[(http://www.pieriandata.com)](http://www.pieriandata.com)

Combining DataFrames

Full Official Guide (Lots of examples!)

[https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html (https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html)](https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html)

In [213]:
```
import numpy as np
import pandas as pd
```

Concatenation

Directly "glue" together dataframes.

In [214]:
```
data_one = {'A': ['A0', 'A1', 'A2', 'A3'],'B': ['B0', 'B1', 'B2', 'B3']}
```

In [215]:
```
data_two = {'C': ['C0', 'C1', 'C2', 'C3'], 'D': ['D0', 'D1', 'D2', 'D3']}
```

In [216]:
```
one = pd.DataFrame(data_one)
```

In [217]:
```
two = pd.DataFrame(data_two)
```

In [218]:
```
one
```

Out[218]:

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

In [219]:
```
two
```

Out[219]:

	C	D
0	C0	D0
1	C1	D1
2	C2	D2
3	C3	D3

Axis = 0

Concatenate along rows

In [220]:
```
axis0 = pd.concat([one,two],axis=0)
```

In [221]:
```
axis0
```

Out[221]:

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

Axis = 1

Concatenate along columns

In [222]:
```
axis1 = pd.concat([one,two],axis=1)
```

In [223]:
```
axis1
```

Out[223]:

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

Axis 0 , but columns match up

In case you wanted this:

In [224]:
```
two.columns = one.columns
```

In [225]:
```
pd.concat([one,two])
```

Out[225]:

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

Merge

Data Tables

In [226]:
```
registrations = pd.DataFrame({'reg_id':[1,2,3,4],'name':['Andrew','Bobo','C
logins = pd.DataFrame({'log_id':[1,2,3,4],'name':['Xavier','Andrew','Yoland
```

In [227]:
```
registrations
```

Out[227]:

	reg_id	name
0	1	Andrew
1	2	Bobo
2	3	Claire
3	4	David

In [228]:
```
logins
```

Out[228]:

	log_id	name
0	1	Xavier
1	2	Andrew
2	3	Yolanda
3	4	Bobo

pd.merge()

Merge pandas DataFrames based on key columns, similar to a SQL join. Results based on the **how** parameter.

In [229]:
```
help(pd.merge)
```

```
Help on function merge in module pandas.core.reshape.merge:

merge(left, right, how: str = 'inner', on=None, left_on=None, right_on=
None, left_index: bool = False, right_index: bool = False, sort: bool =
False, suffixes=('_x', '_y'), copy: bool = True, indicator: bool = Fals
e, validate=None) -> 'DataFrame'
    Merge DataFrame or named Series objects with a database-style join.

    The join is done on columns or indexes. If joining columns on
    columns, the DataFrame indexes *will be ignored*. Otherwise if join
ing indexes
    on indexes or indexes on a column or columns, the index will be pas
sed on.

    Parameters
    ----------
    left : DataFrame
    right : DataFrame or named Series
        Object to merge with.
    how : {'left', 'right', 'outer', 'inner'}, default 'inner'
```

Inner,Left, Right, and Outer Joins

Inner Join

Match up where the key is present in BOTH tables. There should be no NaNs due to the join, since by definition to be part of the Inner Join they need info in both tables. Only Andrew and Bobo both registered and logged in.

In [230]:
```
# Notice pd.merge doesn't take in a list like concat
pd.merge(registrations,logins,how='inner',on='name')
```

Out[230]:

	reg_id	name	log_id
0	1	Andrew	2
1	2	Bobo	4

In [231]:
```
# Pandas smart enough to figure out key column (on parameter) if only one c
pd.merge(registrations,logins,how='inner')
```

Out[231]:

	reg_id	name	log_id
0	1	Andrew	2
1	2	Bobo	4

In [232]:
```
# Pandas reports an error if "on" key column isn't in both dataframes
# pd.merge(registrations,logins,how='inner',on='reg_id')
```

Left Join

Match up AND include all rows from Left Table. Show everyone who registered on Left Table, if they don't have login info, then fill with NaN.

In [233]:
```
pd.merge(registrations,logins,how='left')
```

Out[233]:

	reg_id	name	log_id
0	1	Andrew	2.0
1	2	Bobo	4.0
2	3	Claire	NaN
3	4	David	NaN

Right Join

Match up AND include all rows from Right Table. Show everyone who logged in on the Right Table, if they don't have registration info, then fill with NaN.

In [234]:
```
pd.merge(registrations,logins,how='right')
```

Out[234]:

	reg_id	name	log_id
0	1.0	Andrew	2
1	2.0	Bobo	4
2	NaN	Xavier	1
3	NaN	Yolanda	3

Outer Join

Match up on all info found in either Left or Right Table. Show everyone that's in the Log in table and the registrations table. Fill any missing info with NaN

In [235]:
```
pd.merge(registrations,logins,how='outer')
```

Out[235]:

	reg_id	name	log_id
0	1.0	Andrew	2.0
1	2.0	Bobo	4.0
2	3.0	Claire	NaN
3	4.0	David	NaN
4	NaN	Xavier	1.0
5	NaN	Yolanda	3.0

Join on Index or Column

Use combinations of left_on,right_on,left_index,right_index to merge a column or index on each other

In [236]:
```
registrations
```

Out[236]:

	reg_id	name
0	1	Andrew
1	2	Bobo
2	3	Claire
3	4	David

In [237]:
```
logins
```

Out[237]:

	log_id	name
0	1	Xavier
1	2	Andrew
2	3	Yolanda
3	4	Bobo

In [238]:
```
registrations = registrations.set_index("name")
```

In [239]:
```
registrations
```

Out[239]:

	reg_id
name	
Andrew	1
Bobo	2
Claire	3
David	4

In [240]:
```
pd.merge(registrations,logins,left_index=True,right_on='name')
```

Out[240]:

	reg_id	log_id	name
1	1	2	Andrew
3	2	4	Bobo

In [242]:
```
pd.merge(logins,registrations,right_index=True,left_on='name')
```

Out[242]:

	log_id	name	reg_id
1	2	Andrew	1
3	4	Bobo	2

Dealing with differing key column names in joined tables

In [243]:
```
registrations = registrations.reset_index()
```

In [244]:
```
registrations
```

Out[244]:

	name	reg_id
0	Andrew	1
1	Bobo	2
2	Claire	3
3	David	4

In [245]:
```
logins
```

Out[245]:

	log_id	name
0	1	Xavier
1	2	Andrew
2	3	Yolanda
3	4	Bobo

In [246]:
```
registrations.columns = ['reg_name','reg_id']
```

In [247]:
```
registrations
```

Out[247]:

	reg_name	reg_id
0	Andrew	1
1	Bobo	2
2	Claire	3
3	David	4

In [248]:
```
# ERROR
# pd.merge(registrations,logins)
```

In [249]:
```
pd.merge(registrations,logins,left_on='reg_name',right_on='name')
```

Out[249]:

	reg_name	reg_id	log_id	name
0	Andrew	1	2	Andrew
1	Bobo	2	4	Bobo

In [250]:
```
pd.merge(registrations,logins,left_on='reg_name',right_on='name').drop('reg
```

Out[250]:

	reg_id	log_id	name
0	1	2	Andrew
1	2	4	Bobo

Pandas automatically tags duplicate columns

In [255]:
```
registrations.columns = ['name','id']
```

In [256]:
```
logins.columns = ['id','name']
```

In [257]:
```
registrations
```

Out[257]:

	name	id
0	Andrew	1
1	Bobo	2
2	Claire	3
3	David	4

In [258]:
```
logins
```

Out[258]:

	id	name
0	1	Xavier
1	2	Andrew
2	3	Yolanda
3	4	Bobo

In [259]:
```
# _x is for left
# _y is for right
pd.merge(registrations,logins,on='name')
```

Out[259]:

	name	id_x	id_y
0	Andrew	1	2
1	Bobo	2	4

In [260]:
```
pd.merge(registrations,logins,on='name',suffixes=('_reg','_log'))
```

Out[260]:

	name	id_reg	id_log
0	Andrew	1	2
1	Bobo	2	4