Data Wrangling

Python basics

Kunal Khurana

2024-02-23

Table of contents

Data Wrangling: Join,Combine, and Reshape

Hierarchical Indexing

- Reordering and Sorting levels
- Summary statitics by level
- Indexing with DataFrame's Columns

Combining and Merging Datasets

- Database-Style DataFrame joins
- Merging on Index
- Concatenating Along an Axis
- Combining Data with Overlap

Reshaping and Pivoting

- Reshaping with hierarchical Indexing
- Pivoting 'long' to 'wide' format
- pivoting 'wide' to 'long' format

Hierarchical Indexing (Series)

```
import pandas as pd
import numpy as np
```

```
data = pd.Series(np.random.uniform(size = 9),
                 index =  [['a', 'a', 'b', 'c', 'c', 'b', 'c', 'b','a'],
                           [1, 2, 3, 1, 3, 4, 3, 2, 1]])
data
```

```
a  1    0.684862
   2    0.701188
b  3    0.870829
c  1    0.958994
   3    0.042434
b  4    0.539591
c  3    0.668997
b  2    0.501304
a  1    0.260682
dtype: float64
```

```
# gaps for 'multi-index'
data.index
```

```
MultiIndex([('a', 1),
            ('a', 2),
            ('b', 3),
            ('c', 1),
            ('c', 3),
            ('b', 4),
            ('c', 3),
            ('b', 2),
            ('a', 1)],
           )
```

```
mean = [0, 0]
cov =  [[1,0], [0, 100]]
```

```
data
```

```
a  1    0.684862
   2    0.701188
b  3    0.870829
c  1    0.958994
   3    0.042434
b  4    0.539591
c  3    0.668997
b  2    0.501304
a  1    0.260682
dtype: float64
```

```
# selecting subset
data['b']
```

```
3    0.870829
4    0.539591
2    0.501304
dtype: float64
```

```
# selecting the data values with loc operator
data.loc[['a','b']]
```

```
a  1    0.684862
   2    0.701188
   1    0.260682
b  3    0.870829
   4    0.539591
   2    0.501304
dtype: float64
```

```
data.loc[:, 2]
```

```
a    0.701188
b    0.501304
dtype: float64
```

Hierarchical index (DataFrame)

```
frame = pd.DataFrame(np.arange(12).reshape((4,3)),
                     index = [["a", "a", "b", "b"], [1, 2, 1, 2]],
                     columns = [['fdk', 'fzp', 'chd'],
                                ['PB', 'PB', 'CHD']])
```

```
frame
```

		fdk PB	fzp PB	chd CHD
a	1	0	1	2
	2	3	4	5
b	1	6	7	8
	2	9	10	11

```
frame.index.names = ['key1', 'key2']
```

```
frame.columns.names = ['city', 'province']
```

```
frame
```

	city province	fdk PB	fzp PB	chd CHD
key1	key2			
a	1	0	1	2
	2	3	4	5
b	1	6	7	8
	2	9	10	11

```
# to check how many levels an index has
frame.index.nlevels
```

```
2
```

```
# partial column indexing
frame['fdk']
```

	province	PB
key1	key2	
a	1	0
	2	3
b	1	6
	2	9

```
frame['fzp']
```

key1	province key2	PB
a	1	1
	2	4
b	1	7
	2	10

```
frame['chd']
```

key1	province key2	CHD
a	1	2
	2	5
b	1	8
	2	11

```
pd.MultiIndex.from_arrays([['fdk', 'fzp', 'chd'],
                           ['PB', 'PB', 'CHD'],
                            names=['city', 'capital'])
```

Reordering and Sorting levels

```
frame.swaplevel('key1', 'key2')
```

key2	city province key1	fdk PB	fzp PB	chd CHD
1	a	0	1	2
2	a	3	4	5
1	b	6	7	8
2	b	9	10	11

```
frame.sort_index(level=1)
```

key1	city province key2	fdk PB	fzp PB	chd CHD
a	1	0	1	2
b	1	6	7	8
a	2	3	4	5
b	2	9	10	11

```
frame.swaplevel(0,1).sort_index(level=0)
```

key2	city province key1	fdk PB	fzp PB	chd CHD
1	a	0	1	2
	b	6	7	8
2	a	3	4	5
	b	9	10	11

Summary Statistics by Level

```
frame.groupby(level='key2').sum()
```

city province key2	fdk PB	fzp PB	chd CHD
1	6	8	10
2	12	14	16

```
frame.groupby(level= 'province', axis = 'columns').sum()
```

key1	province key2	CHD	PB
a	1	2	1
	2	5	7
b	1	8	13
	2	11	19

Indexing with a DataFrame's columns

```
frame2 = pd.DataFrame({'a': range(7), 'b': range(7,0,-1),
                       'c': ['one', 'one', 'one', 'two', 'two',
                            'two', 'two'],
                       'd': [0, 1,2,0,1,3,2]})
```

```
frame2
```

	a	b	c	d
0	0	7	one	0
1	1	6	one	1
2	2	5	one	2
3	3	4	two	0
4	4	3	two	1
5	5	2	two	3
6	6	1	two	2

```
# set_index to create a new DataFrame

frame3 = frame2.set_index(['c', 'd'])

frame3
```

		a	b
c	d		
one	0	0	7
	1	1	6
	2	2	5
two	0	3	4

c	d	a	b
	1	4	3
	3	5	2
	2	6	1

```
# we can set it to index by doing drop= False

frame2.set_index(["c",'d'], drop= False)
```

c	d	a	b	c	d
one	0	0	7	one	0
	1	1	6	one	1
	2	2	5	one	2
two	0	3	4	two	0
	1	4	3	two	1
	3	5	2	two	3
	2	6	1	two	2

```
# reset_index brings it back to the orignal position

frame2.reset_index()
```

	index	a	b	c	d
0	0	0	7	one	0
1	1	1	6	one	1
2	2	2	5	one	2
3	3	3	4	two	0
4	4	4	3	two	1
5	5	5	2	two	3
6	6	6	1	two	2

Combining and Merging Datasets

- pandas.merge (connects rows based on one/more keys) how

- pandas.concat (stacks objects together on axis)
- combine_first (slice together overlapping data to fill missing values)
- merge function arguments

```
# DataFrame joins
df1 = pd.DataFrame({"key": ['a', 'c', 'd', 'b', 'a', 'c'],
                    'data1': pd.Series(range(6), dtype= 'Int64')})

df2 = pd.DataFrame({'key': ['a', 'b', 'c'],
                    'data2': pd.Series(range(3), dtype='Int64')})
```

```
df1
```

	key	data1
0	a	0
1	c	1
2	d	2
3	b	3
4	a	4
5	c	5

```
df2
```

	key	data2
0	a	0
1	b	1
2	c	2

```
pd.merge(df1, df2)
```

	key	data1	data2
0	a	0	0
1	a	4	0
2	c	1	2
3	c	5	2
4	b	3	1

```
# specifying the column
pd.merge(df1, df2, on= 'key')
```

	key	data1	data2
0	a	0	0
1	a	4	0
2	c	1	2
3	c	5	2
4	b	3	1

```
pd.merge(df1, df2, how= 'outer')
```

	key	data1	data2
0	a	0	0
1	a	4	0
2	c	1	2
3	c	5	2
4	d	2	<NA>
5	b	3	1

Renaming Axis Indexes

```
pd.merge(df1, df2, on= 'key', suffixes = ("_left", "_right"))
```

	key	data1	data2
0	a	0	0
1	a	4	0
2	c	1	2
3	c	5	2
4	b	3	1

Join instance

```
df1.join(df1, on= 'key')
```

```
another = pd.DataFrame([[7., 8.], [9., 10.],
                        [11., 12.], [16., 17.]],
                       index = ['a', 'c', 'e', 'f'],
                       columns= ['jandiala', 'faridkot'])
```

```
another
```

	jandiala	faridkot
a	7.0	8.0
c	9.0	10.0
e	11.0	12.0
f	16.0	17.0

```
df1.join(another, how= 'outer')
```

	key	data1	jandiala	faridkot
0	a	0	NaN	NaN
1	c	1	NaN	NaN
2	d	2	NaN	NaN
3	b	3	NaN	NaN
4	a	4	NaN	NaN
5	c	5	NaN	NaN
a	NaN	<NA>	7.0	8.0
c	NaN	<NA>	9.0	10.0
e	NaN	<NA>	11.0	12.0
f	NaN	<NA>	16.0	17.0

Concatinating along the axis

- data combination
- function agruments pandas.concat

```
arr = np.arange(12).reshape((3,4))
```

```
arr
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
np.concatenate([arr, arr])
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
np.concatenate([arr, arr], axis = 1)
```

```
array([[ 0,  1,  2,  3,  0,  1,  2,  3],
       [ 4,  5,  6,  7,  4,  5,  6,  7],
       [ 8,  9, 10, 11,  8,  9, 10, 11]])
```

```
## series with no index overlap

s1 = pd.Series([0, 1], index = ['a', 'b'], dtype = 'Int64')
s2 = pd.Series([2,3,4], index = ['c', 'd', 'e'], dtype= 'Int64')
s3 = pd.Series([5,6], index =['e', 'f'], dtype = 'Int64')
```

```
s1
```

```
a    0
b    1
dtype: Int64
```

```
s2
```

```
c    2
d    3
e    4
dtype: Int64
```

```
s3
```

```
e    5
f    6
dtype: Int64
```

```
pd.concat([s1, s2, s3])
```

```
a    0
b    1
c    2
d    3
e    4
e    5
f    6
dtype: Int64
```

```
# the result will be a DataFrame if we pass axis = 'columns'

pd.concat([s1, s2, s3], axis = 'columns')
```

	0	1	2
a	0	<NA>	<NA>
b	1	<NA>	<NA>
c	<NA>	2	<NA>
d	<NA>	3	<NA>
e	<NA>	4	5
f	<NA>	<NA>	6

```
# trying inner join()

s4 = pd.concat([s1, s3])
```

```
s4
```

```
a    0
b    1
e    5
f    6
dtype: Int64
```

```
pd.concat([s1, s4], axis = 'columns')
```

	0	1
a	0	0
b	1	1
e	<NA>	5
f	<NA>	6

```
# because of inner join, labels 'f' and 'g' disappeared

pd.concat([s1, s4], axis = 'columns', join = 'inner')
```

	0	1
a	0	0
b	1	1

```
result = pd.concat([s1, s1, s3], keys= ['one', 'two', 'three'])
```

```
result
```

```
one    a    0
       b    1
two    a    0
       b    1
three  e    5
       f    6
dtype: Int64
```

```
result.unstack()
```

	a	b	e	f
one	0	1	<NA>	<NA>
two	0	1	<NA>	<NA>
three	<NA>	<NA>	5	6

- in case of combining Series along axis= 'columns', > keys become DataFrame column headers

```
pd.concat([s1, s2, s3], axis = 'columns',
          keys = ['one', 'two', 'three'])
```

	one	two	three
a	0	<NA>	<NA>
b	1	<NA>	<NA>
c	<NA>	2	<NA>
d	<NA>	3	<NA>
e	<NA>	4	5
f	<NA>	<NA>	6

```
# same logic extends to DataFrame objects
pd.concat([df1, df2], axis = 'columns')
```

	key	data1	key	data2
0	a	0	a	0
1	c	1	b	1
2	d	2	c	2
3	b	3	NaN	<NA>
4	a	4	NaN	<NA>
5	c	5	NaN	<NA>

- In dictionary objects, the keys will be used > for **key** option

```
pd.concat({'level1': df1, 'level2': df2},
          axis = 'columns')
```

	level1		level2	
	key	data1	key	data2
0	a	0	a	0
1	c	1	b	1
2	d	2	c	2
3	b	3	NaN	<NA>
4	a	4	NaN	<NA>
5	c	5	NaN	<NA>

```
# additional arguments

pd.concat([df1, df2], axis = 'columns',
          keys = ['level1', 'level2'],
          names = ['upper', 'lower'])
```

upper	level1		level2	
lower	key	data1	key	data2
0	a	0	a	0
1	c	1	b	1
2	d	2	c	2
3	b	3	NaN	<NA>
4	a	4	NaN	<NA>
5	c	5	NaN	<NA>

```
# merging by ignoring_index in DataFrame

df3 = pd.DataFrame(np.random.standard_normal((3, 4)),
                   columns = ['a', 'b', 'c', 'd'])


df4 = pd.DataFrame(np.random.standard_normal((2,3)),
                    columns = ['g', 'd', 'a'])
```

```
df3
```

	a	b	c	d
0	-0.692867	-0.923164	-1.055435	0.938207
1	-0.060941	1.029882	-0.332099	-1.697114
2	-0.274830	1.991366	-0.540897	0.961377

```
df4
```

	g	d	a
0	-1.397642	1.511266	-0.920547
1	0.518125	-1.409185	-1.092790

```
pd.concat([df3, df4], ignore_index = True)
```

	a	b	c	d	g
0	1.711731	-0.644975	-0.093205	0.074968	NaN
1	-1.397718	-1.585621	0.808180	-0.492032	NaN
2	0.923910	0.606571	-1.045814	1.247491	NaN
3	-0.905022	NaN	NaN	-1.122829	-0.352158
4	0.091307	NaN	NaN	-0.122968	-0.349629

Combining Data with Overlap

```
a = pd.Series([np.nan, 2.5, 0.0, 4.5, 3, np.nan],
              index = ['a', 'b', 'c', 'g', 'k', 'o'])

b = pd.Series([0., np.nan, 3., np.nan, 5., 2.],
              index = ['a', 'b', 'c', 'd', 'e', 'f'])
```

```
a
```

```
a    NaN
b    2.5
c    0.0
c    4.5
a    3.0
b    NaN
dtype: float64
```

```
b
```

```
a    0.0
b    NaN
c    3.0
d    NaN
e    5.0
f    2.0
dtype: float64
```

- Explanation - > selects non-null values from **a or b**

    np.where doesnot check the index labels

    better to use **combine_first** method

    combine_first method will have the union of all column names

```
np.where(pd.isna(a), b, a)
```

```
array([0. , 2.5, 0. , 4.5, 3. , 2. ])
```

```
a.combine_first(b)
```

```
a    0.0
b    2.5
c    0.0
d    NaN
e    5.0
f    2.0
g    4.5
k    3.0
o    NaN
dtype: float64
```

```
# using combine_first on DataFrame

df1.combine_first(df2)
```

	data1	data2	key
0	0	0	a
1	1	1	c
2	2	2	d
3	3	<NA>	b
4	4	<NA>	a
5	5	<NA>	c

Reshaing and Pivoting

- **stack method** - rotates or pivots the columns
- **unstack method** - pivots the rows into columns

```
data = pd.DataFrame(np.arange(6).reshape((2,3)),
                    index = pd.Index(['fdk', 'golewala'],
                                     name = 'city'),
                     columns = pd.Index(['one','two', 'three'],
                                        name= 'number'))
```

```
data
```

number city	one	two	three
fdk	0	1	2
golewala	3	4	5

```
result_stack= data.stack()

result_stack
```

```
city      number
fdk       one       0
          two       1
          three     2
golewala  one       3
          two       4
          three     5
dtype: int32
```

```
result_stack.unstack()
```

number city	one	two	three
fdk	0	1	2
golewala	3	4	5

```
result_stack.unstack(level = 0)
```

city number	fdk	golewala
one	0	3
two	1	4
three	2	5

```
result_stack.unstack(level = 'city')
```

city number	fdk	golewala
one	0	3
two	1	4
three	2	5

```
# unstacking a DataFrame

df5 = pd.DataFrame({'left': result_stack, 'right': result_stack+ 5},
                   columns = pd.Index(['left', 'right']))
```

```
df5
```

city	number	left	right
	one	0	5
fdk	two	1	6

city	number	left	right
	three	2	7
	one	3	8
golewala	two	4	9
	three	5	10

Pivoting 'long' to 'wide' Format

```
data = pd.read_csv("E:\pythonfordatanalysis\\machine-readable-business-employment-data-sep
```

```
data.head()
```

	Series_reference	Period	Data_value	Suppressed	STATUS	UNITS	Magnitude	Subject
0	BDCQ.SEA1AA	2011.06	80078.0	NaN	F	Number	0	Business Data
1	BDCQ.SEA1AA	2011.09	78324.0	NaN	F	Number	0	Business Data
2	BDCQ.SEA1AA	2011.12	85850.0	NaN	F	Number	0	Business Data
3	BDCQ.SEA1AA	2012.03	90743.0	NaN	F	Number	0	Business Data
4	BDCQ.SEA1AA	2012.06	81780.0	NaN	F	Number	0	Business Data

```
data2 = data.loc[:, ['Period', 'Group', 'Magnitude']]
```

```
data2.head()
```

	Period	Group	Magnitude
0	2011.06	Industry by employment variable	0
1	2011.09	Industry by employment variable	0
2	2011.12	Industry by employment variable	0
3	2012.03	Industry by employment variable	0
4	2012.06	Industry by employment variable	0

```
help(pd.PeriodIndex)
```

```
divide = pd.PeriodIndex(year = [2000, 2002],
                        quarter = [1,4])
```

```
divide
```

```
PeriodIndex(['2000Q1', '2002Q4'], dtype='period[Q-DEC]')
```

```
data.columns
```

```
Index(['Series_reference', 'Period', 'Data_value', 'Suppressed', 'STATUS',
       'UNITS', 'Magnitude', 'Subject', 'Group', 'Series_title_1',
       'Series_title_2', 'Series_title_3', 'Series_title_4', 'Series_title_5'],
      dtype='object')
```

```
data.columns.name = 'item'
```

```
data.head()
```

item	Series_reference	Data_value	Suppressed	STATUS	UNITS	Magnitude	Subject
0	BDCQ.SEA1AA	80078.0	NaN	F	Number	0	Business Data Collect
1	BDCQ.SEA1AA	78324.0	NaN	F	Number	0	Business Data Collect
2	BDCQ.SEA1AA	85850.0	NaN	F	Number	0	Business Data Collect
3	BDCQ.SEA1AA	90743.0	NaN	F	Number	0	Business Data Collect
4	BDCQ.SEA1AA	81780.0	NaN	F	Number	0	Business Data Collect

```
long_data = (data.stack()
             .reset_index()
             .rename(columns = {0:'value'}))
```

```
long_data[:10]
```

	level_0	item	value
0	0	Series_reference	BDCQ.SEA1AA
1	0	Data_value	80078.0
2	0	STATUS	F
3	0	UNITS	Number
4	0	Magnitude	0
5	0	Subject	Business Data Collection - BDC
6	0	Series_title_1	Filled jobs

	level_0	item	value
7	0	Series_title_2	Agriculture, Forestry and Fishing
8	0	Series_title_3	Actual
9	1	Series_reference	BDCQ.SEA1AA

Pivoting 'wide' to 'long' Format

- pd.melt- using particular coloumn as a key indicator
- pd.pivot- used to reset_index to move data back to column

```
df6 = pd.DataFrame({'key': ['foo', 'bar', 'xyz'],
                    'A': [1, 3, 5],
                     'C': [4, 6, 3],
                     'D': [4, 64, 2]})
```

```
df6
```

	key	A	C	D
0	foo	1	4	4
1	bar	3	6	64
2	xyz	5	3	2

```
# using pd.melt to use key as group indicator
melted = pd.melt(df6, id_vars = 'key')
```

```
melted
```

	key	variable	value
0	foo	A	1
1	bar	A	3
2	xyz	A	5
3	foo	C	4
4	bar	C	6
5	xyz	C	3
6	foo	D	4
7	bar	D	64

	key	variable	value
8	xyz	D	2

```
# back to orignal
reshaped = melted.pivot(index = 'key',
                        columns = 'variable',
                        values = 'value')
```

```
reshaped
```

variable key	A	C	D
bar	3	6	64
foo	1	4	4
xyz	5	3	2

```
reshaped.reset_index()
```

variable	key	A	C	D
0	bar	3	6	64
1	foo	1	4	4
2	xyz	5	3	2

```
df6
```

	key	A	C	D
0	foo	1	4	4
1	bar	3	6	64
2	xyz	5	3	2

```
# specify a subset of columns to use as a value columns

pd.melt(df6, id_vars = "key", value_vars = ['A', 'C'])
```

	key	variable	value
0	foo	A	1
1	bar	A	3
2	xyz	A	5
3	foo	C	4
4	bar	C	6
5	xyz	C	3