Pandas Join

The join operation in Pandas joins two DataFrames based on their indexes.

Let's see an example.

```python
import pandas as pd

# create dataframe 1
data1 = {
    'A': ['A0', 'A1', 'A2', 'A3'],
    'B': ['B0', 'B1', 'B2', 'B3'],
}
df1 = pd.DataFrame(data1, index=['K0', 'K1', 'K2', 'K3'])

# create dataframe 2
data2 = {
    'C': ['C0', 'C1', 'C2', 'C3'],
    'D': ['D0', 'D1', 'D2', 'D3'],
}
df2 = pd.DataFrame(data2, index=['K0', 'K1', 'K2', 'K3'])

# join dataframes
df_join = df1.join(df2)

# display DataFrames
print("DataFrame 1:\n", df1)
print("\nDataFrame 2:\n", df2)
print("\nJoined DataFrame:\n", df_join)
```

```
DataFrame 1:
      A   B
K0  A0  B0
K1  A1  B1
K2  A2  B2
K3  A3  B3

DataFrame 2:
      C   D
K0  C0  D0
K1  C1  D1
K2  C2  D2
K3  C3  D3

Joined DataFrame:
      A   B   C   D
K0  A0  B0  C0  D0
K1  A1  B1  C1  D1
```

```
K2  A2  B2  C2  D2
K3  A3  B3  C3  D3
```

In this example, we joined DataFrames `df1` and `df2` using `join()`.

Here, we have specified `index= ['K0', 'K1', 'K2', 'K3']` in both the DataFrames. This is to provide a common index column based on which we can perform the join operation.

join() Syntax

The syntax of the `join()` method in Pandas is: ```python df1.join(df2, on=None, how='left', lsuffix='', rsuffix='', sort=False)

Here,

- `df1`: is the first DataFrame
- `df2`: is the dataframe to be joined to the first DataFrame
- `on(optional)`: specifies the index column(s) based on which the DataFrames are joined
- `how(optional)`: specifies the type of join to perform
- `lsuffix(optional)`: specifies a suffix that will be appended to a column name of the first DataFrame if there is a collision or conflict with another column name
- `rsuffix(optional)`: specifies a suffix that will be appended to a column name of the second DataFrame if there is a collision or conflict with another column name
- `sort(optional)`: determines whether to sort the result DataFrame by the join keys

Example: Join DataFrames

As discussed above, the `join()` method can only join DataFrames based on an index. However, we can treat a column as an index by passing it to `set_index()`. We can then use the column to join DataFrames.

Let's see an example.

```python
import pandas as pd

# create dataframes from the dictionaries
data1 = {
    'EmployeeID' : ['E001', 'E002', 'E003', 'E004', 'E005'],
    'Name' : ['John Doe', 'Jane Smith', 'Peter Brown', 'Tom Johnson', 
'Rita Patel'],
    'DeptID': ['D001', 'D003', 'D001', 'D002', 'D006'],
    'DeptName': ['Sales1', 'Admin1', 'Sales1', 'HR1', 'N/A']
}
employees = pd.DataFrame(data1)

data2 = {
    'DeptID' : ['D001', 'D002', 'D003', 'D004'],
```

```
    'DeptName' : ['Sales2', 'HR2', 'Admin2', 'Marketing2']
}
departments = pd.DataFrame(data2)

# set DeptID as index for departments dataframe
departments = departments.set_index('DeptID')

# join the dataframes based on columns
df_join = employees.join(departments, on = 'DeptID', lsuffix =
'_left', rsuffix = '_right')

print(df_join)
```

```
  EmployeeID         Name DeptID DeptName_left DeptName_right
0       E001     John Doe   D001        Sales1         Sales2
1       E002   Jane Smith   D003        Admin1         Admin2
2       E003  Peter Brown   D001        Sales1         Sales2
3       E004  Tom Johnson   D002           HR1            HR2
4       E005   Rita Patel   D006           N/A            NaN
```

In the above example, we performed a join operation on two DataFrames `employees` and `departments` using the `join()` method.

Notice the line,

```python departments = departments.set_index('DeptID')

Here, we have set the column `DeptID` as the index.

Also, notice we've made `DeptID` the index for `departments` but not `employees`. This is because the column used for the join should be the index of the right DataFrame, not always the left one.

In such cases, we need to use the `on` argument.

```python df_join = employees.join(departments, on = 'DeptID', lsuffix = '_left', rsuffix = '_right')

In this line, we've used the `on`argument with `lsuffix` and `rsuffix`.

Both DataFrames have a `DeptID` column. To tell them apart, we added `_left` to the employees and `_right` to the departments on `DeptID` columns.

# Types of Join

When joining DataFrames using the `merge()` method in pandas, the default join type is a left join if not specified otherwise. You can control the type of join performed using the `how` argument. Below are the five types of joins available:

1. **Left Join** (Default)
   – Includes all rows from the left DataFrame and only matching rows from the right DataFrame.

    - Rows in the left DataFrame without corresponding matches in the right DataFrame will have `NaN` values for columns from the right DataFrame.

2. **Right Join**
    - Includes all rows from the right DataFrame and only matching rows from the left DataFrame.
    - Rows in the right DataFrame without corresponding matches in the left DataFrame will have `NaN` values for columns from the left DataFrame.
3. **Outer Join**
    - Includes all rows from both DataFrames, with `NaN` values in the places where there are no matches.
    - Useful for combining data where you want to retain all entries from both DataFrames, regardless of whether they have matching keys.
4. **Inner Join**
    - Includes only the rows that have matching keys in both DataFrames.
    - Rows with no matching keys in either DataFrame will be excluded from the result.
5. **Cross Join**
    - Performs a Cartesian product of the two DataFrames.
    - Every row from the left DataFrame is paired with every row from the right DataFrame.
    - Be cautious with this join type, as it can result in a very large DataFrame if the original DataFrames are large.

## Left Join

A left join combines two DataFrames based on a common key and returns a new DataFrame that contains all rows from the left data frame and the matched rows from the right DataFrame.

If values are not found in the right dataframe, it fills the space with `NaN`. For example,

```python
import pandas as pd

# create dataframes from the dictionaries
data1 = {
    'EmployeeID' : ['E001', 'E002', 'E003', 'E004', 'E005'],
    'Name' : ['John Doe', 'Jane Smith', 'Peter Brown', 'Tom Johnson',
'Rita Patel'],
    'DeptID': ['D001', 'D003', 'D001', 'D002', 'D005'],
}
employees = pd.DataFrame(data1)

data2 = {
    'DeptID': ['D001', 'D002', 'D003','D004'],
    'DeptName': ['Sales', 'HR', 'Admin', 'Marketing']
}
departments = pd.DataFrame(data2)
```

```
# set DeptID as index for departments
departments.set_index('DeptID',inplace=True)

# left join
df_join = employees.join(departments, on = 'DeptID', how = 'left')

print(df_join)
```

```
  EmployeeID         Name DeptID DeptName
0       E001     John Doe   D001    Sales
1       E002   Jane Smith   D003    Admin
2       E003  Peter Brown   D001    Sales
3       E004  Tom Johnson   D002       HR
4       E005   Rita Patel   D005      NaN
```

Right Join

A right join is the opposite of a left join. It returns a new data frame that contains all rows from the right data frame and the matched rows from the left data frame.

If values are not found in the left dataframe, it fills the space with `NaN`. For example,

```
import pandas as pd

# create dataframes from the dictionaries
data1 = {
    'EmployeeID' : ['E001', 'E002', 'E003', 'E004', 'E005'],
    'Name' : ['John Doe', 'Jane Smith', 'Peter Brown', 'Tom Johnson', 
'Rita Patel'],
    'DeptID': ['D001', 'D003', 'D001', 'D002', 'D005'],
}
employees = pd.DataFrame(data1)

data2 = {
    'DeptID': ['D001', 'D002', 'D003','D004'],
    'DeptName': ['Sales', 'HR', 'Admin', 'Marketing']
}
departments = pd.DataFrame(data2)

# set DeptID as index for departments
departments.set_index('DeptID', inplace=True)

# right join
df_join = employees.join(departments, on = 'DeptID', how = 'right')

# reset index
df_join.reset_index(drop=True, inplace=True)

print(df_join)
```

```
  EmployeeID         Name DeptID   DeptName
0       E001     John Doe   D001      Sales
1       E003  Peter Brown   D001      Sales
2       E004  Tom Johnson   D002         HR
3       E002   Jane Smith   D003      Admin
4        NaN          NaN   D004  Marketing
```

Inner Join

An inner join combines two data frames based on a common key and returns a new data frame that contains only rows that have matching values in both of the original data frames.

For example,

```
import pandas as pd

# create dataframes from the dictionaries
data1 = {
    'EmployeeID' : ['E001', 'E002', 'E003', 'E004', 'E005'],
    'Name' : ['John Doe', 'Jane Smith', 'Peter Brown', 'Tom Johnson',
'Rita Patel'],
    'DeptID': ['D001', 'D003', 'D001', 'D002', 'D005'],
}
employees = pd.DataFrame(data1)

data2 = {
    'DeptID': ['D001', 'D002', 'D003','D004'],
    'DeptName': ['Sales', 'HR', 'Admin', 'Marketing']
}
departments = pd.DataFrame(data2)

# set DeptID as index for departments
departments.set_index('DeptID',inplace=True)

# inner join
df_join = employees.join(departments, on = 'DeptID', how = 'inner')

# reset index
df_join.reset_index(drop=True, inplace=True)

print(df_join)
```

```
  EmployeeID         Name DeptID DeptName
0       E001     John Doe   D001    Sales
1       E003  Peter Brown   D001    Sales
2       E002   Jane Smith   D003    Admin
3       E004  Tom Johnson   D002       HR
```

Outer Join

An outer join combines two data frames based on a common key. Unlike an inner join, an outer join returns a new data frame that contains all rows from both original data frames.

If values are not found in the DataFrames, it fills the space with `NaN`. For example,

```
import pandas as pd

# create dataframes from the dictionaries
data1 = {
    'EmployeeID' : ['E001', 'E002', 'E003', 'E004', 'E005'],
    'Name' : ['John Doe', 'Jane Smith', 'Peter Brown', 'Tom Johnson',
'Rita Patel'],
    'DeptID': ['D001', 'D003', 'D001', 'D002', 'D005'],
}
employees = pd.DataFrame(data1)

data2 = {
    'DeptID': ['D001', 'D002', 'D003','D004'],
    'DeptName': ['Sales', 'HR', 'Admin', 'Marketing']
}
departments = pd.DataFrame(data2)

# set DeptID as index for departments
departments.set_index('DeptID',inplace=True)

# outer join
df_join = employees.join(departments, on = 'DeptID', how = 'outer')

# reset index
df_join.reset_index(drop=True, inplace=True)

print(df_join)
```

```
  EmployeeID         Name DeptID   DeptName
0       E001     John Doe   D001      Sales
1       E003  Peter Brown   D001      Sales
2       E002   Jane Smith   D003      Admin
3       E004  Tom Johnson   D002         HR
4       E005   Rita Patel   D005        NaN
5        NaN          NaN   D004  Marketing
```

Cross Join

A cross join in Pandas creates the cartesian product of both DataFrames while preserving the order of the left DataFrame.

For example,

```
import pandas as pd

# create dataframes from the dictionaries
data1 = {
    'EmployeeID' : ['E001', 'E002', 'E003', 'E004', 'E005'],
    'Name' : ['John Doe', 'Jane Smith', 'Peter Brown', 'Tom Johnson',
'Rita Patel'],
    'DeptID': ['D001', 'D003', 'D001', 'D002', 'D005'],
}
employees = pd.DataFrame(data1)

data2 = {
    'DeptID': ['D001', 'D002', 'D003','D004'],
    'DeptName': ['Sales', 'HR', 'Admin', 'Marketing']
}
departments = pd.DataFrame(data2)

# set DeptID as index for departments
departments.set_index('DeptID',inplace=True)

# cross join
df_join = employees.join(departments, how = 'cross')

print(df_join)
```

```
   EmployeeID         Name DeptID   DeptName
0        E001     John Doe   D001      Sales
1        E001     John Doe   D001         HR
2        E001     John Doe   D001      Admin
3        E001     John Doe   D001  Marketing
4        E002   Jane Smith   D003      Sales
5        E002   Jane Smith   D003         HR
6        E002   Jane Smith   D003      Admin
7        E002   Jane Smith   D003  Marketing
8        E003  Peter Brown   D001      Sales
9        E003  Peter Brown   D001         HR
10       E003  Peter Brown   D001      Admin
11       E003  Peter Brown   D001  Marketing
12       E004  Tom Johnson   D002      Sales
13       E004  Tom Johnson   D002         HR
14       E004  Tom Johnson   D002      Admin
15       E004  Tom Johnson   D002  Marketing
16       E005   Rita Patel   D005      Sales
17       E005   Rita Patel   D005         HR
18       E005   Rita Patel   D005      Admin
19       E005   Rita Patel   D005  Marketing
```

Join vs Merge vs Concat

There are three different methods to combine DataFrames in Pandas:

`join()`: joins two DataFrames based on their indexes, performs left join by default `merge()`: joins two DataFrames based on any specified columns, performs inner join by default `concat()`: stacks two DataFrames along the vertical or horizontal axis