          4         False       False
2         20         False       False
3         20         False       False
4        100         False       False
5        200         False       False
6        200         False       False
```

(continues on next page)

(continued from previous page)

```
7        600        False       False
8        600        False       False
9       2400        False       False
```

CHAPTER 9

Case When in Series

tools.case_when.**coalesce**(*df*, *columns*, *default_value=nan*)

Coalesce column by list of column

Parameters

- **df** (*Pandas DataFrame*) –
- **columns** (*list or pandas index*) –
- **default_value** (*value which replace None or NaN in return series*) –

Returns

Return type Pandas Series

Examples

```
>>> import pandas as pd
>>> from tidyframe import coalesce
>>> df = pd.DataFrame()
>>> df['a'] = [None, pd.np.NaN, pd.np.nan, pd.np.nan]
>>> df['b'] = [None, 4, 6, pd.np.nan]
>>> df['c'] = [None, pd.np.NaN, 6, pd.np.nan]
>>> coalesce(df, ['a', 'b', 'c'], default_value=10)
0    10.0
1     4.0
2     6.0
3    10.0
Name: a, dtype: float64
```

tools.case_when.**fillna**(**args*)

Fill non null value

Parameters ***args** (*list or series*) –

Returns

Return type list

Examples

```
>>> import pandas as pd
>>> from tidyframe import fillna
>>> fillna([None] * 3, [1, pd.np.NaN, None], [1, 2, 3])
[1, 2, 3]
```

tools.case_when.**nvl**(*obj*, *default=nan*, *copy=True*)

replace None or NaN value by default

Parameters

- **obj** (*Series, list, or primitive variable types*) –
- **default** (*defalut*) –
- **copy** (*copy list or not if obj is list type*) –

Returns

Return type series or list or primitive variable types

Examples

```
>>> import pandas as pd
>>> from tidyframe.tools import nvl
>>> nvl(None, 10)
10
>>> test_list = [0, 1, None, pd.np.NaN]
>>> test_series = pd.Series(test_list)
>>> nvl(test_series, 10)
0     0.0
1     1.0
2    10.0
3    10.0
dtype: float64
```

tools.case_when.**try_expect_raw**(*function*)

A decorator which return first args when execpt happen

Parameters ***args** (*list or series*) –

Returns

Return type list

Examples

```
>>> from tidyframe import try_expect_raw
>>> my_sum = try_expect_raw(lambda x, y: x + y)
>>> my_sum(1, y='a')
1
```

CHAPTER 10

Transform DataFrame

Convert Pandas DataFrame to nest DataFrame

transform.**add_columns**(*df*, *columns*, *default=None*, *deepcopy=False*)

Add column if column is not exist

Parameters

- **df** (*pandas DataFrame*) –
- **columns** (*list, add column names*) –
- **default** (*list or a object(defalut: NaN)*) –
- **deepcopy** (*bool, deepcopy df or not(default: True)*) –

Returns

Return type pandas DataFrame

Examples

```
>>> import pandas as pd
>>> from tidyframe import add_columns
>>> df = pd.DataFrame()
>>> df['a'] = [1, 6]
>>> df['b'] = [2, 7]
>>> df['c'] = [3, 8]
>>> df['d'] = [4, 9]
>>> df['e'] = [5, 10]
>>> add_columns(df, columns=['a', 'f'], default=[30, [10, 11]])
>>> df
a  b  c  d   e   f
0  1  2  3  4   5  10
1  6  7  8  9  10  11
```

transform.**apply_window**(*df*, *func*, *partition=None*, *columns=None*)
apply window function in DataFrame

Parameters

- **df** (*DataFrameGroupBy or DataFrame*) –
- **func** (*list of function*) –
- **partition** (*list of partition columns*) –
- **columns** (*list of columns which need to apply func*) –

Returns

Return type Pandas Series

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> from tidyframe import apply_window
>>>
>>> iris = datasets.load_iris()
>>> df = pd.DataFrame({"range":[1,2,3,4,5,6],"target":[1,1,1,2,2,2]})
>>> apply_window(df, np.mean, partition=['target'], columns=df.columns[1])
0    1
1    1
2    1
3    2
4    2
5    2
Name: target, dtype: int64
```

transform.**nest**(*df*, *columns=[]*, *columns_minus=[]*, *columns_between=[]*, *key='data'*, *copy=False*)
Nest repeated values

Parameters

- **df** (*DataFrameGroupBy or DataFrame*) –
- **columns** (*list or index, nest columns*) –
- **columns_minus** (*list or index, columns which do not want to nest*) – (must choose one of columns and columns_minus)
- **columns_between** (*list with length 2, assigin nest columns between to two columns*) –
- **copy** (*False, return DataFrame using copy.deepcopy*) –

transform.**rolling**(*list_object*, *window_size*, *missing=nan*)
Rolling list of object

Parameters

- **list_object** (*list of objects*) –
- **window_size** (*rolling windows size*) –
- **missing** (*default value if missing value in rolling window*) –

Returns

Return type list of list

Examples

```
>>> import pandas as pd
>>> from tidyframe import rolling
>>> a = list(range(10))
>>> pd.DataFrame({'a': a, 'b': rolling(a, 3)})
a              b
0  0  [nan, nan, 0]
1  1     [nan, 0, 1]
2  2        [0, 1, 2]
3  3        [1, 2, 3]
4  4        [2, 3, 4]
5  5        [3, 4, 5]
6  6        [4, 5, 6]
7  7        [5, 6, 7]
8  8        [6, 7, 8]
9  9        [7, 8, 9]
```

transform.**to_dataframe**(*data*, *index_name='index'*)

Change list of Pandas Serice to Pandas DataFrame

Parameters

- **data** (*list of pandas Series*) –
- **index_name** (*return index DataFrame column name*) –

Examples

```
>>> import pandas as pd
>>> from tidyframe import to_dataframe
>>> list_series = [
...     pd.Series([1, 2], index=['i_1', 'i_2']),
...     pd.Series([3, 4], index=['i_1', 'i_2'])
... ]
>>> to_dataframe(list_series)
   i_1  i_2 index
0    1    2  None
1    3    4  None
```

transform.**unnest**(*df*, *drop=[]*, *copy=False*)

Inverse Nest DataFrame

Parameters

- **df** (*DataFrame with Series of Dataframe*) –
- **drop** (*list of column which do not return*) –

CHAPTER 11

Reshape DataFrame

Convert Pandas DataFrame Between Wide Format and Long Format

reshape.**gather**(*df*, *key_col=None*, *key='key'*, *value='value'*, *dropna=True*)

Gather column to key-value pairs

Parameters

- **df** (*DataFrame*) –
- **key** (*return DataFrame column name of key*) –
- **value** (*return DataFrame column name fo value*) –
- **dropna** (*boolean, default True*) – Whether to drop rows in the resulting Frame/Series with no valid values

Returns

Return type Pandas DataFrame

reshape.**spread**(*df*, *row_index*, *key*)

Spread key-value pair to multiple columns

Parameters

- **df** (*long format Dataframe*) –
- **row_index** (*transform to wide format row index column*) –
- **key** (*key column which return DataFrame column name*) –

Returns

Return type Pandas DataFrame

CHAPTER 12

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

t

Index