PANDAS

(in depth - 1)

Data Manipulation

Can be considered of comprising 3 stages:

- ❑ Data preparation
- ❑ Data transformation
- ❑ Data aggregation

This lecture will look at Data Preparation!

Data preparation

- Merge(): connects the rows in a dataframe based on one or more keys (somewhat like Join in SQL).
- Concat(): concatenates the objects along an axis.
- combine_first(): connects overlapping data in a dataframe to enable the filling in of missing values by taking data from another structure.
- Pivot: enables the interchange of rows and columns in a dataframe.

Setting up

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

>>> frame1 = pd.DataFrame( {'id':['ball','pencil','pen','mug','ashtray’], 'price’:
                    [12.33,11.44,33.21,13.23,33.62]})
>>> frame2 = pd.DataFrame( {'id':['pencil','pencil','ball','pen’], 'color': ['white','red','red','black']})
```

```
>>> print(frame1)

        id          price
0       ball        12.33
1       pencil      11.44
2       pen         33.21
3       mug         13.23
4       ashtray     33.62
```

```
>>> print(frame2)

     color    id
0    white    pencil
1    red      pencil
2    red      ball
3    black    pen
```

Merge operation

To merge the 2 dataframes:

```
>>>> merged_frame = pd.merge(frame1,frame2)
```

the resulting DataFrame consists of all rows that have an ID in common between the two DataFrames.

```
>>> print(merged_frame)

        id        price    color
 0      ball      12.33    red
 1      pencil    11.44    white
 2      pencil    11.44    red
 3      pen       33.21    black
```

Usually you need to specify the merge column:

```
>>> pd.merge(frame1,frame2,on='id’)
>>> frame1 = pd.DataFrame(
{'id':['ball','pencil','pen','mug','ashtray'],
 'color': ['white','red','red','black','green'],
 'brand': ['OMG','ABC','ABC','POD','POD']})

>>> frame1

    brand  color    id
0   OMG   white    ball
1   ABC   red      pencil
2   ABC   red      pen
3   POD   black    mug
4   POD   green    ashtray
```

```
>>> frame2 = pd.DataFrame(
{'id':['pencil','pencil','ball','pen’],
  'brand': ['OMG','POD','ABC','POD']})

>>> frame2

    brand    id
0   OMG      pencil
1   POD      pencil
2   ABC      ball
3   POD      pen
```

```
>>> pd.merge(frame1,frame2)

Empty DataFrame
Columns: [brand, color, id]
Index: []
```

# Since both columns of frame2 are present in frame1. Ambiguity !

Use ‘on’ option to explicitly define the criterion of merging that pandas must follow

```
>>> pd.merge(frame1,frame2,on='id’)

     brand_x  color  id       brand_y
0    OMG      white  ball     ABC
1    ABC      red    pencil   OMG
2    ABC      red    pencil   POD
3    ABC      red    pen      POD
```

```
>>> pd.merge(frame1,frame2,on='brand’)

   brand  color    id_x     id_y
0  OMG    white    ball     pencil
1  ABC    red      pencil   ball
2  ABC    red      pen      ball
3  POD    black    mug      pencil
4  POD    black    mug      pen
5  POD    green    ashtray  pencil
6  POD    green    ashtray  pen
```

What if key columns in two DataFrames do not have the same name?

- use the **left_on** and **right_on** options that specify the key column for the first and for the second DataFrame.

```
>>> frame1 = pd.DataFrame(
{'id':['ball','pencil','pen','mug','ashtray'],
 'color': ['white','red','red','black','green'],
 'brand': ['OMG','ABC','ABC','POD','POD']})

>>> frame1

   brand  color    id
0  OMG    white    ball
1  ABC    red      pencil
2  ABC    red      pen
3  POD    black    mug
4  POD    green    ashtray
```

```
>>> frame2 = pd.DataFrame(
{‘sid':['pencil','pencil','ball','pen’],
 'brand': ['OMG','POD','ABC','POD']})

>>> frame2
   brand   sid
0  OMG     pencil
1  POD     pencil
2  ABC     ball
3  POD     pen
```

```
>>> pd.merge(frame1, frame2, left_on='id', right_on='sid’)

  brand_x  color   id      brand_y   sid
0   OMG    white   ball    ABC       ball
1   ABC    red     pencil  OMG       pencil
2   ABC    red     pencil  POD       pencil
3   ABC    red     pen     POD       pen
```

Here, id is the join column for the first dataframe and sid the join column for the second dataframe.

The sql equivalent would be: where frame1.id = frame2.sid

- ✓ By default, the merge( ) function performs an inner join; the keys in the result are the result of an intersection.

Merge contd..

- Other possible merge operations are the left join, the right join, and the outer join.
- The outer join produces the union of all keys, combining the effect of a left join with a right join.
- To select the type of join you have to use the “how” option.

```
>>> pd.merge(frame1,frame2,on='id',how='outer’)     # ensures all rows included from both frame even if they don’t match
```

- Driving the merge from the left:

```
>>> pd.merge(frame1,frame2,on='id',how='left’)     # all rows from frame1 and any rows from frame2 that match
```

- Or from the right:

```
>>> pd.merge(frame1,frame2,on='id',how='right’)   # all rows from frame2 and any rows from frame1 that match
```

To merge multiple keys, simply add a list to the on option:

```
>>> pd.merge(frame1,frame2,on=['id','brand'],how='outer')
```

See the tutorial for more examples !

Concatenating

- NumPy has a concatenate function for concatenating arrays:

```
>>> array1 = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
>>> array2 = np.arange(9).reshape((3,3))+6
>>> array3 = np.concatenate([array1,array2],axis=1)
```

```
print(array3)
array([[ 0,  1,  2,  6,  7,  8],
       [ 3,  4,  5,  9, 10, 11],
       [ 6,  7,  8, 12, 13, 14]])
```

Concatenating rows of arrays

```
>>> np.concatenate([array1,array2],axis=0)
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 6,  7,  8],
       [ 9, 10, 11],
       [12, 13, 14]])
```

The Pandas concat() function

```
>>> ser1 = pd.Series(np.random.rand(4), index=[1,2,3,4])
>>> ser2 = pd.Series(np.random.rand(4), index=[5,6,7,8])
```

```
print(ser1)

1    0.636584
2    0.345030
3    0.157537
4    0.070351
dtype: float64
```

```
print(ser2)

5    0.411319
6    0.359946
7    0.987651
8    0.329173
dtype: float64
```

```
>>> ser3 = pd.concat([ser1,ser2])

print(ser3)

1    0.636584
2    0.345030
3    0.157537
4    0.070351
5    0.411319
6    0.359946
7    0.987651
8    0.329173
dtype: float64
```

The Pandas concat() function..

- By default, the concat() function works on axis = 0, returning a series object.
- If you set the axis = 1, then the result will be a DataFrame.

```
>>> ser3 = pd.concat([ser1,ser2],axis=1)


print(ser3)

        0          1
1  0.636584      NaN
2  0.345030      NaN
3  0.157537      NaN
4  0.070351      NaN
5       NaN    0.411319
6       NaN    0.359946
7       NaN    0.987651
8       NaN    0.329173
```

This has performed an outer join. This can be changed by setting the join option to 'inner’:

```
>>> pd.concat([ser1,ser2],axis=1,join='inner’)

Empty dataframe
Columns: [0,1]
Index = []
```

What will be output for

```
>>> pd.concat([ser1,ser3],axis=1,join='inner’)    ??
```

The Pandas concat() function..

- To create a hierarchical index on the axis of concatenation we need to use the keys option:

```
>>> pd.concat([ser1,ser2], keys=[1,2])

1     1   0.636584
      2   0.345030
      3   0.157537
      4   0.070351
2     5   0.411319
      6   0.359946
      7   0.987651
      8   0.329173
dtype: float64
```

Concatenating dataframes

- In the case of combinations between Series along the axis = 1, the keys become the column headers of the DataFrame.
- Essentially applies the same approach as we saw with series objects:

```
>>> frame1 = pd.DataFrame(np.random.rand(9).reshape(3,3), index=[1,2,3], columns=['A','B','C'])
>>> frame2 = pd.DataFrame(np.random.rand(9).reshape(3,3), index=[4,5,6], columns=['A','B','C'])
```

```
>>> pd.concat([frame1, frame2])          # defaults to rows
       A          B          C
1      0.400663   0.937932   0.938035
2      0.202442   0.001500   0.231215
3      0.940898   0.045196   0.723390
4      0.568636   0.477043   0.913326
5      0.598378   0.315435   0.311443
6      0.619859   0.198060   0.647902
```

Concatenating dataframes ..

For columns:

```
>>> pd.concat([frame1, frame2], axis=1)

        A          B          C          A          B          C
1    0.400663   0.937932   0.938035   NaN        NaN        NaN
2    0.202442   0.001500   0.231215   NaN        NaN        NaN
3    0.940898   0.045196   0.723390   NaN        NaN        NaN
4    NaN        NaN        NaN        0.568636   0.477043   0.913326
5    NaN        NaN        NaN        0.598378   0.315435   0.311443
6    NaN        NaN        NaN        0.619859   0.198060   0.647902
```

Combine()

If we wish the two datasets to have indexes that overlap in their entirety or at least partially, we can use combine_first().

Lets define two series:

```
>>> ser1 = pd.Series(np.random.rand(5),index=[1,2,3,4,5])
>>> ser2 = pd.Series(np.random.rand(4),index=[2,4,5,6])
```

```
print(ser1)

1    0.942631
2    0.033523
3    0.886323
4    0.809757
5    0.800295
dtype: float64
```

```
print(ser2)

2    0.739982
4    0.225647
5    0.709576
6    0.214882
dtype: float64
```

```
>>> ser1.combine_first(ser2)
1    0.942631
2    0.033523
3    0.886323
4    0.809757
5    0.800295
6    0.214882
dtype: float64
```

Combine() ..

- If you want a partial overlap, you can specify only the portion of the Series you want to overlap.

```
>>> ser1[:3].combine_first(ser2[:3])

1    0.942631
2    0.033523
3    0.886323
4    0.225647
5    0.709576
dtype: float64
```

Pivoting with Hierarchical Indexing

In the context of pivoting there are two basic operations:

- ❑ Stacking: rotates or pivots the data structure converting columns to rows
- ❑ Unstacking: converts rows into columns

```
>>> frame1 = pd.DataFrame(np.arange(9).reshape(3,3), index=['white','black','red’],
                    columns=['ball','pen','pencil’])

>>>print(frame1)

          ball     pen     pencil
white     0        1       2
black     3        4       5
red       6        7       8
```

Pivoting with hierarchical indexing ..

- Using the stack() function on the DataFrame, pivots the columns into rows, thus producing a series:

```
>>> ser5 = frame1.stack()
>>> print(ser5)

white    ball      0
         pen       1
         pencil    2
black    ball      3
         pen       4
         pencil    5
red      ball      6
         pen       7
         pencil    8
dtype: int32
```

- From this hierarchically indexed series, you can reassemble the DataFrame into a pivoted table by use of the unstack() function.

```
>>> ser5.unstack()

        ball    pen    pencil
white   0       1      2
black   3       4      5
red     6       7      8
```

Pivoting with hierarchical indexing ..

- You can also do the unstack on a different level, specifying the number of levels or its name as the argument of the function.

```
>>> ser5.unstack(0)

         white  black    red
ball       0      3       6
pen        1      4       7
pencil     2      5       8
```

Removing columns and rows

Lets define a dataframe:

```
>>> frame1 = pd.DataFrame(np.arange(9).reshape(3,3), index=['white','black','red’],
                    columns=['ball','pen','pencil'])
```

```
>>> frame1
        ball    pen    pencil
white    0       1       2
black    3       4       5
red      6       7       8
```

- To remove a column, simply use the del command applied to the DataFrame with the column name specified

```
>>> del frame1['ball’]              # removes column “ball” from frame1
```

- To remove an unwanted row, you have to use the drop() function with the label of the corresponding index as argument

```
>>> frame1.drop('white’)            # removes the first row “white” from frame1
```