Join, Merge and Concatenate

When handling data, you might want to merge different datasets in specific ways. You might also want to apply different functions to some particular columns.

Joining and Merging Dataframes

Join

Joining is a more convenient method for combining the columns of two potentially differently-indexed DataFrames into a single DataFrame. Joining is similar to merging except join method occurs on the index key instead of any particular column.

There are various join logics available to merge the pandas dataframe based on a common column.

We are creating two datasets.

```
import pandas as pd
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3']},
                     index=['K0', 'K1', 'K2', 'K3'])
df2 = pd.DataFrame({'C': ['A4', 'A5', 'A6', 'A7', 'A8'],
                    'D': ['B4', 'B5', 'B6', 'B7', 'B8']},
                         index=['K0', 'K1', 'K2', 'K3', 'K4'])
print(df1, '\n')
print(df2)
```

```
      A   B
K0  A0  B0
K1  A1  B1
K2  A2  B2
K3  A3  B3

      C   D
K0  A4  B4
K1  A5  B5
K2  A6  B6
K3  A7  B7
K4  A8  B8
```

We can specify the type of join as a parameter to the join function. The join scheme may be Outer join, Inner join, Left join, Right join.

Inner Join

```
df1.join(df2, how = 'inner') # returns K0, K1, K2, K3
```

	A	B	C	D
K0	A0	B0	A4	B4
K1	A1	B1	A5	B5
K2	A2	B2	A6	B6
K3	A3	B3	A7	B7

Inner join returns all rows from both tables where the key record of one is equal to the other. It focuses on the commonality of the two tables.

Outer Join

```
df1.join(df2, how='outer') # returns the record corresponding K0, K1, K2, K3, K4
```

	A	B	C	D
K0	A0	B0	A4	B4
K1	A1	B1	A5	B5
K2	A2	B2	A6	B6
K3	A3	B3	A7	B7
K4	NaN	NaN	A8	B8

It returns a set of records that include what an inner join would return and also adds other rows for which there is no corresponding match in another table.  alt text

Right Join

```
df1.join(df2, how='right')
```

	A	B	C	D
K0	A0	B0	A4	B4
K1	A1	B1	A5	B5
K2	A2	B2	A6	B6
K3	A3	B3	A7	B7
K4	NaN	NaN	A8	B8

It returns a set of records that include all the data in the right dataframe. In case of missing values of the "on" variable in the left dataframe, it adds empty/ Nan values in the result. This is clear from the above example.

Left Join

```
df1.join(df2, how='left')
```

	A	B	C	D
K0	A0	B0	A4	B4
K1	A1	B1	A5	B5
K2	A2	B2	A6	B6
K3	A3	B3	A7	B7

It returns a set of records that include all the data in the left dataframe. In case of missing values of the "on" variable in the right dataframe, it adds empty/ Nan values in the result. This is clear from the above example.

Append Function

```
df1.append(df2)
```

```
/usr/local/lib/python3.6/dist-packages/pandas/core/frame.py:6692: FutureWarning: Sorting because non-concatenation axis is not aligned.
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  sort=sort)
```

	A	B	key
0	B0	NaN	A0
1	B1	NaN	A1
2	B2	NaN	A2
3	B3	NaN	A3
0	NaN	B4	A0
1	NaN	B5	A1
2	NaN	B6	A2
3	NaN	B7	A5
4	NaN	OO	ZZ

As you can see, `df2` is attached to the end of `df1`.

Merge

"Pandas" allows you to merge DataFrames while paying attention to the values as well. For example, you might be working with data about the students of a university. Suppose you have two dataframes, different containing information about the students. However, if both these dataframes has a column called "Name," they will both contain the same names for the same students. We can use it to merge two dataframes by paying attention to the names of the students, and merging wherever the names match.

The syntax for merging DataFrames is

```
pd.merge(left, right, how='inner', on='Key')
```

The left signifies the DataFrame that should appear on the left side; right signifies the DataFrame that should appear on the right side.

Sometimes the data is not complete. You might have information about one student and no information about this student in the other dataframe. If you select `how='inner'`, the join throws out all the entries where it cannot find the names of the students. You can picture this as the central intersection of a Venn diagram. If you choose the outer, it only keeps all the information about the students.

Lastly, on='Key' signifies the key column on which the merge occurs. This key column has to be similar across all the DataFrames before the merge function can occur. In our example, this would be the "Name."

```
df1 = pd.DataFrame({'key': ['A0', 'A1', 'A2', 'A3'],
                    'A': ['B0', 'B1', 'B2', 'B3']})
df2 = pd.DataFrame({'key': ['A0', 'A1', 'A2', 'A5', 'ZZ'],
                    'B': ['B4', 'B5', 'B6', 'B7', 'OO']})
print(df1)
print('\n')
print(df2)
```

```
  key   A
0  A0  B0
1  A1  B1
2  A2  B2
3  A3  B3


  key   B
0  A0  B4
1  A1  B5
2  A2  B6
3  A5  B7
4  ZZ  OO
```

Inner Merge

```
pd.merge(df1, df2, how = 'inner', on = 'key')
```

	key	A	B
0	A0	B0	B4
1	A1	B1	B5
2	A2	B2	B6

Notice how `df2` has a key "ZZ" that `df1` does not have. Since we merged it using `how='inner'`, pandas kept only the keys that we have on the left dataframe `df1`.

Let us see what happens when we put `how='outer'`.

Outer Merge

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

	key	A	B
0	A0	B0	B4
1	A1	B1	B5
2	A2	B2	B6
3	A3	B3	NaN
4	A5	NaN	B7
5	ZZ	NaN	OO

Pandas has preserved all the information. Notice how there are NaN values when the value is not available.

Concatenation

Concatenation glues DataFrames together. When concatenating DataFrames, you have to pay special attention to the dimensions. Concatenate function in pandas concatenates object along a particular axis. It performs all the set logic on other axis.

Creating three DataFrame

```
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                        'B': ['B0', 'B1', 'B2', 'B3'],
                        'C': ['C0', 'C1', 'C2', 'C3'],
                        'D': ['D0', 'D1', 'D2', 'D3']},
                        index=[0, 1, 2, 3])
df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                        'B': ['B4', 'B5', 'B6', 'B7'],
                        'C': ['C4', 'C5', 'C6', 'C7'],
                        'D': ['D4', 'D5', 'D6', 'D7']},
                         index=[4, 5, 6, 7])
df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
                        'B': ['B8', 'B9', 'B10', 'B11'],
                        'C': ['C8', 'C9', 'C10', 'C11'],
                        'D': ['D8', 'D9', 'D10', 'D11']},
                        index=[8, 9, 10, 11])
print(df1, '\n')
print(df2, '\n')
print(df3, '\n')
```

```
    A   B   C   D
0  A0  B0  C0  D0
1  A1  B1  C1  D1
2  A2  B2  C2  D2
3  A3  B3  C3  D3

    A   B   C   D
4  A4  B4  C4  D4
5  A5  B5  C5  D5
6  A6  B6  C6  D6
7  A7  B7  C7  D7

      A    B    C    D
```

```
8    A8   B8   C8   D8
9    A9   B9   C9   D9
10  A10  B10  C10  D10
11  A11  B11  C11  D11
```

Concatinate along the rows

We can concatenate them using pd.concat() method:

```
#passing dataframes as list
pd.concat([df1, df2, df3], axis=0)
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Since we are concatenating these dataframes along the rows, and all three dataframes have matching columns. Therefore each row aligns with their respective columns. If one dataframe had some columns, let's say "Z" that the others didn't; it would create a new column for it. Since the other dataframes do not have any column named "Z," they perceive NaN as values.

Concatenate along the columns

We can specify if we want to concatenate along the rows as well. We can do this by setting `axis=1`

```
pd.concat([df1, df2, df3], axis=1)
```

	A	B	C	D	A	B	C	D	A	B	C	D
0	A0	B0	C0	D0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	A2	B2	C2	D2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	A3	B3	C3	D3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	A4	B4	C4	D4	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	A5	B5	C5	D5	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	A6	B6	C6	D6	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN	A7	B7	C7	D7	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A8	B8	C8	D8
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A9	B9	C9	D9
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A10	B10	C10	D10
11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A11	B11	C11	D11

Since we are concatenating along the columns, Pandas look for matching rows. However, if you look at the three datasets, none of the indices match. So Pandas "glues" them together, filling in NaN values wherever it does not have data.

Map Functions ()

Pandas Map function maps values of Series according to input correspondence. `map` accepts `dict` and `Series` data structure

```
import numpy as np
import pandas as pd
x = pd.Series(['Nepal', 'India', np.nan, 'China'])
```

```
x.map({'Nepal': 'Kathmandu', 'India': 'New Delhi'})
```

```
0    Kathmandu
1    New Delhi
2          NaN
3          NaN
dtype: object
```

`map` also accepts the function.

```
x.map('The country is  {}'.format)
```

```
0    The country is  Nepal
1    The country is  India
2      The country is  nan
3    The country is  China
dtype: object
```

References:

Articles: