Joining Data Frames

Print to PDF

Let us understand how to join Data Frames using Pandas.

```
%run 06_csv_to_pandas_data_frame.ipynb
```

```
orders
```

	order_id	order_date	order_customer_id	order_status
0	1	2013-07-25 00:00:00.0	11599	CLOSED
1	2	2013-07-25 00:00:00.0	256	PENDING_PAYMENT
2	3	2013-07-25 00:00:00.0	12111	COMPLETE
3	4	2013-07-25 00:00:00.0	8827	CLOSED
4	5	2013-07-25 00:00:00.0	11318	COMPLETE
...	...	...	...	...
68878	68879	2014-07-09 00:00:00.0	778	COMPLETE
68879	68880	2014-07-13 00:00:00.0	1117	COMPLETE
68880	68881	2014-07-19 00:00:00.0	2518	PENDING_PAYMENT
68881	68882	2014-07-22 00:00:00.0	10000	ON_HOLD
68882	68883	2014-07-23 00:00:00.0	5533	COMPLETE

68883 rows × 4 columns

```
order_items
```

	order_item_id	order_item_order_id	order_item_product_id	order_item_quantity	order
0	1	1	957	1	
1	2	2	1073	1	
2	3	2	502	5	
3	4	2	403	1	
4	5	4	897	2	
...	...	...	...	...	
172193	172194	68881	403	1	
172194	172195	68882	365	1	
172195	172196	68882	502	1	
172196	172197	68883	208	1	
172197	172198	68883	502	3	

172198 rows × 6 columns

- Join orders and order_items using orders.order_id and order_items.order_item_order_id.

```
orders.join?
```

```
Signature: orders.join(other, on=None, how='left', lsuffix='', rsuffix='', sort=False) ->
'DataFrame'
Docstring:
Join columns of another DataFrame.

Join columns with `other` DataFrame either on index or on a key
column. Efficiently join multiple DataFrame objects by index at once by
passing a list.

Parameters
----------
other : DataFrame, Series, or list of DataFrame
    Index should be similar to one of the columns in this one. If a
    Series is passed, its name attribute must be set, and that will be
    used as the column name in the resulting joined DataFrame.
on : str, list of str, or array-like, optional
    Column or index level name(s) in the caller to join on the index
    in `other`, otherwise joins index-on-index. If multiple
    values given, the `other` DataFrame must have a MultiIndex. Can
    pass an array as the join key if it is not already contained in
    the calling DataFrame. Like an Excel VLOOKUP operation.
how : {'left', 'right', 'outer', 'inner'}, default 'left'
    How to handle the operation of the two objects.

    * left: use calling frame's index (or column if on is specified)
    * right: use `other`'s index.
    * outer: form union of calling frame's index (or column if on is
      specified) with `other`'s index, and sort it.
      lexicographically.
    * inner: form intersection of calling frame's index (or column if
      on is specified) with `other`'s index, preserving the order
      of the calling's one.
lsuffix : str, default ''
    Suffix to use from left frame's overlapping columns.
rsuffix : str, default ''
    Suffix to use from right frame's overlapping columns.
sort : bool, default False
    Order result DataFrame lexicographically by the join key. If False,
    the order of the join key depends on the join type (how keyword).

Returns
-------
DataFrame
    A dataframe containing columns from both the caller and `other`.

See Also
--------
DataFrame.merge : For column(s)-on-columns(s) operations.

Notes
-----
Parameters `on`, `lsuffix`, and `rsuffix` are not supported when
passing a list of `DataFrame` objects.

Support for specifying index levels as the `on` parameter was added
in version 0.23.0.

Examples
--------
>>> df = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3', 'K4', 'K5'],
...                    'A': ['A0', 'A1', 'A2', 'A3', 'A4', 'A5']})

>>> df
  key   A
0  K0  A0
1  K1  A1
2  K2  A2
3  K3  A3
4  K4  A4
5  K5  A5

>>> other = pd.DataFrame({'key': ['K0', 'K1', 'K2'],
...                       'B': ['B0', 'B1', 'B2']})

>>> other
  key   B
0  K0  B0
1  K1  B1
2  K2  B2

Join DataFrames using their indexes.

>>> df.join(other, lsuffix='_caller', rsuffix='_other')
  key_caller   A key_other    B
0         K0  A0        K0   B0
1         K1  A1        K1   B1
2         K2  A2        K2   B2
3         K3  A3       NaN  NaN
```

```
4         K4  A4       NaN  NaN
5         K5  A5       NaN  NaN

If we want to join using the key columns, we need to set key to be
the index in both `df` and `other`. The joined DataFrame will have
key as its index.

>>> df.set_index('key').join(other.set_index('key'))
      A    B
key
K0   A0   B0
K1   A1   B1
K2   A2   B2
K3   A3  NaN
K4   A4  NaN
K5   A5  NaN

Another option to join using the key columns is to use the `on`
parameter. DataFrame.join always uses `other`'s index but we can use
any column in `df`. This method preserves the original DataFrame's
index in the result.

>>> df.join(other.set_index('key'), on='key')
  key   A    B
0  K0  A0   B0
1  K1  A1   B1
2  K2  A2   B2
3  K3  A3  NaN
4  K4  A4  NaN
5  K5  A5  NaN
File:      /opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/pandas/core/frame.py
Type:      method
```

```python
orders.set_index('order_id')
```

order_id	order_date	order_customer_id	order_status
1	2013-07-25 00:00:00.0	11599	CLOSED
2	2013-07-25 00:00:00.0	256	PENDING_PAYMENT
3	2013-07-25 00:00:00.0	12111	COMPLETE
4	2013-07-25 00:00:00.0	8827	CLOSED
5	2013-07-25 00:00:00.0	11318	COMPLETE
...	...	...	...
68879	2014-07-09 00:00:00.0	778	COMPLETE
68880	2014-07-13 00:00:00.0	1117	COMPLETE
68881	2014-07-19 00:00:00.0	2518	PENDING_PAYMENT
68882	2014-07-22 00:00:00.0	10000	ON_HOLD
68883	2014-07-23 00:00:00.0	5533	COMPLETE

68883 rows × 3 columns

```python
order_items.set_index('order_item_order_id')
```

	order_item_id	order_item_product_id	order_item_quantity	order_item_sub
order_item_order_id				
1	1	957	1	29
2	2	1073	1	19
2	3	502	5	25
2	4	403	1	12
4	5	897	2	4
...	...	...	...	
68881	172194	403	1	12
68882	172195	365	1	5
68882	172196	502	1	5
68883	172197	208	1	199
68883	172198	502	3	15

172198 rows × 5 columns

```
# Join orders and order_items using order_id (order_item_order_id from order_items)
orders.set_index('order_id'). \
    join(order_items.set_index('order_item_order_id'))
```

	order_date	order_customer_id	order_status	order_item_id	order_item_produc
1	2013-07-25 00:00:00.0	11599	CLOSED	1.0	95
2	2013-07-25 00:00:00.0	256	PENDING_PAYMENT	2.0	107
2	2013-07-25 00:00:00.0	256	PENDING_PAYMENT	3.0	50
2	2013-07-25 00:00:00.0	256	PENDING_PAYMENT	4.0	40
3	2013-07-25 00:00:00.0	12111	COMPLETE	NaN	N
...	...	...	...	...	
68881	2014-07-19 00:00:00.0	2518	PENDING_PAYMENT	172194.0	40
68882	2014-07-22 00:00:00.0	10000	ON_HOLD	172195.0	36
68882	2014-07-22 00:00:00.0	10000	ON_HOLD	172196.0	50
68883	2014-07-23 00:00:00.0	5533	COMPLETE	172197.0	20
68883	2014-07-23 00:00:00.0	5533	COMPLETE	172198.0	50

183650 rows × 8 columns

```
orders.set_index('order_id'). \
    join(order_items.set_index('order_item_order_id'), how='inner')
```

	order_date	order_customer_id	order_status	order_item_id	order_item_produc
1	2013-07-25 00:00:00.0	11599	CLOSED	1	
2	2013-07-25 00:00:00.0	256	PENDING_PAYMENT	2	1
2	2013-07-25 00:00:00.0	256	PENDING_PAYMENT	3	
2	2013-07-25 00:00:00.0	256	PENDING_PAYMENT	4	
4	2013-07-25 00:00:00.0	8827	CLOSED	5	
...	...	...	...	...	
68881	2014-07-19 00:00:00.0	2518	PENDING_PAYMENT	172194	
68882	2014-07-22 00:00:00.0	10000	ON_HOLD	172195	
68882	2014-07-22 00:00:00.0	10000	ON_HOLD	172196	
68883	2014-07-23 00:00:00.0	5533	COMPLETE	172197	
68883	2014-07-23 00:00:00.0	5533	COMPLETE	172198	

172198 rows × 8 columns

```
orders.set_index('order_id'). \
    join(order_items.set_index('order_item_order_id'), how='inner'). \
    reset_index()
```

	index	order_date	order_customer_id	order_status	order_item_id	order_ite
0	1	2013-07-25 00:00:00.0	11599	CLOSED	1	
1	2	2013-07-25 00:00:00.0	256	PENDING_PAYMENT	2	
2	2	2013-07-25 00:00:00.0	256	PENDING_PAYMENT	3	
3	2	2013-07-25 00:00:00.0	256	PENDING_PAYMENT	4	
4	4	2013-07-25 00:00:00.0	8827	CLOSED	5	
...	...	...	...	...	...	
172193	68881	2014-07-19 00:00:00.0	2518	PENDING_PAYMENT	172194	
172194	68882	2014-07-22 00:00:00.0	10000	ON_HOLD	172195	
172195	68882	2014-07-22 00:00:00.0	10000	ON_HOLD	172196	
172196	68883	2014-07-23 00:00:00.0	5533	COMPLETE	172197	
172197	68883	2014-07-23 00:00:00.0	5533	COMPLETE	172198	

172198 rows × 9 columns

```
orders.set_index('order_id'). \
    join(order_items.set_index('order_item_order_id'), how='inner'). \
    reset_index(). \
    rename(columns={'index': 'order_id'})
```

	order_id	order_date	order_customer_id	order_status	order_item_id	order_i
0	1	2013-07-25 00:00:00.0	11599	CLOSED	1	
1	2	2013-07-25 00:00:00.0	256	PENDING_PAYMENT	2	
2	2	2013-07-25 00:00:00.0	256	PENDING_PAYMENT	3	
3	2	2013-07-25 00:00:00.0	256	PENDING_PAYMENT	4	
4	4	2013-07-25 00:00:00.0	8827	CLOSED	5	
...	...	...	...	...	...	
172193	68881	2014-07-19 00:00:00.0	2518	PENDING_PAYMENT	172194	
172194	68882	2014-07-22 00:00:00.0	10000	ON_HOLD	172195	
172195	68882	2014-07-22 00:00:00.0	10000	ON_HOLD	172196	
172196	68883	2014-07-23 00:00:00.0	5533	COMPLETE	172197	
172197	68883	2014-07-23 00:00:00.0	5533	COMPLETE	172198	

172198 rows × 9 columns

Task 1

Compute Daily Revenue using orders.order_date and order_items.order_item_order_subtotal considering only COMPLETE and CLOSED orders.

- Here are the steps to join orders and order_items and get daily revenue.
    - Create Data Frames for both orders and order_items using data in files.
    - Filter for orders which are either in **COMPLETE** or **CLOSED** status.
    - Set the join index for both the Data Frames.
    - Join both the Data Frames using `inner`.
    - Group the join results using **order_date** and get daily revenue by using `sum` on top of **order_item_subtotal**.

```
orders_considered = orders.query("order_status in ('COMPLETE', 'CLOSED')")
```

```
orders_filtered = orders[orders.order_status.isin(["COMPLETE", "CLOSED"])]
```

```
orders_considered. \
    set_index('order_id'). \
    join(order_items.set_index('order_item_order_id'), how='inner'). \
    groupby('order_date')['order_item_subtotal']. \
    agg(['sum']). \
    rename(columns={'sum': 'revenue'})
```

	revenue
order_date	
2013-07-25 00:00:00.0	31547.23
2013-07-26 00:00:00.0	54713.23
2013-07-27 00:00:00.0	48411.48
2013-07-28 00:00:00.0	35672.03
2013-07-29 00:00:00.0	54579.70
...	...
2014-07-20 00:00:00.0	60047.45
2014-07-21 00:00:00.0	51427.70
2014-07-22 00:00:00.0	36717.24
2014-07-23 00:00:00.0	38795.23
2014-07-24 00:00:00.0	50885.19

364 rows × 1 columns

Task 2

Get all the orders for which there are no corresponding order items.

- We can use default join (`left`) to get orders with out corresponding order items.

```
orders.set_index('order_id'). \
    join(order_items.set_index('order_item_order_id'))
```

	order_date	order_customer_id	order_status	order_item_id	order_item_produc
1	2013-07-25 00:00:00.0	11599	CLOSED	1.0	95
2	2013-07-25 00:00:00.0	256	PENDING_PAYMENT	2.0	107
2	2013-07-25 00:00:00.0	256	PENDING_PAYMENT	3.0	50
2	2013-07-25 00:00:00.0	256	PENDING_PAYMENT	4.0	40
3	2013-07-25 00:00:00.0	12111	COMPLETE	NaN	N
...	...	...	...	...	
68881	2014-07-19 00:00:00.0	2518	PENDING_PAYMENT	172194.0	40
68882	2014-07-22 00:00:00.0	10000	ON_HOLD	172195.0	36
68882	2014-07-22 00:00:00.0	10000	ON_HOLD	172196.0	50
68883	2014-07-23 00:00:00.0	5533	COMPLETE	172197.0	20
68883	2014-07-23 00:00:00.0	5533	COMPLETE	172198.0	50

183650 rows × 8 columns

```
orders.set_index('order_id'). \
    join(order_items.set_index('order_item_order_id')). \
    query('order_item_id.isna()')
```

	order_date	order_customer_id	order_status	order_item_id	order_item_product_id	orc
3	2013-07-25 00:00:00.0	12111	COMPLETE	NaN	NaN	
6	2013-07-25 00:00:00.0	7130	COMPLETE	NaN	NaN	
22	2013-07-25 00:00:00.0	333	COMPLETE	NaN	NaN	
26	2013-07-25 00:00:00.0	7562	COMPLETE	NaN	NaN	
32	2013-07-25 00:00:00.0	3960	COMPLETE	NaN	NaN	
...	...	...	...	...	...	
68867	2014-06-23 00:00:00.0	869	CANCELED	NaN	NaN	
68872	2014-06-29 00:00:00.0	3354	COMPLETE	NaN	NaN	
68874	2014-07-03 00:00:00.0	1601	COMPLETE	NaN	NaN	
68876	2014-07-06 00:00:00.0	4124	COMPLETE	NaN	NaN	
68877	2014-07-07 00:00:00.0	9692	ON_HOLD	NaN	NaN	

11452 rows × 8 columns

```
orders_joined = orders.set_index('order_id'). \
    join(order_items.set_index('order_item_order_id'))
```

```
orders_joined[orders_joined['order_item_id'].isna()]
```

	order_date	order_customer_id	order_status	order_item_id	order_item_product_id	orc
3	2013-07-25 00:00:00.0	12111	COMPLETE	NaN	NaN	
6	2013-07-25 00:00:00.0	7130	COMPLETE	NaN	NaN	
22	2013-07-25 00:00:00.0	333	COMPLETE	NaN	NaN	
26	2013-07-25 00:00:00.0	7562	COMPLETE	NaN	NaN	
32	2013-07-25 00:00:00.0	3960	COMPLETE	NaN	NaN	
...	...	...	...	...	...	
68867	2014-06-23 00:00:00.0	869	CANCELED	NaN	NaN	
68872	2014-06-29 00:00:00.0	3354	COMPLETE	NaN	NaN	
68874	2014-07-03 00:00:00.0	1601	COMPLETE	NaN	NaN	
68876	2014-07-06 00:00:00.0	4124	COMPLETE	NaN	NaN	
68877	2014-07-07 00:00:00.0	9692	ON_HOLD	NaN	NaN	

11452 rows × 8 columns

Task 3

Compute Daily Product Revenue using orders.order_date as well as order_items.order_item_product_id and order_items.order_item_order_subtotal considering only COMPLETE and CLOSED orders.

```
orders_considered = orders.query("order_status in ('COMPLETE', 'CLOSED')")
```

```
orders_filtered = orders[orders.order_status.isin(["COMPLETE", "CLOSED"])]
```

```
orders_considered. \
    set_index('order_id'). \
    join(order_items.set_index('order_item_order_id'), how='inner')
```

	order_date	order_customer_id	order_status	order_item_id	order_item_product_id	orc
1	2013-07-25 00:00:00.0	11599	CLOSED	1	957	
4	2013-07-25 00:00:00.0	8827	CLOSED	5	897	
4	2013-07-25 00:00:00.0	8827	CLOSED	6	365	
4	2013-07-25 00:00:00.0	8827	CLOSED	7	502	
4	2013-07-25 00:00:00.0	8827	CLOSED	8	1014	
...	...	...	...	...	...	
68880	2014-07-13 00:00:00.0	1117	COMPLETE	172191	1073	
68880	2014-07-13 00:00:00.0	1117	COMPLETE	172192	1014	
68880	2014-07-13 00:00:00.0	1117	COMPLETE	172193	1014	
68883	2014-07-23 00:00:00.0	5533	COMPLETE	172197	208	
68883	2014-07-23 00:00:00.0	5533	COMPLETE	172198	502	

75408 rows × 8 columns

```
orders_considered. \
    set_index('order_id'). \
    join(order_items.set_index('order_item_order_id'), how='inner'). \
    groupby(['order_date', 'order_item_product_id'])['order_item_subtotal']
```

```
<pandas.core.groupby.generic.SeriesGroupBy object at 0x7f6dc89feeb8>
```

```
list(orders_considered. \
    set_index('order_id'). \
    join(order_items.set_index('order_item_order_id'), how='inner'). \
    groupby(['order_date', 'order_item_product_id'])['order_item_subtotal'])[:10]
```

```
[(('2013-07-25 00:00:00.0', 24),
  57762    319.96
  Name: order_item_subtotal, dtype: float64),
 (('2013-07-25 00:00:00.0', 93),
  17    74.97
  Name: order_item_subtotal, dtype: float64),
 (('2013-07-25 00:00:00.0', 134),
  12    100.0
  Name: order_item_subtotal, dtype: float64),
 (('2013-07-25 00:00:00.0', 191),
  12       499.95
  28       399.96
  28        99.99
  61       399.96
  71       499.95
  101       99.99
  57757    499.95
  57764    499.95
  57768    499.95
  57776     99.99
  57776     99.99
  57779    499.95
  57782    199.98
  57788    199.98
  57788    499.95
  Name: order_item_subtotal, dtype: float64),
 (('2013-07-25 00:00:00.0', 226),
  68691    599.99
  Name: order_item_subtotal, dtype: float64),
 (('2013-07-25 00:00:00.0', 365),
  4        299.95
  5        299.95
  15       179.97
  17       239.96
  18       119.98
  28        59.99
  37        59.99
  45        59.99
  57       179.97
  57       119.98
  61       119.98
  61       119.98
  71       119.98
  91       299.95
  57756     59.99
  57757    119.98
  57779    299.95
  57781    299.95
  57788    239.96
  67416     59.99
  Name: order_item_subtotal, dtype: float64),
 (('2013-07-25 00:00:00.0', 403),
  5        129.99
  18       129.99
  24       129.99
  35       129.99
  57       129.99
  88       129.99
  98       129.99
  57754    129.99
  57756    129.99
  57757    129.99
  57762    129.99
  57762    129.99
  57768    129.99
  57788    129.99
  68691    129.99
  Name: order_item_subtotal, dtype: float64),
 (('2013-07-25 00:00:00.0', 502),
  4        150.0
  12       250.0
  15        50.0
  24        50.0
  24       250.0
  51        50.0
  62        50.0
  67       150.0
  98       100.0
  57757    150.0
  57758     50.0
  57758    100.0
  57764    150.0
  67416    100.0
  Name: order_item_subtotal, dtype: float64),
 (('2013-07-25 00:00:00.0', 572),
  72    119.97
  Name: order_item_subtotal, dtype: float64),
 (('2013-07-25 00:00:00.0', 625),
 57764    199.99
 Name: order_item_subtotal, dtype: float64)]
```

```python
orders_considered. \
    set_index('order_id'). \
    join(order_items.set_index('order_item_order_id'), how='inner'). \
    groupby(['order_date', 'order_item_product_id'])['order_item_subtotal']. \
    agg(['sum']). \
    rename(columns={'sum': 'revenue'})
```

		revenue
order_date	**order_item_product_id**	
2013-07-25 00:00:00.0	**24**	319.96
	93	74.97
	134	100.00
	191	5099.49
	226	599.99
...	**...**	...
2014-07-24 00:00:00.0	**926**	31.98
	957	5399.64
	1004	10399.48
	1014	3148.74
	1073	4199.79

9120 rows × 1 columns

```python
orders_considered. \
    set_index('order_id'). \
    join(order_items.set_index('order_item_order_id'), how='inner'). \
    groupby(['order_date', 'order_item_product_id'])['order_item_subtotal']. \
    agg(['sum']). \
    rename(columns={'sum': 'revenue'}). \
    reset_index()
```

	order_date	order_item_product_id	revenue
0	2013-07-25 00:00:00.0	24	319.96
1	2013-07-25 00:00:00.0	93	74.97
2	2013-07-25 00:00:00.0	134	100.00
3	2013-07-25 00:00:00.0	191	5099.49
4	2013-07-25 00:00:00.0	226	599.99
...	...	...	...
9115	2014-07-24 00:00:00.0	926	31.98
9116	2014-07-24 00:00:00.0	957	5399.64
9117	2014-07-24 00:00:00.0	1004	10399.48
9118	2014-07-24 00:00:00.0	1014	3148.74
9119	2014-07-24 00:00:00.0	1073	4199.79

9120 rows × 3 columns