Pandas Concatenation

The concatenation operation in Pandas appends one DataFrame to another along an axis. It works similar to **SQL UNION ALL** operation.

We use the `concat()` method to concatenate two or more DataFrames in Pandas. For example,

```
import pandas as pd

# create dataframes
df1 = pd.DataFrame({'A': ['A0', 'A1'],
                    'B': ['B0', 'B1']},
                    index=[0, 1])

df2 = pd.DataFrame({'A': ['A2', 'A3'],
                    'B': ['B2', 'B3']},
                    index=[2, 3])

# concatenate two dataframes
result = pd.concat([df1, df2])

print(result)
```

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

In this example, we created two DataFrames (`df1` and `df2`) and stacked them vertically (along axis 0).

concat() Syntax

The syntax of the `concat()` method in Pandas is:

```python pd.concat(objs, axis=0, join='outer', ignore_index=False, keys=None, levels=None, names=None, verify_integrity=False, sort=False, copy=True)

Here,

`objs`: sequence of Series or DataFrame objects `axis` (optional): the axis to concatenate along `join` (optional): the type of join to perform `ignore_index` (optional): if `True`, it will not use the index values on the concatenation axis and will result in a default integer index `keys` (optional): used to construct hierarchical index using the passed keys as the outermost level `verify_integrity` (optional): If `True`, it checks whether the new concatenated axis contains duplicates and raises `ValueError` if duplicates are found `sort` (optional): sorts the non-concatenation axis if it is not already aligned
```

Example: concat() With Arguments

Let's see an example of arguments like `ignore_index` and `sort`.

```
import pandas as pd

# create dataframes
df1 = pd.DataFrame({'Name': ['John', 'Alice', 'Bob'],
                    'Age': [25, 30, 35],
                    'City': ['New York', 'Paris', 'London']})

df2 = pd.DataFrame({'Name': ['Emily', 'Michael', 'Sophia', 'Rita'],
                    'Age': [28, 32, 27, 22],
                    'City': ['Berlin', 'Tokyo', 'Sydney', 'Delhi']})

# concatenate dataframes while ignoring index
result_ignore_index = pd.concat([df1, df2], ignore_index = True)

# concatenate dataframes and sort the result
result_sort = pd.concat([df1, df2], sort = True)

# display the concatenated results
print('ignore_index = True\n', result_ignore_index)
print('\nsort = True\n', result_sort)
```

```
ignore_index = True
       Name  Age      City
0     John   25  New York
1    Alice   30     Paris
2      Bob   35    London
3    Emily   28    Berlin
4  Michael   32     Tokyo
5   Sophia   27    Sydney
6     Rita   22     Delhi

sort = True
    Age      City     Name
0   25  New York     John
1   30     Paris    Alice
2   35    London      Bob
0   28    Berlin    Emily
1   32     Tokyo  Michael
2   27    Sydney   Sophia
3   22     Delhi     Rita
```

In this example, we used the `ignore_index` and `sort` argument in the `concat()` method.

When `ignore_index` is set to `True`, the index values of individual DataFrames are ignored and new index values are used in the resulting DataFrame.

When `sort` is set to `True`, the non-concatenation axis (axis 0 in this case) is sorted alphabetically. Hence in the resulting DataFrame, the columns are sorted alphabetically based on their names.

Concatenation Along Axis 1

By specifying `axis=1`, we can concatenate along the columns (horizontal). For example,

```
import pandas as pd

# create dataframes
df1 = pd.DataFrame({'Name': ['John', 'Alice', 'Bob'],
                    'Age': [25, 30, 35],
                    'City': ['New York', 'Paris', 'London']})

df2 = pd.DataFrame({'Name': ['Emily', 'Michael', 'Sophia', 'Rita'],
                    'Age': [28, 32, 27, 22],
                    'City': ['Berlin', 'Tokyo', 'Sydney', 'Delhi']})

# concatenate dataframes along axis 1
result = pd.concat([df1, df2], axis=1)

print(result)
```

```
    Name   Age      City     Name  Age    City
0   John  25.0  New York    Emily   28  Berlin
1  Alice  30.0     Paris  Michael   32   Tokyo
2    Bob  35.0    London   Sophia   27  Sydney
3    NaN   NaN       NaN     Rita   22   Delhi
```

Here, we concatenated two DataFrames `df1` and `df2` along the horizontal axis.

An outer join is performed by default while concatenating DataFrames along axis 1. This means it returns a new DataFrame that contains all rows from both original DataFrames. If there is no match for a given row, the missing values are filled with `NaN`.

If we want to return a DataFrame that contains only rows that have matching values in both of the original DataFrames, we need to perform an inner join by specifying `join = 'inner'`.

Example: Inner Join Vs Outer Join

```
import pandas as pd

# create dataframes
df1 = pd.DataFrame({'Name': ['John', 'Alice', 'Bob'],
                    'Age': [25, 30, 35],
                    'City': ['New York', 'Paris', 'London']})

df2 = pd.DataFrame({'Name': ['Emily', 'Michael', 'Sophia', 'Rita'],
                    'Age': [28, 32, 27, 22],
```

```
                    'City': ['Berlin', 'Tokyo', 'Sydney', 'Delhi']})


# concatenate dataframes with outer join
result_outer = pd.concat([df1, df2], axis = 1)

# concatenate dataframes with inner join
result_inner = pd.concat([df1, df2], axis = 1, join = 'inner')

# display the concatenated results
print('Outer Join\n', result_outer)
print('\nInner Join\n', result_inner)
```

```
Outer Join
     Name   Age      City     Name  Age    City
0   John  25.0  New York    Emily   28  Berlin
1  Alice  30.0     Paris  Michael   32   Tokyo
2    Bob  35.0    London   Sophia   27  Sydney
3    NaN   NaN       NaN     Rita   22   Delhi

Inner Join
     Name  Age      City     Name  Age    City
0   John   25  New York    Emily   28  Berlin
1  Alice   30     Paris  Michael   32   Tokyo
2    Bob   35    London   Sophia   27  Sydney
```

Notice that `NaN` values are filled in empty places to include all the rows of `df2` in case of outer join.

While in case of inner join, the row without matching index is dropped altogether.

Concatenation With Keys

The `keys` parameter is particularly useful when we want to add an extra level of information to the resulting dataframe.

When we pass a list of keys to the `concat()` function, Pandas will create a new hierarchical index level. The new index level contains the information according to the origin of the data. For example,

```
import pandas as pd

# create dataframes
df1 = pd.DataFrame({'Name': ['John', 'Alice', 'Bob'],
                    'Age': [25, 30, 35],
                    'City': ['New York', 'Paris', 'London']})

df2 = pd.DataFrame({'Name': ['Emily', 'Michael', 'Sophia', 'Rita'],
                    'Age': [28, 32, 27, 22],
                    'City': ['Berlin', 'Tokyo', 'Sydney', 'Delhi']})
```

```
# concatenate dataframes while ignoring index
result = pd.concat([df1, df2], keys = ['from_df1', 'from_df2'])

print(result)
```

```
                Name  Age      City
from_df1 0      John   25  New York
         1     Alice   30     Paris
         2       Bob   35    London
from_df2 0     Emily   28    Berlin
         1   Michael   32     Tokyo
         2    Sophia   27    Sydney
         3      Rita   22     Delhi
```

In this example, we passed the list of keys `['from_df1', 'from_df2']`.

This created a two-level index in the resulting DataFrame. The first level of the index is the keys we specified `('from_df1' and 'from_df2')`, and the second level of the index is the original index from `df1` and `df2`.

This feature is particularly useful when the origin of data is important for further data analysis.