



MERGING DATAFRAMES WITH PANDAS

# Reading multiple data files

# Tools for pandas data import

- `pd.read_csv()` for CSV files
  - `dataframe = pd.read_csv(filepath)`
  - dozens of optional input parameters
- Other data import tools:
  - `pd.read_excel()`
  - `pd.read_html()`
  - `pd.read_json()`



# Loading separate files

```
In [1]: import pandas as pd
```

```
In [2]: dataframe0 = pd.read_csv('sales-jan-2015.csv')
```

```
In [3]: dataframe1 = pd.read_csv('sales-feb-2015.csv')
```

# Using a loop

```
In [4]: filenames = ['sales-jan-2015.csv', 'sales-feb-2015.csv']
```

```
In [5]: dataframes = []
```

```
In [6]: for f in filenames:  
...:     dataframes.append(pd.read_csv(f))
```



# Using a comprehension

```
In [7]: filenames = ['sales-jan-2015.csv', 'sales-feb-2015.csv']
```

```
In [8]: dataframes = [pd.read_csv(f) for f in filenames]
```



# Using glob

```
In [9]: from glob import glob
```

```
In [10]: filenames = glob('sales*.csv')
```

```
In [11]: dataframes = [pd.read_csv(f) for f in filenames]
```



MERGING DATAFRAMES WITH PANDAS

**Let's practice!**



MERGING DATAFRAMES WITH PANDAS

# Reindexing DataFrames





# “Indexes” vs. “Indices”

- *indices*: many *index labels* within *Index data structures*
- *indexes*: many pandas *Index data structures*

indexes

indices

	NOC	count	Host NOC	Grand Total	fraction	change
Edition						
1896	GRE	52	GRE	151	0.344371	NaN
1900	FRA	185	FRA	512	0.361328	198.002486
1904	USA	394	USA	470	0.838298	199.651245
1908	GBR	347	GBR	804	0.431592	134.489218
1912	SWE	173	SWE	885	0.195480	71.896226
1920	BEL	188	BEL	1298	0.144838	54.757887
1924	FRA	122	FRA	884	0.138009	2.046362
1928	NED	65	NED	710	0.091549	36.315243
1932	USA	181	USA	615	0.294309	3.739184
1936	GER	210	GER	875	0.240000	24.108011
1948	GBR	56	GBR	814	0.068796	-3.635059
1952	FIN	40	FIN	889	0.044994	0.121662

NOC	AFG	AHO	ALG	ANZ	ARG	ARM	...	VEN	VIE	YUG	ZAM	ZIM	ZZX
Edition													
1896	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
1900	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	33.561198
1904	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	-22.642384
1908	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	0.000000
1912	NaN	NaN	NaN	-26.092774	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	0.000000
1920	NaN	NaN	NaN	0.000000	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	0.000000
1924	NaN	NaN	NaN	0.000000	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	0.000000
1928	NaN	NaN	NaN	0.000000	131.101152	NaN	...	NaN	NaN	323.521127	NaN	NaN	0.000000
1932	NaN	NaN	NaN	0.000000	-25.794206	NaN	...	NaN	NaN	0.000000	NaN	NaN	0.000000
1936	NaN	NaN	NaN	0.000000	-10.271982	NaN	...	NaN	NaN	-29.357594	NaN	NaN	0.000000
1948	NaN	NaN	NaN	0.000000	-4.601500	NaN	...	NaN	NaN	47.596769	NaN	NaN	0.000000
1952	NaN	NaN	NaN	0.000000	-10.508545	NaN	...	NaN	NaN	34.043608	NaN	NaN	0.000000



# Importing weather data

```
In [1]: import pandas as pd
```

```
In [2]: w_mean = pd.read_csv('quarterly_mean_temp.csv', index_col='Month')
```

```
In [3]: w_max = pd.read_csv('quarterly_max_temp.csv', index_col='Month')
```



# Examining the data

```
In [4]: print(w_mean)
      Mean TemperatureF
```

Month	
Apr	61.956044
Jan	32.133333
Jul	68.934783
Oct	43.434783

```
In [5]: print(w_max)
      Max TemperatureF
```

Month	
Jan	68
Apr	89
Jul	91
Oct	84



# The DataFrame indexes

```
In [6]: print(w_mean.index)
Index(['Apr', 'Jan', 'Jul', 'Oct'], dtype='object', name='Month')

In [7]: print(w_max.index)
Index(['Jan', 'Apr', 'Jul', 'Oct'], dtype='object', name='Month')

In [8]: print(type(w_mean.index))
<class 'pandas.indexes.base.Index'>
```



# Using `.reindex()`

```
In [9]: ordered = ['Jan', 'Apr', 'Jul', 'Oct']
```

```
In [10]: w_mean2 = w_mean.reindex(ordered)
```

```
In [11]: print(w_mean2)
          Mean TemperatureF
```

Month	
Jan	32.133333
Apr	61.956044
Jul	68.934783
Oct	43.434783



# Using `.sort_index()`

```
In [12]: w_mean2.sort_index()
```

```
Out[12]:
```

	Mean TemperatureF
Month	
Apr	61.956044
Jan	32.133333
Jul	68.934783
Oct	43.434783



# Reindex from a DataFrame Index

```
In [13]: w_mean.reindex(w_max.index)
```

```
Out[13]:
```

	Mean TemperatureF
Month	
Jan	32.133333
Apr	61.956044
Jul	68.934783
Oct	43.434783





# Reindexing with missing labels

```
In [14]: w_mean3 = w_mean.reindex(['Jan', 'Apr', 'Dec'])
```

```
In [15]: print(w_mean3)
```

Mean TemperatureF

Month

Jan 32.133333

Apr 61.956044

Dec NaN





# Reindex from a DataFrame Index

```
In [16]: w_max.reindex(w_mean3.index)
```

```
Out[16]:
```

	Max TemperatureF
Month	
Jan	68.0
Apr	89.0
Dec	NaN

```
In [17]: w_max.reindex(w_mean3.index).dropna()
```

```
Out[17]:
```

	Max TemperatureF
Month	
Jan	68.0
Apr	89.0



# Order matters

```
In [18]: w_max.reindex(w_mean.index)
```

```
Out[18]:
```

Max TemperatureF

Month

Apr	89
Jan	68
Jul	91
Oct	84

```
In [19]: w_mean.reindex(w_max.index)
```

```
Out[19]:
```

Mean TemperatureF

Month

Jan	32.133333
Apr	61.956044
Jul	68.934783
Oct	43.434783



MERGING DATAFRAMES WITH PANDAS

**Let's practice!**



MERGING DATAFRAMES WITH PANDAS

# **Arithmetic with Series & DataFrames**



# Loading weather data

```
In [1]: import pandas as pd
```

```
In [2]: weather = pd.read_csv('pittsburgh2013.csv',  
    ....:                      index_col='Date', parse_dates=True)
```

```
In [3]: weather.loc['2013-7-1':'2013-7-7', 'PrecipitationIn']
```

```
Out[3]:
```

Date

2013-07-01	0.18
2013-07-02	0.14
2013-07-03	0.00
2013-07-04	0.25
2013-07-05	0.02
2013-07-06	0.06
2013-07-07	0.10

Name: PrecipitationIn, dtype: float64



# Scalar multiplication

```
In [4]: weather.loc['2013-07-01':'2013-07-07', 'PrecipitationIn'] * 2.54
```

```
Out[4]:
```

```
Date
```

```
2013-07-01    0.4572
```

```
2013-07-02    0.3556
```

```
2013-07-03    0.0000
```

```
2013-07-04    0.6350
```

```
2013-07-05    0.0508
```

```
2013-07-06    0.1524
```

```
2013-07-07    0.2540
```

```
Name: PrecipitationIn, dtype: float64
```



# Absolute temperature range

```
In [5]: week1_range = weather.loc['2013-07-01':'2013-07-07',  
    ....:                        ['Min TemperatureF', 'Max TemperatureF']]
```

```
In [6]: print(week1_range)
```

	Min TemperatureF	Max TemperatureF
Date		
2013-07-01	66	79
2013-07-02	66	84
2013-07-03	71	86
2013-07-04	70	86
2013-07-05	69	86
2013-07-06	70	89
2013-07-07	70	77



# Average temperature

```
In [7]: week1_mean = weather.loc['2013-07-01':'2013-07-07',  
    ....:                        'Mean TemperatureF']
```

```
In [8]: print(week1_mean)
```

Date

2013-07-01	72
2013-07-02	74
2013-07-03	78
2013-07-04	77
2013-07-05	76
2013-07-06	78
2013-07-07	72

Name: Mean TemperatureF, dtype: int64





# Relative temperature range

```
In [9]: week1_range / week1_mean
```

```
RuntimeWarning: Cannot compare type 'Timestamp' with type 'str', sort order is  
undefined for incomparable objects
```

```
return this.join(other, how=how, return_indexers=return_indexers)
```

```
Out[9]:
```

```
                2013-07-01 00:00:00    2013-07-02 00:00:00    2013-07-03 00:00:00    \  
Date  
2013-07-01                NaN                NaN                NaN  
2013-07-02                NaN                NaN                NaN  
2013-07-03                NaN                NaN                NaN  
2013-07-04                NaN                NaN                NaN  
2013-07-05                NaN                NaN                NaN  
2013-07-06                NaN                NaN                NaN  
2013-07-07                NaN                NaN                NaN  
  
                2013-07-04 00:00:00    2013-07-05 00:00:00    2013-07-06 00:00:00    \  
Date  
2013-07-01                NaN                NaN                NaN  
... ..
```



# Relative temperature range

```
In [10]: week1_range.divide(week1_mean, axis='rows')
```

```
Out[10]:
```

	Min TemperatureF	Max TemperatureF
Date		
2013-07-01	0.916667	1.097222
2013-07-02	0.891892	1.135135
2013-07-03	0.910256	1.102564
2013-07-04	0.909091	1.116883
2013-07-05	0.907895	1.131579
2013-07-06	0.897436	1.141026
2013-07-07	0.972222	1.069444



# Percentage changes

```
In [11]: week1_mean.pct_change() * 100
```

```
Out[11]:
```

```
Date
```

```
2013-07-01      NaN
```

```
2013-07-02      2.777778
```

```
2013-07-03      5.405405
```

```
2013-07-04     -1.282051
```

```
2013-07-05     -1.298701
```

```
2013-07-06      2.631579
```

```
2013-07-07     -7.692308
```

```
Name: Mean TemperatureF, dtype: float64
```



# Bronze Olympic medals

```
In [12]: bronze = pd.read_csv('bronze_top5.csv', index_col=0)
```

```
In [13]: print(bronze)
```

	Total
Country	
United States	1052.0
Soviet Union	584.0
United Kingdom	505.0
France	475.0
Germany	454.0



# Silver Olympic medals

```
In [14]: silver = pd.read_csv('silver_top5.csv', index_col=0)
```

```
In [15]: print(silver)
```

	Total
Country	
United States	1195.0
Soviet Union	627.0
United Kingdom	591.0
France	461.0
Italy	394.0



# Gold Olympic medals

```
In [16]: gold = pd.read_csv('gold_top5.csv', index_col=0)
```

```
In [17]: print(gold)
```

Country	Total
United States	2088.0
Soviet Union	838.0
United Kingdom	498.0
Italy	460.0
Germany	407.0



# Adding bronze, silver

```
In [18]: bronze + silver
Out[18]:
Country
France          936.0
Germany         NaN
Italy           NaN
Soviet Union    1211.0
United Kingdom  1096.0
United States   2247.0
Name: Total, dtype: float64
```



# Adding bronze, silver

```
In [19]: bronze + silver
```

```
Out[19]:
```

```
Country
```

```
France          936.0
```

```
Germany         NaN
```

```
Italy           NaN
```

```
Soviet Union    1211.0
```

```
United Kingdom  1096.0
```

```
United States   2247.0
```

```
Name: Total, dtype: float64
```

```
In [22]: print(bronze['United States'])
```

```
1052.0
```

```
In [23]: print(silver['United States'])
```

```
1195.0
```





# Using the .add() method

```
In [21]: bronze.add(silver)
```

```
Out[21]:
```

```
Country
```

```
France          936.0
```

```
Germany         NaN
```

```
Italy           NaN
```

```
Soviet Union    1211.0
```

```
United Kingdom  1096.0
```

```
United States   2247.0
```

```
Name: Total, dtype: float64
```



# Using a fill\_value

```
In [22]: bronze.add(silver, fill_value=0)
```

```
Out[22]:
```

```
Country
```

```
France          936.0
```

```
Germany         454.0
```

```
Italy           394.0
```

```
Soviet Union    1211.0
```

```
United Kingdom  1096.0
```

```
United States   2247.0
```

```
Name: Total, dtype: float64
```



# Adding bronze, silver, gold

```
In [23]: bronze + silver + gold
```

```
Out[23]:
```

```
Country
```

France	NaN
Germany	NaN
Italy	NaN

Soviet Union	2049.0
--------------	--------

United Kingdom	1594.0
----------------	--------

United States	4335.0
---------------	--------

```
Name: Total, dtype: float64
```



# Chaining .add()

```
In [24]: bronze.add(silver, fill_value=0).add(gold, fill_value=0)
```

```
Out[24]:
```

```
Country
```

```
France          936.0
```

```
Germany         861.0
```

```
Italy           854.0
```

```
Soviet Union    2049.0
```

```
United Kingdom  1594.0
```

```
United States   4335.0
```

```
Name: Total, dtype: float64
```



MERGING DATAFRAMES WITH PANDAS

**Let's practice!**



MERGING DATAFRAMES WITH PANDAS

# **Appending & concatenating Series**



# append()

- `.append()`: Series & DataFrame *method*
- Invocation:
  - `s1.append(s2)`
- Stacks rows of `s2` below `s1`
- Method for Series & DataFrames



# concat()

- `concat()`: pandas module *function*
- Invocation:
  - `pd.concat([s1, s2, s3])`
- Can stack row-wise or column-wise





# concat() & .append()

- Equivalence of concat() & .append():
  - `result1 = pd.concat([s1, s2, s3])`
  - `result2 = s1.append(s2).append(s3)`
- `result1 == result2` elementwise



# Series of US states

```
In [1]: import pandas as pd
```

```
In [2]: northeast = pd.Series(['CT', 'ME', 'MA', 'NH', 'RI', 'VT',  
....: 'NJ', 'NY', 'PA'])
```

```
In [3]: south = pd.Series(['DE', 'FL', 'GA', 'MD', 'NC', 'SC', 'VA',  
....: 'DC', 'WV', 'AL', 'KY', 'MS', 'TN', 'AR', 'LA', 'OK', 'TX'])
```

```
In [4]: midwest = pd.Series(['IL', 'IN', 'MN', 'MO', 'NE', 'ND',  
....: 'SD', 'IA', 'KS', 'MI', 'OH', 'WI'])
```

```
In [5]: west = pd.Series(['AZ', 'CO', 'ID', 'MT', 'NV', 'NM',  
....: 'UT', 'WY', 'AK', 'CA', 'HI', 'OR', 'WA'])
```



# Using .append()

```
In [6]: east = northeast.append(south)
```

```
In [7]: print(east)
```

0	CT
1	ME
2	MA
3	NH
4	RI
5	VT
6	NJ
7	NY
8	PA

0	DE
1	FL
2	GA
3	MD
4	NC
5	SC
6	VA

7	DC
8	WV
9	AL
10	KY
11	MS
12	TN
13	AR
14	LA
15	OK
16	TX

dtype: object



# The appended Index

```
In [8]: print(east.index)
Int64Index([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  0,  1,  2,  3,  4,
            5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16], dtype='int64')
```

```
In [9]: print(east.loc[3])
3    NH
3    MD
dtype: object
```



# Using `.reset_index()`

```
In [10]: new_east = northeast.append(south).reset_index(drop=True)
```

```
In [11]: print(new_east.head(11))
```

```
0      CT
```

```
1      ME
```

```
2      MA
```

```
3      NH
```

```
4      RI
```

```
5      VT
```

```
6      NJ
```

```
7      NY
```

```
8      PA
```

```
9      DE
```

```
10     FL
```

```
dtype: object
```

```
In [12]: print(new_east.index)
```

```
RangeIndex(start=0, stop=26, step=1)
```

# Using concat()

```
In [13]: east = pd.concat([northeast, south])
```

```
In [14]: print(east.head(11))
```

```
0    CT
1    ME
2    MA
3    NH
4    RI
5    VT
6    NJ
7    NY
8    PA
0    DE
1    FL
```

```
dtype: object
```

```
In [15]: print(east.index)
```

```
Int64Index([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  0,  1,  2,  3,  4,
            5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16], dtype='int64')
```



# Using ignore\_index

```
In [16]: new_east = pd.concat([northeast, south],  
    ...:                      ignore_index=True)
```

```
In [17]: print(new_east.head(11))
```

```
0      CT  
1      ME  
2      MA  
3      NH  
4      RI  
5      VT  
6      NJ  
7      NY  
8      PA  
9      DE  
10     FL  
dtype: object
```

```
In [18]: print(new_east.index)  
RangeIndex(start=0, stop=26, step=1)
```



MERGING DATAFRAMES WITH PANDAS

**Let's practice!**





MERGING DATAFRAMES WITH PANDAS

# **Appending & concatenating DataFrames**



# Loading population data

```
In [1]: import pandas as pd
```

```
In [2]: pop1 = pd.read_csv('population_01.csv', index_col=0)
```

```
In [3]: pop2 = pd.read_csv('population_02.csv', index_col=0)
```

```
In [4]: print(type(pop1), pop1.shape)
<class 'pandas.core.frame.DataFrame'> (4, 1)
```

```
In [5]: print(type(pop2), pop2.shape)
<class 'pandas.core.frame.DataFrame'> (4, 1)
```



# Examining population data

```
In [6]: print(pop1)
```

```
2010 Census Population
```

```
Zip Code ZCTA
```

66407	479
72732	4716
50579	2405
46241	30670

```
In [7]: print(pop2)
```

```
2010 Census Population
```

```
Zip Code ZCTA
```

12776	2180
76092	26669
98360	12221
49464	27481



# Appending population DataFrames

```
In [8]: pop1.append(pop2)
```

```
Out[8]:
```

Zip Code ZCTA	2010 Census Population
66407	479
72732	4716
50579	2405
46241	30670
12776	2180
76092	26669
98360	12221
49464	27481

```
In [9]: print(pop1.index.name, pop1.columns)
```

```
Zip Code ZCTA Index(['2010 Census Population'], dtype='object')
```

```
In [10]: print(pop2.index.name, pop2.columns)
```

```
Zip Code ZCTA Index(['2010 Census Population'], dtype='object')
```



# Population & unemployment data

```
In [11]: population = pd.read_csv('population_00.csv',  
    ....:                          index_col=0)
```

```
In [12]: unemployment = pd.read_csv('unemployment_00.csv',  
    ....:                             index_col=0)
```

```
In [13]: print(population)
```

2010 Census Population	
Zip Code ZCTA	
57538	322
59916	130
37660	40038
2860	45199

```
In [14]: print(unemployment)
```

unemployment participants		
Zip		
2860	0.11	34447
46167	0.02	4800
1097	0.33	42
80808	0.07	4310



# Appending population & unemployment

```
In [15]: population.append(unemployment)
```

```
Out[15]:
```

	2010 Census Population	participants	unemployment
57538	322.0	NaN	NaN
59916	130.0	NaN	NaN
37660	40038.0	NaN	NaN
2860	45199.0	NaN	NaN
2860	NaN	34447.0	0.11
46167	NaN	4800.0	0.02
1097	NaN	42.0	0.33
80808	NaN	4310.0	0.07



# Repeated index labels

```
In [15]: population.append(unemployment)
```

```
Out[15]:
```

	2010 Census Population	participants	unemployment
57538	322.0	NaN	NaN
59916	130.0	NaN	NaN
37660	40038.0	NaN	NaN
2860	45199.0	NaN	NaN
2860	NaN	34447.0	0.11
46167	NaN	4800.0	0.02
1097	NaN	42.0	0.33
80808	NaN	4310.0	0.07



# Concatenating rows

```
In [16]: pd.concat([population, unemployment], axis=0)
```

```
Out[16]:
```

	2010 Census Population	participants	unemployment
57538	322.0	NaN	NaN
59916	130.0	NaN	NaN
37660	40038.0	NaN	NaN
2860	45199.0	NaN	NaN
2860	NaN	34447.0	0.11
46167	NaN	4800.0	0.02
1097	NaN	42.0	0.33
80808	NaN	4310.0	0.07





# Concatenating columns

```
In [17]: pd.concat([population, unemployment], axis=1)
```

```
Out[17]:
```

	2010 Census Population	unemployment	participants
1097	NaN	0.33	42.0
2860	45199.0	0.11	34447.0
37660	40038.0	NaN	NaN
46167	NaN	0.02	4800.0
57538	322.0	NaN	NaN
59916	130.0	NaN	NaN
80808	NaN	0.07	4310.0



MERGING DATAFRAMES WITH PANDAS

**Let's practice!**



MERGING DATAFRAMES WITH PANDAS

# **Concatenation, keys, & MultiIndexes**



# Loading rainfall data

```
In [1]: import pandas as pd
```

```
In [2]: file1 = 'q1_rainfall_2013.csv'
```

```
In [3]: rain2013 = pd.read_csv(file1, index_col='Month', parse_dates=True)
```

```
In [4]: file2 = 'q1_rainfall_2014.csv'
```

```
In [5]: rain2014 = pd.read_csv(file2, index_col='Month', parse_dates=True)
```



# Examining rainfall data

```
In [6]: print(rain2013)
```

	Precipitation
--	---------------

Month	
-------	--

Jan	0.096129
Feb	0.067143
Mar	0.061613

```
In [7]: print(rain2014)
```

	Precipitation
--	---------------

Month	
-------	--

Jan	0.050323
Feb	0.082143
Mar	0.070968



# Concatenating rows

```
In [8]: pd.concat([rain2013, rain2014], axis=0)
```

```
Out[8]:
```

	Precipitation
Jan	0.096129
Feb	0.067143
Mar	0.061613
Jan	0.050323
Feb	0.082143
Mar	0.070968



# Using multi-index on rows

```
In [7]: rain1314 = pd.concat([rain2013, rain2014], keys=[2013, 2014], axis=0)
```

```
In [8]: print(rain1314)
```

		Precipitation
2013	Jan	0.096129
	Feb	0.067143
	Mar	0.061613
2014	Jan	0.050323
	Feb	0.082143
	Mar	0.070968



# Accessing a multi-index

```
In [9]: print(rain1314.loc[2014])
```

Precipitation	
Jan	0.050323
Feb	0.082143
Mar	0.070968





# Concatenating columns

```
In [10]: rain1314 = pd.concat([rain2013, rain2014], axis='columns')
```

```
In [11]: print(rain1314)
```

	Precipitation	Precipitation
Jan	0.096129	0.050323
Feb	0.067143	0.082143
Mar	0.061613	0.070968



# Using a multi-index on columns

```
In [12]: rain1314 = pd.concat([rain2013, rain2014], keys=[2013, 2014], axis='columns')
```

```
In [13]: print(rain1314)
```

	2013	2014
	Precipitation	Precipitation
Jan	0.096129	0.050323
Feb	0.067143	0.082143
Mar	0.061613	0.070968

```
In [14]: rain1314[2013]
```

```
Out[14]:
```

	Precipitation
Jan	0.096129
Feb	0.067143
Mar	0.061613



# pd.concat() with dict

```
In [15]: rain_dict = {2013: rain2013, 2014: rain2014}

In [16]: rain1314 = pd.concat(rain_dict, axis='columns')

In [17]: print(rain1314)
```

	2013	2014
	Precipitation	Precipitation
Jan	0.096129	0.050323
Feb	0.067143	0.082143
Mar	0.061613	0.070968



MERGING DATAFRAMES WITH PANDAS

**Let's practice!**



MERGING DATAFRAMES WITH PANDAS

# Outer & inner joins



# Using with arrays

```
In [1]: import numpy as np
```

```
In [2]: import pandas as pd
```

```
In [3]: A = np.arange(8).reshape(2,4) + 0.1
```

```
In [4]: print(A)
```

```
[[ 0.1  1.1  2.1  3.1]
 [ 4.1  5.1  6.1  7.1]]
```

```
In [5]: B = np.arange(6).reshape(2,3) + 0.2
```

```
In [6]: print(B)
```

```
[[ 0.2  1.2  2.2]
 [ 3.2  4.2  5.2]]
```

```
In [7]: C = np.arange(12).reshape(3,4) + 0.3
```

```
In [8]: print(C)
```

```
[[ 0.3  1.3  2.3  3.3]
 [ 4.3  5.3  6.3  7.3]
 [ 8.3  9.3 10.3 11.3]]
```



# Stacking arrays horizontally

```
In [6]: np.hstack([B, A])
```

```
Out[6]:
```

```
array([[ 0.2,  1.2,  2.2,  0.1,  1.1,  2.1,  3.1],  
       [ 3.2,  4.2,  5.2,  4.1,  5.1,  6.1,  7.1]])
```

```
In [7]: np.concatenate([B, A], axis=1)
```

```
Out[7]:
```

```
array([[ 0.2,  1.2,  2.2,  0.1,  1.1,  2.1,  3.1],  
       [ 3.2,  4.2,  5.2,  4.1,  5.1,  6.1,  7.1]])
```



# Stacking arrays vertically

```
In [8]: np.vstack([A, C])
```

```
Out[8]:
```

```
array([[ 0.1,  1.1,  2.1,  3.1],  
       [ 4.1,  5.1,  6.1,  7.1],  
       [ 0.3,  1.3,  2.3,  3.3],  
       [ 4.3,  5.3,  6.3,  7.3],  
       [ 8.3,  9.3, 10.3, 11.3]])
```

```
In [9]: np.concatenate([A, C], axis=0)
```

```
Out[9]:
```

```
array([[ 0.1,  1.1,  2.1,  3.1],  
       [ 4.1,  5.1,  6.1,  7.1],  
       [ 0.3,  1.3,  2.3,  3.3],  
       [ 4.3,  5.3,  6.3,  7.3],  
       [ 8.3,  9.3, 10.3, 11.3]])
```





# Incompatible array dimensions

```
In [11]: np.concatenate([A, B], axis=0) # incompatible columns
```

```
-----  
ValueError                                Traceback (most recent call last)  
----> 1 np.concatenate([A, B], axis=0) # incompatible columns
```

ValueError: all the input array dimensions except for the concatenation axis must match exactly

```
In [12]: np.concatenate([A, C], axis=1) # incompatible rows
```

```
-----  
ValueError                                Traceback (most recent call last)  
----> 1 np.concatenate([A, C], axis=1) # incompatible rows
```

ValueError: all the input array dimensions except for the concatenation axis must match exactly



# Population & unemployment data

```
In [13]: population = pd.read_csv('population_00.csv',  
....:                             index_col=0)
```

```
In [14]: unemployment = pd.read_csv('unemployment_00.csv',  
...:                                index_col=0)
```

```
In [15]: print(population)
```

2010 Census Population	
Zip Code ZCTA	
57538	322
59916	130
37660	40038
2860	45199

```
In [16]: print(unemployment)
```

unemployment participants		
Zip		
2860	0.11	34447
46167	0.02	4800
1097	0.33	42
80808	0.07	4310



# Converting to arrays

```
In [17]: population_array = np.array(population)
```

```
In [18]: print(population_array) # Index info is lost
```

```
[[ 322]
 [ 130]
 [40038]
 [45199]]
```

```
In [19]: unemployment_array = np.array(unemployment)
```

```
In [20]: print(population_array)
```

```
[[ 1.10000000e-01  3.44470000e+04]
 [ 2.00000000e-02  4.80000000e+03]
 [ 3.30000000e-01  4.20000000e+01]
 [ 7.00000000e-02  4.31000000e+03]]
```



# Manipulating data as arrays

```
In [21]: print(np.concatenate([population_array, unemployment_array],  
    ....:                      axis=1))
```

```
[[ 3.22000000e+02  1.10000000e-01  3.44470000e+04]  
 [ 1.30000000e+02  2.00000000e-02  4.80000000e+03]  
 [ 4.00380000e+04  3.30000000e-01  4.20000000e+01]  
 [ 4.51990000e+04  7.00000000e-02  4.31000000e+03]]
```



# Joins

- Joining tables: Combining rows of multiple tables
- Outer join
  - Union of index sets (all labels, no repetition)
  - Missing fields filled with NaN
- Inner join
  - Intersection of index sets (only common labels)



# Concatenation & inner join

```
In [22]: pd.concat([population, unemployment], axis=1, join='inner')
```

```
Out[22]:
```

	2010 Census Population	unemployment	participants
2860	45199	0.11	34447



# Concatenation & outer join

```
In [23]: pd.concat([population, unemployment], axis=1, join='outer')
```

```
Out[23]:
```

	2010 Census Population	unemployment	participants
1097	NaN	0.33	42.0
2860	45199.0	0.11	34447.0
37660	40038.0	NaN	NaN
46167	NaN	0.02	4800.0
57538	322.0	NaN	NaN
59916	130.0	NaN	NaN
80808	NaN	0.07	4310.0



# Inner join on other axis

```
In [24]: pd.concat([population, unemployment], join='inner', axis=0)
Out[24]:
Empty DataFrame
Columns: []
Index: [2860, 46167, 1097, 80808, 57538, 59916, 37660, 2860]
```





MERGING DATAFRAMES WITH PANDAS

**Let's practice!**



MERGING DATAFRAMES WITH PANDAS

# Merging DataFrames



# Population DataFrame

```
In [1]: import pandas as pd
```

```
In [2]: population = pd.read_csv('pa_zipcode_population.csv')
```

```
In [3]: print(population)
```

	Zipcode	2010 Census Population
0	16855	282
1	15681	5241
2	18657	11985
3	17307	5899
4	15635	220



# Cities DataFrame

```
In [4]: cities = pd.read_csv('pa_zipcode_city.csv')
```

```
In [5]: print(cities)
```

	Zipcode	City	State
0	17545	MANHEIM	PA
1	18455	PRESTON PARK	PA
2	17307	BIGLERVILLE	PA
3	15705	INDIANA	PA
4	16833	CURWENSVILLE	PA
5	16220	CROWN	PA
6	18618	HARVEYS LAKE	PA
7	16855	MINERAL SPRINGS	PA
8	16623	CASSVILLE	PA
9	15635	HANNASTOWN	PA
10	15681	SALTSBURG	PA
11	18657	TUNKHANNOCK	PA
12	15279	PITTSBURGH	PA
13	17231	LEMASTERS	PA
14	18821	GREAT BEND	PA



# Merging

```
In [6]: pd.merge(population, cities)
```

```
Out[6]:
```

	Zipcode	2010 Census Population	City	State
0	16855	282	MINERAL SPRINGS	PA
1	15681	5241	SALTSBURG	PA
2	18657	11985	TUNKHANNOCK	PA
3	17307	5899	BIGLERVILLE	PA
4	15635	220	HANNASTOWN	PA



# Medal DataFrames

```
In [7]: bronze = pd.read_csv('bronze_sorted.csv')
```

```
In [8]: gold = pd.read_csv('gold_sorted.csv')
```

```
In [9]: print(bronze)
```

	NOC	Country	Total
0	USA	United States	1052.0
1	URS	Soviet Union	584.0
2	GBR	United Kingdom	505.0
3	FRA	France	475.0
4	GER	Germany	454.0

```
In [10]: print(gold)
```

	NOC	Country	Total
0	USA	United States	2088.0
1	URS	Soviet Union	838.0
2	GBR	United Kingdom	498.0
3	ITA	Italy	460.0
4	GER	Germany	407.0



# Merging all columns

```
In [11]: pd.merge(bronze, gold)
Out[11]:
Empty DataFrame
Columns: [NOC, Country, Total]
Index: []
```



# Merging on

```
In [12]: pd.merge(bronze, gold, on='NOC')
```

```
Out[12]:
```

	NOC	Country_x	Total_x	Country_y	Total_y
0	USA	United States	1052.0	United States	2088.0
1	URS	Soviet Union	584.0	Soviet Union	838.0
2	GBR	United Kingdom	505.0	United Kingdom	498.0
3	GER	Germany	454.0	Germany	407.0





# Merging on multiple columns

```
In [13]: pd.merge(bronze, gold, on=['NOC', 'Country'])
```

```
Out[13]:
```

	NOC	Country	Total_x	Total_y
0	USA	United States	1052.0	2088.0
1	URS	Soviet Union	584.0	838.0
2	GBR	United Kingdom	505.0	498.0
3	GER	Germany	454.0	407.0



# Using suffixes

```
In [14]: pd.merge(bronze, gold, on=['NOC', 'Country'], suffixes=['_bronze', '_gold'])
```

```
Out[14]:
```

	NOC	Country	Total_bronze	Total_gold
0	USA	United States	1052.0	2088.0
1	URS	Soviet Union	584.0	838.0
2	GBR	United Kingdom	505.0	498.0
3	GER	Germany	454.0	407.0



# Counties DataFrame

```
In [15]: counties = pd.read_csv('pa_counties.csv')
```

```
In [16]: print(counties)
```

	CITY NAME	COUNTY NAME
0	SALTSBURG	INDIANA
1	MINERAL SPRINGS	CLEARFIELD
2	BIGLERVILLE	ADAMS
3	HANNASTOWN	WESTMORELAND
4	TUNKHANNOCK	WYOMING

```
In [17]: print(cities.tail())
```

	Zipcode	City	State
10	15681	SALTSBURG	PA
11	18657	TUNKHANNOCK	PA
12	15279	PITTSBURGH	PA
13	17231	LEMASTERS	PA
14	18821	GREAT BEND	PA



# Specifying columns to merge

```
In [18]: pd.merge(counties, cities, left_on='CITY NAME', right_on='City')  
Out[18]:
```

	CITY NAME	COUNTY NAME	Zipcode	City	State
0	SALTSBURG	INDIANA	15681	SALTSBURG	PA
1	MINERAL SPRINGS	CLEARFIELD	16855	MINERAL SPRINGS	PA
2	BIGLERVILLE	ADAMS	17307	BIGLERVILLE	PA
3	HANNASTOWN	WESTMORELAND	15635	HANNASTOWN	PA
4	TUNKHANNOCK	WYOMING	18657	TUNKHANNOCK	PA



# Switching left/right DataFrames

```
In [19]: pd.merge(cities, counties, left_on='City', right_on='CITY NAME')  
Out[19]:
```

	Zipcode	City	State	CITY NAME	COUNTY NAME
0	17307	BIGLERVILLE	PA	BIGLERVILLE	ADAMS
1	16855	MINERAL SPRINGS	PA	MINERAL SPRINGS	CLEARFIELD
2	15635	HANNASTOWN	PA	HANNASTOWN	WESTMORELAND
3	15681	SALTSBURG	PA	SALTSBURG	INDIANA
4	18657	TUNKHANNOCK	PA	TUNKHANNOCK	WYOMING



MERGING DATAFRAMES WITH PANDAS

**Let's practice!**



MERGING DATAFRAMES WITH PANDAS

# Joining DataFrames



# Medal DataFrames

```
In [1]: import pandas as pd
```

```
In [2]: bronze = pd.read_csv('bronze_sorted.csv')
```

```
In [3]: gold = pd.read_csv('gold_sorted.csv')
```

```
In [4]: print(bronze)
```

	NOC	Country	Total
0	USA	United States	1052.0
1	URS	Soviet Union	584.0
2	GBR	United Kingdom	505.0
3	FRA	France	475.0
4	GER	Germany	454.0

```
In [5]: print(gold)
```

	NOC	Country	Total
0	USA	United States	2088.0
1	URS	Soviet Union	838.0
2	GBR	United Kingdom	498.0
3	ITA	Italy	460.0
4	GER	Germany	407.0





# Merging with inner join

```
In [6]: pd.merge(bronze, gold, on=['NOC', 'Country'],  
....:           suffixes=['_bronze', '_gold'], how='inner')
```

Out[6]:

	NOC	Country	Total_bronze	Total_gold
0	USA	United States	1052.0	2088.0
1	URS	Soviet Union	584.0	838.0
2	GBR	United Kingdom	505.0	498.0
3	GER	Germany	454.0	407.0



# Merging with left join

- Keeps all rows of the left DF in the merged DF
- For rows in the left DF with matches in the right DF:
  - Non-joining columns of right DF are appended to left DF
- For rows in the left DF with no matches in the right DF:
  - Non-joining columns are filled with nulls



# Merging with left join

```
In [7]: pd.merge(bronze, gold, on=['NOC', 'Country'],  
....:           suffixes=['_bronze', '_gold'], how='left')
```

```
Out[7]:
```

	NOC	Country	Total_bronze	Total_gold
0	USA	United States	1052.0	2088.0
1	URS	Soviet Union	584.0	838.0
2	GBR	United Kingdom	505.0	498.0
3	FRA	France	475.0	NaN
4	GER	Germany	454.0	407.0



# Merging with right join

```
In [8]: pd.merge(bronze, gold, on=['NOC', 'Country'],  
....:           suffixes=['_bronze', '_gold'], how='right')
```

```
Out[8]:
```

	NOC	Country	Total_bronze	Total_gold
0	USA	United States	1052.0	2088.0
1	URS	Soviet Union	584.0	838.0
2	GBR	United Kingdom	505.0	498.0
3	GER	Germany	454.0	407.0
4	ITA	Italy	NaN	460.0



# Merging with outer join

```
In [9]: pd.merge(bronze, gold, on=['NOC', 'Country'],  
....:           suffixes=['_bronze', '_gold'], how='outer')
```

```
Out[9]:
```

	NOC	Country	Total_bronze	Total_gold
0	USA	United States	1052.0	2088.0
1	URS	Soviet Union	584.0	838.0
2	GBR	United Kingdom	505.0	498.0
3	FRA	France	475.0	NaN
4	GER	Germany	454.0	407.0
5	ITA	Italy	NaN	460.0



# Population & unemployment data

```
In [10]: population = pd.read_csv('population_00.csv', index_col=0)
```

```
In [11]: unemployment = pd.read_csv('unemployment_00.csv', index_col=0)
```

```
In [12]: print(population)
```

```
                2010 Census Population
Zip Code ZCTA
57538                322
59916                130
37660            40038
2860            45199
```

```
In [13]: print(unemployment)
```

```
      unemployment  participants
Zip
2860            0.11        34447
46167           0.02         4800
1097            0.33           42
80808           0.07        4310
```



# Using `.join(how='left')`

```
In [16]: population.join(unemployment)
```

```
Out[16]:
```

	2010 Census Population	unemployment	participants
Zip Code ZCTA			
57538	322	NaN	NaN
59916	130	NaN	NaN
37660	40038	NaN	NaN
2860	45199	0.11	34447.0



# Using `.join(how='right')`

```
In [17]: population.join(unemployment, how= 'right')
```

```
Out[17]:
```

	2010 Census Population	unemployment	participants
Zip			
2860	45199.0	0.11	34447
46167	NaN	0.02	4800
1097	NaN	0.33	42
80808	NaN	0.07	4310





# Using `.join(how='inner')`

```
In [18]: population.join(unemployment, how='inner')
```

```
Out[18]:
```

	2010 Census Population	unemployment	participants
2860	45199	0.11	34447



# Using `.join(how='outer')`

```
In [19]: population.join(unemployment, how= 'outer')
```

```
Out[19]:
```

	2010 Census Population	unemployment	participants
1097	NaN	0.33	42.0
2860	45199.0	0.11	34447.0
37660	40038.0	NaN	NaN
46167	NaN	0.02	4800.0
57538	322.0	NaN	NaN
59916	130.0	NaN	NaN
80808	NaN	0.07	4310.0



# Which should you use?

- `df1.append(df2)`: stacking vertically
- `pd.concat([df1, df2])`:
  - stacking many horizontally or vertically
  - simple inner/outer joins on Indexes
- `df1.join(df2)`: inner/outer/left/right joins on Indexes
- `pd.merge(df1, df2)`: many joins on multiple columns



MERGING DATAFRAMES WITH PANDAS

**Let's practice!**



MERGING DATAFRAMES WITH PANDAS

# Ordered merges



# Software & hardware sales

```
In [1]: import pandas as pd

In [2]: software = pd.read_csv('feb-sales-Software.csv', parse_dates=['Date'])
      ....:         .sort_values('Date')

In [3]: hardware = pd.read_csv('feb-sales-Hardware.csv', parse_dates=['Date'])
      ....:         .sort_values('Date')
```



# Software & hardware sales

```
In [4]: print(software)
```

	Date	Company	Product	Units
2	2015-02-02 08:33:01	Hooli	Software	3
1	2015-02-03 14:14:18	Initech	Software	13
7	2015-02-04 15:36:29	Streeplex	Software	13
3	2015-02-05 01:53:06	Acme Coporation	Software	19
5	2015-02-09 13:09:55	Mediacore	Software	7
4	2015-02-11 20:03:08	Initech	Software	7
6	2015-02-11 22:50:44	Hooli	Software	4
0	2015-02-16 12:09:19	Hooli	Software	10
8	2015-02-21 05:01:26	Mediacore	Software	3

```
In [5]: print(hardware)
```

	Date	Company	Product	Units
3	2015-02-02 20:54:49	Mediacore	Hardware	9
0	2015-02-04 21:52:45	Acme Coporation	Hardware	14
1	2015-02-07 22:58:10	Acme Coporation	Hardware	1
2	2015-02-19 10:59:33	Mediacore	Hardware	16
4	2015-02-21 20:41:47	Hooli	Hardware	3



# Using merge()

```
In [6]: pd.merge(hardware, software)
Out[6]:
Empty DataFrame
Columns: [Date, Company, Product, Units]
Index: []
```





# Using merge(how='outer')

```
In [7]: pd.merge(hardware, software, how='outer')
```

```
Out[7]:
```

	Date	Company	Product	Units
0	2015-02-02 20:54:49	Mediacore	Hardware	9
1	2015-02-04 21:52:45	Acme Coporation	Hardware	14
2	2015-02-07 22:58:10	Acme Coporation	Hardware	1
3	2015-02-19 10:59:33	Mediacore	Hardware	16
4	2015-02-21 20:41:47	Hooli	Hardware	3
5	2015-02-02 08:33:01	Hooli	Software	3
6	2015-02-03 14:14:18	Initech	Software	13
7	2015-02-04 15:36:29	Streeplex	Software	13
8	2015-02-05 01:53:06	Acme Coporation	Software	19
9	2015-02-09 13:09:55	Mediacore	Software	7
10	2015-02-11 20:03:08	Initech	Software	7
11	2015-02-11 22:50:44	Hooli	Software	4
12	2015-02-16 12:09:19	Hooli	Software	10
13	2015-02-21 05:01:26	Mediacore	Software	3



# Sorting merge(how='outer')

```
In [8]: pd.merge(hardware, software, how='outer').sorted_values('Date')
```

```
Out[8]:
```

	Date	Company	Product	Units
0	2015-02-02 20:54:49	Mediacore	Hardware	9
1	2015-02-04 21:52:45	Acme Coporation	Hardware	14
2	2015-02-07 22:58:10	Acme Coporation	Hardware	1
3	2015-02-19 10:59:33	Mediacore	Hardware	16
4	2015-02-21 20:41:47	Hooli	Hardware	3
5	2015-02-02 08:33:01	Hooli	Software	3
6	2015-02-03 14:14:18	Initech	Software	13
7	2015-02-04 15:36:29	Streeplex	Software	13
8	2015-02-05 01:53:06	Acme Coporation	Software	19
9	2015-02-09 13:09:55	Mediacore	Software	7
10	2015-02-11 20:03:08	Initech	Software	7
11	2015-02-11 22:50:44	Hooli	Software	4
12	2015-02-16 12:09:19	Hooli	Software	10
13	2015-02-21 05:01:26	Mediacore	Software	3



# Using `merge_ordered()`

```
In [9]: pd.merge_ordered(hardware, software)
```

```
Out[9]:
```

	Date	Company	Product	Units
0	2015-02-02 08:33:01	Hooli	Software	3.0
1	2015-02-02 20:54:49	Mediacore	Hardware	9.0
2	2015-02-03 14:14:18	Initech	Software	13.0
3	2015-02-04 15:36:29	Streeplex	Software	13.0
4	2015-02-04 21:52:45	Acme Coporation	Hardware	14.0
5	2015-02-05 01:53:06	Acme Coporation	Software	19.0
6	2015-02-07 22:58:10	Acme Coporation	Hardware	1.0
7	2015-02-09 13:09:55	Mediacore	Software	7.0
8	2015-02-11 20:03:08	Initech	Software	7.0
9	2015-02-11 22:50:44	Hooli	Software	4.0
10	2015-02-16 12:09:19	Hooli	Software	10.0
11	2015-02-19 10:59:33	Mediacore	Hardware	16.0
12	2015-02-21 05:01:26	Mediacore	Software	3.0
13	2015-02-21 20:41:47	Hooli	Hardware	3.0



# Using on & suffixes

```
In [10]: pd.merge_ordered(hardware, software, on=['Date', 'Company'],  
      ....:               suffixes=['_hardware', '_software']).head()
```

Out[10]:

	Date	Company	Product_hardware \
0	2015-02-02 08:33:01	Hooli	NaN
1	2015-02-02 20:54:49	Mediacore	Hardware
2	2015-02-03 14:14:18	Initech	NaN
3	2015-02-04 15:36:29	Streeplex	NaN
4	2015-02-04 21:52:45	Acme Coporation	Hardware

	Units_hardware	Product_software	Units_software
0	NaN	Software	3.0
1	9.0	NaN	NaN
2	NaN	Software	13.0
3	NaN	Software	13.0
4	14.0	NaN	NaN



# Stocks data

```
In [11]: stocks = pd.read_csv('stocks-2013.csv')
```

```
In [12]: print(stocks)
```

	Date	AAPL	IBM	CSCO	MSFT
0	2013-01-31	497.822381	197.271905	20.699524	27.236667
1	2013-02-28	456.808953	200.735788	20.988947	27.704211
2	2013-03-31	441.840998	210.978001	21.335000	28.141000
3	2013-04-30	419.764998	204.733636	20.914545	29.870909
4	2013-05-31	446.452730	205.263639	22.386364	33.950909
5	2013-06-30	425.537999	200.850000	24.375500	34.632500
6	2013-07-31	429.157272	194.354546	25.378636	33.650454
7	2013-08-31	484.843635	187.125000	24.948636	32.485000
8	2013-09-30	480.184499	188.767000	24.080000	32.523500
9	2013-10-31	504.744783	180.710002	22.847391	34.382174
10	2013-11-30	524.616499	181.333502	22.204000	37.362500
11	2013-12-31	559.657613	179.114763	21.257619	37.455715





# GDP data

```
In [13]: gdp = pd.read_csv('gdp-2013.csv')
```

```
In [14]: print(gdp)
```

	Date	GDP
0	2012-03-31	15973.9
1	2012-06-30	16121.9
2	2012-09-30	16227.9
3	2012-12-31	16297.3
4	2013-03-31	16475.4
5	2013-06-30	16541.4
6	2013-09-30	16749.3
7	2013-12-31	16999.9



# Ordered merge

```
In [15]: pd.merge_ordered(stocks, gdp, on='Date')
```

```
Out[15]:
```

	Date	AAPL	IBM	CSCO	MSFT	GDP
0	2012-03-31	NaN	NaN	NaN	NaN	15973.9
1	2012-06-30	NaN	NaN	NaN	NaN	16121.9
2	2012-09-30	NaN	NaN	NaN	NaN	16227.9
3	2012-12-31	NaN	NaN	NaN	NaN	16297.3
4	2013-01-31	497.822381	197.271905	20.699524	27.236667	NaN
5	2013-02-28	456.808953	200.735788	20.988947	27.704211	NaN
6	2013-03-31	441.840998	210.978001	21.335000	28.141000	16475.4
7	2013-04-30	419.764998	204.733636	20.914545	29.870909	NaN
8	2013-05-31	446.452730	205.263639	22.386364	33.950909	NaN
9	2013-06-30	425.537999	200.850000	24.375500	34.632500	16541.4
10	2013-07-31	429.157272	194.354546	25.378636	33.650454	NaN
11	2013-08-31	484.843635	187.125000	24.948636	32.485000	NaN
12	2013-09-30	480.184499	188.767000	24.080000	32.523500	16749.3
13	2013-10-31	504.744783	180.710002	22.847391	34.382174	NaN
14	2013-11-30	524.616499	181.333502	22.204000	37.362500	NaN
15	2013-12-31	559.657613	179.114763	21.257619	37.455715	16999.9



# Ordered merge with ffill

```
In [16]: pd.merge_ordered(stocks, gdp, on='Date', fill_method='ffill')
```

```
Out[16]:
```

	Date	AAPL	IBM	CSCO	MSFT	GDP
0	2012-03-31	NaN	NaN	NaN	NaN	15973.9
1	2012-06-30	NaN	NaN	NaN	NaN	16121.9
2	2012-09-30	NaN	NaN	NaN	NaN	16227.9
3	2012-12-31	NaN	NaN	NaN	NaN	16297.3
4	2013-01-31	497.822381	197.271905	20.699524	27.236667	16297.3
5	2013-02-28	456.808953	200.735788	20.988947	27.704211	16297.3
6	2013-03-31	441.840998	210.978001	21.335000	28.141000	16475.4
7	2013-04-30	419.764998	204.733636	20.914545	29.870909	16475.4
8	2013-05-31	446.452730	205.263639	22.386364	33.950909	16475.4
9	2013-06-30	425.537999	200.850000	24.375500	34.632500	16541.4
10	2013-07-31	429.157272	194.354546	25.378636	33.650454	16541.4
11	2013-08-31	484.843635	187.125000	24.948636	32.485000	16541.4
12	2013-09-30	480.184499	188.767000	24.080000	32.523500	16749.3
13	2013-10-31	504.744783	180.710002	22.847391	34.382174	16749.3
14	2013-11-30	524.616499	181.333502	22.204000	37.362500	16749.3
15	2013-12-31	559.657613	179.114763	21.257619	37.455715	16999.9





MERGING DATAFRAMES WITH PANDAS

**Let's practice!**

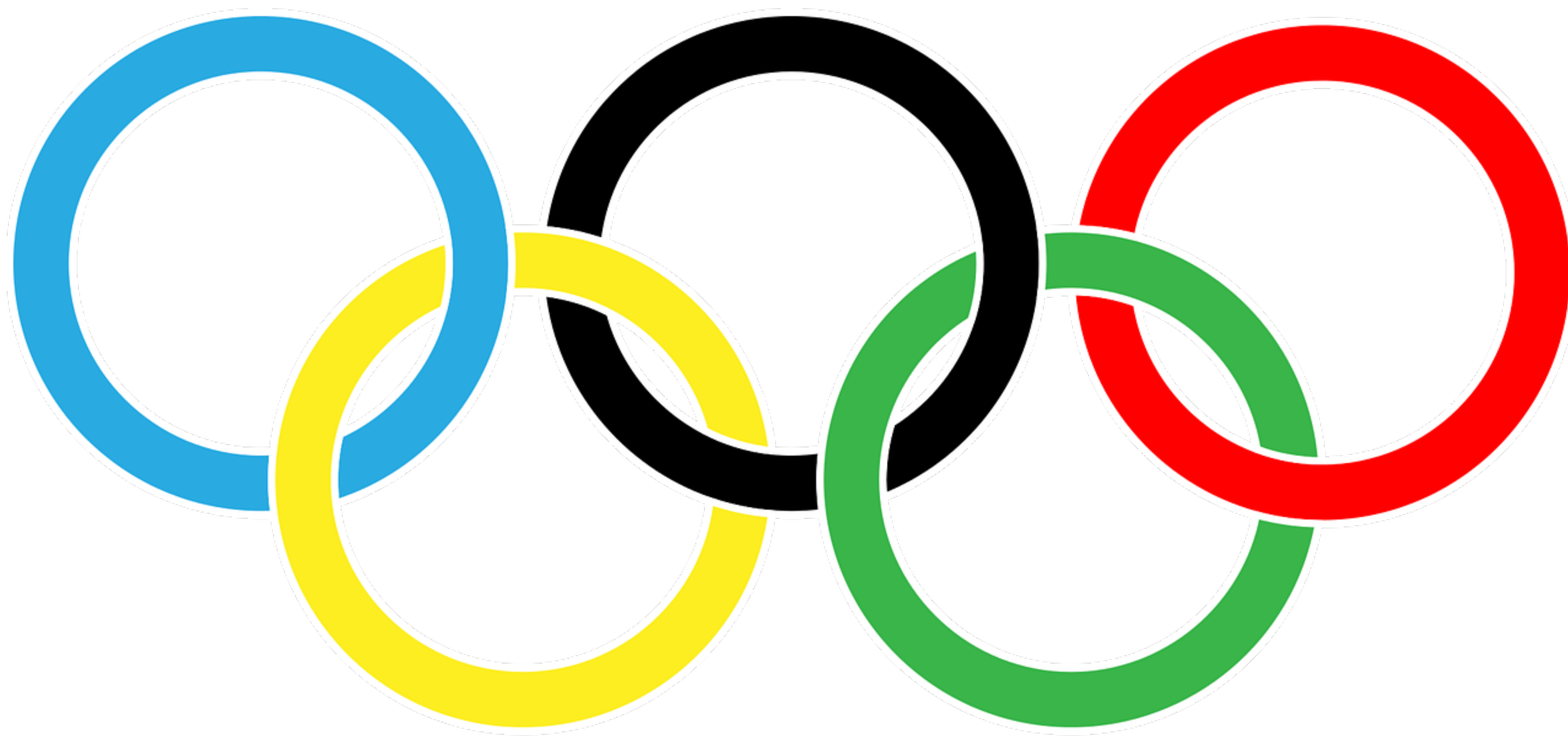


MERGING DATAFRAMES WITH PANDAS

# Medals in the Summer Olympics



# Does a host country win more medals?





# Summer Olympic medalists 1896 to 2008 - IOC COUNTRY CODES.csv

Country	NOC	ISO code
Afghanistan	AFG	AF
Albania	ALB	AL
Algeria	ALG	DZ
American Samoa*	ASA	AS
Andorra	AND	AD
Angola	ANG	AO
Antigua and Barbuda	ANT	AG
Argentina	ARG	AR
Armenia	ARM	AM
Aruba*	ARU	AW
Australia	AUS	AU
Austria	AUT	AT





## Summer Olympic medalists 1896 to 2008 - EDITIONS.tsv

Edition	Bronze	Gold	Silver	Grand Total	City	Country
1896	40	64	47	151	Athens	Greece
1900	142	178	192	512	Paris	France
1904	123	188	159	470	St. Louis	United States
1908	211	311	282	804	London	United Kingdom
1912	284	301	300	885	Stockholm	Sweden
1920	355	497	446	1298	Antwerp	Belgium
1924	285	301	298	884	Paris	France
1928	242	229	239	710	Amsterdam	Netherlands
1932	196	213	206	615	Los Angeles	United States
1936	282	299	294	875	Berlin	Germany
1948	268	276	270	814	London	United Kingdom
1952	299	300	290	889	Helsinki	Finland





# summer\_1896.csv, summer\_1900.csv, ..., summer\_2008.csv

Sport	Discipline	Athlete	NOC	Gender	Event	Event_gender	Medal
Aquatics	Diving	XIAO, Hailiang	CHN	Men	10m platform	M	Bronze
Aquatics	Diving	SAUTIN, Dmitry	RUS	Men	10m platform	M	Gold
Aquatics	Diving	HEMPEL, Jan	GER	Men	10m platform	M	Silver
Aquatics	Diving	CLARK, Mary Ellen	USA	Women	10m platform	W	Bronze
Aquatics	Diving	FU, Mingxia	CHN	Women	10m platform	W	Gold
Aquatics	Diving	WALTER, Annika	GER	Women	10m platform	W	Silver
Aquatics	Diving	LENZI, Mark Edward	USA	Men	3m springboard	M	Bronze
Aquatics	Diving	XIONG, Ni	CHN	Men	3m springboard	M	Gold
Aquatics	Diving	YU, Zhuocheng	CHN	Men	3m springboard	M	Silver
Aquatics	Diving	PELLETIER, Annie	CAN	Women	3m springboard	W	Bronze
Aquatics	Diving	FU, Mingxia	CHN	Women	3m springboard	W	Gold
Aquatics	Diving	LASHKO, Irina	RUS	Women	3m springboard	W	Silver

# Reminder: loading & merging files

- `pd.read_csv()` (& its many options)
- Looping over files, e.g.,
  - `[pd.read_csv(f) for f in glob('*.csv']`
- Concatenating & appending, e.g.,
  - `pd.concat([df1, df2], axis=0)`
  - `df1.append(df2)`



MERGING DATAFRAMES WITH PANDAS

**Let's practice!**





MERGING DATAFRAMES WITH PANDAS

# Quantifying performance



# Medals DataFrame

	Sport	Discipline	Athlete	NOC	Gender	Event	Event_gender	Medal	Edition
0	Aquatics	Swimming	HAJOS, Alfred	HUN	Men	100m freestyle	M	Gold	1896
1	Aquatics	Swimming	HERSCHMANN, Otto	AUT	Men	100m freestyle	M	Silver	1896
2	Aquatics	Swimming	DRIVAS, Dimitrios	GRE	Men	100m freestyle for sailors	M	Bronze	1896
3	Aquatics	Swimming	MALOKINIS, Ioannis	GRE	Men	100m freestyle for sailors	M	Gold	1896
4	Aquatics	Swimming	CHASAPIS, Spiridon	GRE	Men	100m freestyle for sailors	M	Silver	1896
5	Aquatics	Swimming	CHOROPHAS, Efstathios	GRE	Men	1200m freestyle	M	Bronze	1896
6	Aquatics	Swimming	HAJOS, Alfred	HUN	Men	1200m freestyle	M	Gold	1896
7	Aquatics	Swimming	ANDREOU, Joannis	GRE	Men	1200m freestyle	M	Silver	1896
8	Aquatics	Swimming	CHOROPHAS, Efstathios	GRE	Men	400m freestyle	M	Bronze	1896
9	Aquatics	Swimming	NEUMANN, Paul	AUT	Men	400m freestyle	M	Gold	1896
10	Aquatics	Swimming	PEPANOS, Antonios	GRE	Men	400m freestyle	M	Silver	1896
11	Athletics	Athletics	LANE, Francis	USA	Men	100m	M	Bronze	1896



# Constructing a pivot table

- Apply DataFrame `pivot_table()` method
  - `index`: column to use as index of pivot table
  - `values`: column(s) to aggregate
  - `aggfunc`: function to apply for aggregation
  - `columns`: categories as columns of pivot table





# Constructing a pivot table

NOC	AFG	AHO	ALG	ANZ	ARG	ARM	AUS	AUT	AZE	BAH	...	URS	URU	USA	UZB	VEN	VIE	YUG	ZAM	ZIM	ZZX
Edition																					
1896	NaN	NaN	NaN	NaN	NaN	NaN	2.0	5.0	NaN	NaN	...	NaN	NaN	20.0	NaN	NaN	NaN	NaN	NaN	NaN	6.0
1900	NaN	NaN	NaN	NaN	NaN	NaN	5.0	6.0	NaN	NaN	...	NaN	NaN	55.0	NaN	NaN	NaN	NaN	NaN	NaN	34.0
1904	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN	...	NaN	NaN	394.0	NaN	NaN	NaN	NaN	NaN	NaN	8.0
1908	NaN	NaN	NaN	19.0	NaN	NaN	NaN	1.0	NaN	NaN	...	NaN	NaN	63.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1912	NaN	NaN	NaN	10.0	NaN	NaN	NaN	14.0	NaN	NaN	...	NaN	NaN	101.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1920	NaN	NaN	NaN	NaN	NaN	NaN	6.0	NaN	NaN	NaN	...	NaN	NaN	193.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1924	NaN	NaN	NaN	NaN	11.0	NaN	10.0	4.0	NaN	NaN	...	NaN	22.0	198.0	NaN	NaN	NaN	2.0	NaN	NaN	NaN
1928	NaN	NaN	NaN	NaN	32.0	NaN	4.0	4.0	NaN	NaN	...	NaN	22.0	84.0	NaN	NaN	NaN	12.0	NaN	NaN	NaN
1932	NaN	NaN	NaN	NaN	4.0	NaN	5.0	5.0	NaN	NaN	...	NaN	1.0	181.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1936	NaN	NaN	NaN	NaN	11.0	NaN	1.0	50.0	NaN	NaN	...	NaN	NaN	92.0	NaN	NaN	NaN	1.0	NaN	NaN	NaN
1948	NaN	NaN	NaN	NaN	12.0	NaN	16.0	4.0	NaN	NaN	...	NaN	3.0	148.0	NaN	NaN	NaN	16.0	NaN	NaN	NaN
1952	NaN	NaN	NaN	NaN	6.0	NaN	20.0	3.0	NaN	NaN	...	117.0	14.0	130.0	NaN	1.0	NaN	24.0	NaN	NaN	NaN





# Computing fractions

NOC	AFG	AHO	ALG	ANZ	ARG	ARM	AUS	AUT	AZE	BAH	...	URS	URU	USA	UZB
Edition															
1896	NaN	NaN	NaN	NaN	NaN	NaN	0.013245	0.033113	NaN	NaN	...	NaN	NaN	0.132450	NaN
1900	NaN	NaN	NaN	NaN	NaN	NaN	0.009766	0.011719	NaN	NaN	...	NaN	NaN	0.107422	NaN
1904	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.002128	NaN	NaN	...	NaN	NaN	0.838298	NaN
1908	NaN	NaN	NaN	0.023632	NaN	NaN	NaN	0.001244	NaN	NaN	...	NaN	NaN	0.078358	NaN
1912	NaN	NaN	NaN	0.011299	NaN	NaN	NaN	0.015819	NaN	NaN	...	NaN	NaN	0.114124	NaN
1920	NaN	NaN	NaN	NaN	NaN	NaN	0.004622	NaN	NaN	NaN	...	NaN	NaN	0.148690	NaN
1924	NaN	NaN	NaN	NaN	0.012443	NaN	0.011312	0.004525	NaN	NaN	...	NaN	0.024887	0.223982	NaN
1928	NaN	NaN	NaN	NaN	0.045070	NaN	0.005634	0.005634	NaN	NaN	...	NaN	0.030986	0.118310	NaN
1932	NaN	NaN	NaN	NaN	0.006504	NaN	0.008130	0.008130	NaN	NaN	...	NaN	0.001626	0.294309	NaN
1936	NaN	NaN	NaN	NaN	0.012571	NaN	0.001143	0.057143	NaN	NaN	...	NaN	NaN	0.105143	NaN



MERGING DATAFRAMES WITH PANDAS

**Let's practice!**



MERGING DATAFRAMES WITH PANDAS

# Reshaping and plotting





# Reshaping the data

NOC	AFG	AHO	ALG	ANZ	ARG	ARM	...	VEN	VIE	YUG	ZAM	ZIM	ZZX
Edition													
1896	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
1900	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	33.561198
1904	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	-22.642384
1908	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	0.000000
1912	NaN	NaN	NaN	-26.092774	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	0.000000
1920	NaN	NaN	NaN	0.000000	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	0.000000
1924	NaN	NaN	NaN	0.000000	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	0.000000
1928	NaN	NaN	NaN	0.000000	131.101152	NaN	...	NaN	NaN	323.521127	NaN	NaN	0.000000
1932	NaN	NaN	NaN	0.000000	-25.794206	NaN	...	NaN	NaN	0.000000	NaN	NaN	0.000000
1936	NaN	NaN	NaN	0.000000	-10.271982	NaN	...	NaN	NaN	-29.357594	NaN	NaN	0.000000
1948	NaN	NaN	NaN	0.000000	-4.601500	NaN	...	NaN	NaN	47.596769	NaN	NaN	0.000000
1952	NaN	NaN	NaN	0.000000	-10.508545	NaN	...	NaN	NaN	34.043608	NaN	NaN	0.000000



	Edition	NOC	Change
0	1896	AFG	NaN
1	1900	AFG	NaN
2	1904	AFG	NaN
3	1908	AFG	NaN
4	1912	AFG	NaN
5	1920	AFG	NaN
6	1924	AFG	NaN
7	1928	AFG	NaN
8	1932	AFG	NaN
9	1936	AFG	NaN
10	1948	AFG	NaN
11	1952	AFG	NaN





# Host country data

	Edition	Bronze	Gold	Silver	Grand Total	City	Country	Host_NOC
0	1896	40	64	47	151	Athens	Greece	GRE
1	1900	142	178	192	512	Paris	France	FRA
2	1904	123	188	159	470	St. Louis	United States	USA
3	1908	211	311	282	804	London	United Kingdom	GBR
4	1912	284	301	300	885	Stockholm	Sweden	SWE
5	1920	355	497	446	1298	Antwerp	Belgium	BEL
6	1924	285	301	298	884	Paris	France	FRA
7	1928	242	229	239	710	Amsterdam	Netherlands	NED
8	1932	196	213	206	615	Los Angeles	United States	USA
9	1936	282	299	294	875	Berlin	Germany	GER
10	1948	268	276	270	814	London	United Kingdom	GBR
11	1952	299	300	290	889	Helsinki	Finland	FIN

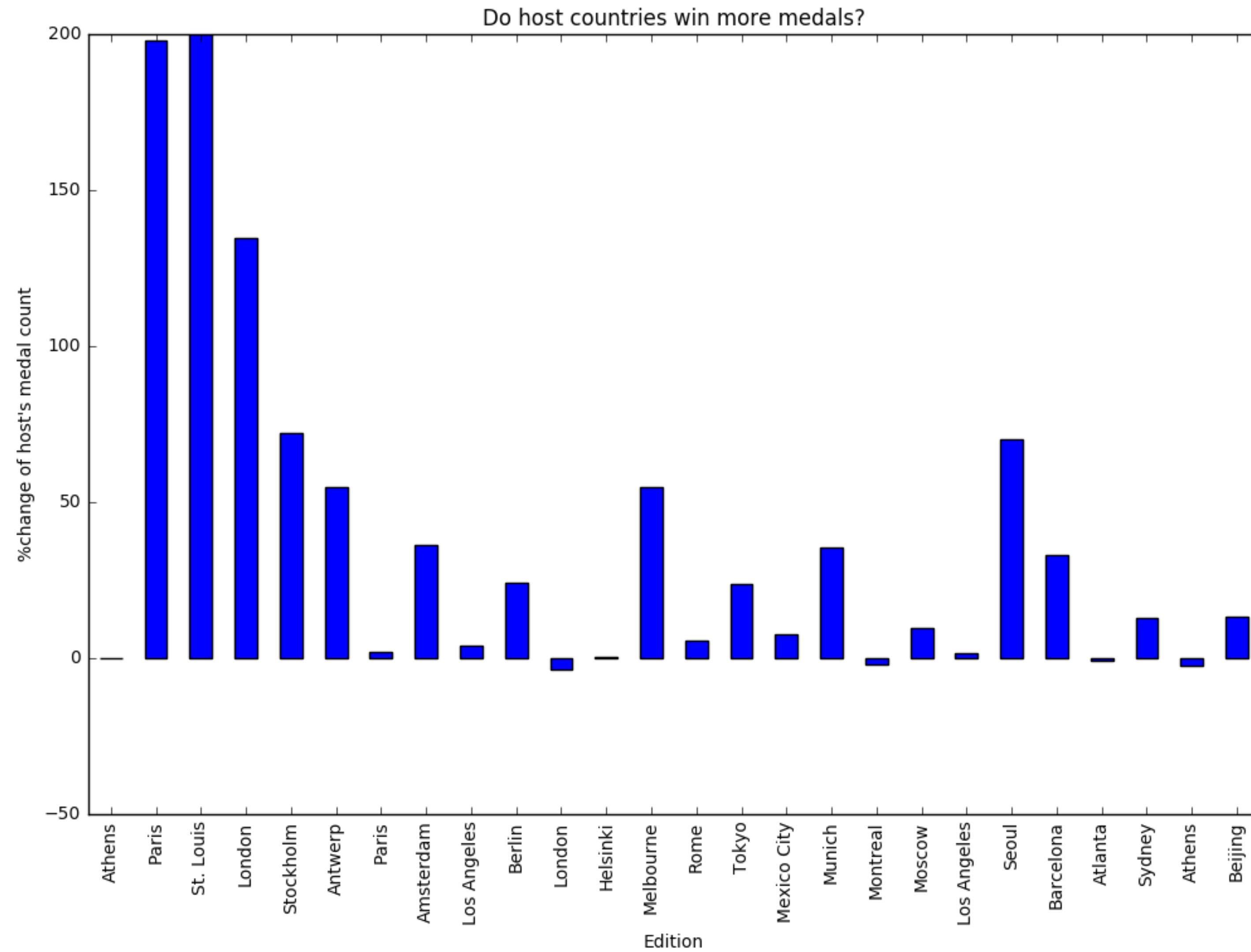


# Quantifying influence

	NOC	count	Host_NOC	Grand Total	fraction	change
Edition						
1896	GRE	52	GRE	151	0.344371	NaN
1900	FRA	185	FRA	512	0.361328	198.002486
1904	USA	394	USA	470	0.838298	199.651245
1908	GBR	347	GBR	804	0.431592	134.489218
1912	SWE	173	SWE	885	0.195480	71.896226
1920	BEL	188	BEL	1298	0.144838	54.757887
1924	FRA	122	FRA	884	0.138009	2.046362
1928	NED	65	NED	710	0.091549	36.315243
1932	USA	181	USA	615	0.294309	3.739184
1936	GER	210	GER	875	0.240000	24.108011
1948	GBR	56	GBR	814	0.068796	-3.635059
1952	FIN	40	FIN	889	0.044994	0.121662



# Graphical summary





MERGING DATAFRAMES WITH PANDAS

**Let's practice!**





MERGING DATAFRAMES WITH PANDAS

# Final thoughts