Pandas - Merge & Join

- Reference: Data Science Handbook

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

In [2]:
```
class display(object):
    """Display HTML representation of multiple objects"""
    template = """<div style="float: left; padding: 10px;">
    <p style='font-family:"Courier New", Courier, monospace'>{0}</p>{1}
    </div>"""
    def __init__(self, *args):
        self.args = args

    def _repr_html_(self):
        return '\n'.join(self.template.format(a, eval(a)._repr_html_())
                         for a in self.args)

    def __repr__(self):
        return '\n\n'.join(a + '\n' + repr(eval(a))
                           for a in self.args)
```

One-to-one joins

In [3]:
```
df1 = pd.DataFrame({'employee': ['Bob', 'Jake', 'Lisa', 'Sue'],
                    'group': ['Accounting', 'Engineering', 'Engineering', 'HR']})

df2 = pd.DataFrame({'employee': ['Lisa', 'Bob', 'Jake', 'Sue'],
                    'hire_date': [2004, 2008, 2012, 2014]})

display('df1', 'df2')
```

Out[3]:

df1

	employee	group
0	Bob	Accounting
1	Jake	Engineering
2	Lisa	Engineering
3	Sue	HR

df2

	employee	hire_date
0	Lisa	2004
1	Bob	2008
2	Jake	2012
3	Sue	2014

In [4]:
```
# default - inner merge (intersection of keys from both frames)

df3 = pd.merge(df1, df2)

display('df3')
```

Out[4]:

df3

	employee	group	hire_date
0	Bob	Accounting	2008
1	Jake	Engineering	2012
2	Lisa	Engineering	2004
3	Sue	HR	2014

In [5]:
```
pd.merge(df1, df2, how='inner', on='employee')
```

Out[5]:

	employee	group	hire_date
0	Bob	Accounting	2008
1	Jake	Engineering	2012
2	Lisa	Engineering	2004
3	Sue	HR	2014

Many-to-one joins

- one of the two key columns contains duplicate entries

In [6]:
```
df4 = pd.DataFrame({'group': ['Accounting', 'Engineering', 'HR'],
                    'supervisor': ['Carly', 'Guido', 'Steve']})

display('df3', 'df4',
        'pd.merge(df3, df4)')
```

Out[6]:

df3

	employee	group	hire_date
0	Bob	Accounting	2008
1	Jake	Engineering	2012
2	Lisa	Engineering	2004
3	Sue	HR	2014

df4

	group	supervisor
0	Accounting	Carly
1	Engineering	Guido
2	HR	Steve

pd.merge(df3, df4)

	employee	group	hire_date	supervisor
0	Bob	Accounting	2008	Carly
1	Jake	Engineering	2012	Guido
2	Lisa	Engineering	2004	Guido
3	Sue	HR	2014	Steve

Many-to-many joins

- key column in both the left and right contains duplicates

In [7]:
```
df5 = pd.DataFrame({'group': ['Accounting', 'Accounting',
                              'Engineering', 'Engineering', 'HR', 'HR'],
                    'skills': ['math', 'spreadsheets', 'coding', 'linux',
                               'spreadsheets', 'organization']})

display('df1', 'df5',
        "pd.merge(df1, df5)")
```

Out[7]:

df1

	employee	group
0	Bob	Accounting
1	Jake	Engineering
2	Lisa	Engineering
3	Sue	HR

df5

	group	skills
0	Accounting	math
1	Accounting	spreadsheets
2	Engineering	coding
3	Engineering	linux
4	HR	spreadsheets
5	HR	organization

pd.merge(df1, df5)

	employee	group	skills
0	Bob	Accounting	math
1	Bob	Accounting	spreadsheets
2	Jake	Engineering	coding
3	Jake	Engineering	linux
4	Lisa	Engineering	coding
5	Lisa	Engineering	linux
6	Sue	HR	spreadsheets
7	Sue	HR	organization

Specification of the Merge Key

In [8]:
```
# on keyword
# left and right data frames have the same column name

display('df1', 'df2',
        "pd.merge(df1, df2, on='employee')")
```

Out[8]:

df1

	employee	group
0	Bob	Accounting
1	Jake	Engineering
2	Lisa	Engineering
3	Sue	HR

df2

	employee	hire_date
0	Lisa	2004
1	Bob	2008
2	Jake	2012
3	Sue	2014

pd.merge(df1, df2, on='employee')

	employee	group	hire_date
0	Bob	Accounting	2008
1	Jake	Engineering	2012
2	Lisa	Engineering	2004
3	Sue	HR	2014

In [9]:
```
# left_on and right_on keywords

df3 = pd.DataFrame({'name': ['Bob', 'Jake', 'Lisa', 'Sue'],
                    'salary': [70000, 80000, 120000, 90000]})

display('df1', 'df3',
        'pd.merge(df1, df3, left_on="employee", right_on="name")')
```

Out[9]:

df1

	employee	group
0	Bob	Accounting
1	Jake	Engineering
2	Lisa	Engineering
3	Sue	HR

df3

	name	salary
0	Bob	70000
1	Jake	80000
2	Lisa	120000
3	Sue	90000

pd.merge(df1, df3, left_on="employee", right_on="name")

	employee	group	name	salary
0	Bob	Accounting	Bob	70000
1	Jake	Engineering	Jake	80000
2	Lisa	Engineering	Lisa	120000
3	Sue	HR	Sue	90000

In [10]:
```
pd.merge(df1, df3,
         left_on="employee",
         right_on="name").drop('name', axis=1)
```

Out[10]:

	employee	group	salary
0	Bob	Accounting	70000
1	Jake	Engineering	80000
2	Lisa	Engineering	120000
3	Sue	HR	90000

Merge on an index

In [11]:
```
df1a = df1.set_index('employee')
df2a = df2.set_index('employee')

display('df1a', 'df2a')
```

Out[11]:

df1a

employee	group
Bob	Accounting
Jake	Engineering
Lisa	Engineering
Sue	HR

df2a

employee	hire_date
Lisa	2004
Bob	2008
Jake	2012
Sue	2014

In [12]:
```
display('df1a', 'df2a',
    "pd.merge(df1a, df2a, left_index=True, right_index=True)")
```

Out[12]:

df1a

employee	group
Bob	Accounting
Jake	Engineering
Lisa	Engineering
Sue	HR

df2a

employee	hire_date
Lisa	2004
Bob	2008
Jake	2012
Sue	2014

pd.merge(df1a, df2a, left_index=True, right_index=True)

employee	group	hire_date
Bob	Accounting	2008
Jake	Engineering	2012
Lisa	Engineering	2004
Sue	HR	2014

- join() method, which performs a merge that defaults to joining on indices

In [13]:
```
display('df1a', 'df2a',
        'df1a.join(df2a)')
```

Out[13]:

df1a

employee	group
Bob	Accounting
Jake	Engineering
Lisa	Engineering
Sue	HR

df2a

employee	hire_date
Lisa	2004
Bob	2008
Jake	2012
Sue	2014

df1a.join(df2a)

employee	group	hire_date
Bob	Accounting	2008
Jake	Engineering	2012
Lisa	Engineering	2004
Sue	HR	2014

Mix indices and columns

In [14]:
```
display('df1a', 'df2',
        'pd.merge(df1a, df2, left_index=True, right_on="employee")')
```

Out[14]:

df1a

employee	group
Bob	Accounting
Jake	Engineering
Lisa	Engineering
Sue	HR

df2

	employee	hire_date
0	Lisa	2004
1	Bob	2008
2	Jake	2012
3	Sue	2014

pd.merge(df1a, df2, left_index=True, right_on="employee")

	group	employee	hire_date
1	Accounting	Bob	2008
2	Engineering	Jake	2012
0	Engineering	Lisa	2004
3	HR	Sue	2014

In [15]:
```
display('df1a', 'df3',
        "pd.merge(df1a, df3, left_index=True, right_on='name')")
```

Out[15]:

df1a

employee	group
Bob	Accounting
Jake	Engineering
Lisa	Engineering
Sue	HR

df3

	name	salary
0	Bob	70000
1	Jake	80000
2	Lisa	120000
3	Sue	90000

pd.merge(df1a, df3, left_index=True, right_on='name')

	group	name	salary
0	Accounting	Bob	70000
1	Engineering	Jake	80000
2	Engineering	Lisa	120000
3	HR	Sue	90000

Specifying Set Arithmetic for Joins

- When a value appears in one key column but not in the other

In [16]:
```
df6 = pd.DataFrame({'name': ['Peter', 'Paul', 'Mary'],
                    'food': ['fish', 'beans', 'bread']},
                   columns=['name', 'food'])

df7 = pd.DataFrame({'name': ['Mary', 'Joseph'],
                    'drink': ['wine', 'beer']},
                   columns=['name', 'drink'])

display('df6', 'df7', 'pd.merge(df6, df7)')
```

Out[16]:

df6

	name	food
0	Peter	fish
1	Paul	beans
2	Mary	bread

df7

	name	drink
0	Mary	wine
1	Joseph	beer

pd.merge(df6, df7)

	name	food	drink
0	Mary	bread	wine

default - inner join

- intersection of the two sets of inputs

In [17]:
```
pd.merge(df6, df7, how='inner')
```

Out[17]:

	name	food	drink
0	Mary	bread	wine

outer join

- returns a join over the union of the input colums
- fills in all the missing values with NAs

In [18]:
```
display('df6', 'df7',
        'pd.merge(df6, df7, how="outer")')
```

Out[18]:

df6

	name	food
0	Peter	fish
1	Paul	beans
2	Mary	bread

df7

	name	drink
0	Mary	wine
1	Joseph	beer

pd.merge(df6, df7, how="outer")

	name	food	drink
0	Peter	fish	NaN
1	Paul	beans	NaN
2	Mary	bread	wine
3	Joseph	NaN	beer

left join

- uses only keys from left frame

In [19]:
```
display('df6', 'df7',
        "pd.merge(df6, df7, how='left')")
```

Out[19]:

df6

	name	food
0	Peter	fish
1	Paul	beans
2	Mary	bread

df7

	name	drink
0	Mary	wine
1	Joseph	beer

pd.merge(df6, df7, how='left')

	name	food	drink
0	Peter	fish	NaN
1	Paul	beans	NaN
2	Mary	bread	wine

right join

- uses only keys from right frame

In [20]:
```
display('df6', 'df7',
        "pd.merge(df6, df7, how='right')")
```

Out[20]:

df6

	name	food
0	Peter	fish
1	Paul	beans
2	Mary	bread

df7

	name	drink
0	Mary	wine
1	Joseph	beer

pd.merge(df6, df7, how='right')

	name	food	drink
0	Mary	bread	wine
1	Joseph	NaN	beer

Overlapping Column Names: The suffixes Keyword

- When the output would have two conflicting column names, the merge function automatically appends a suffix `_x` or `_y` to make the output columns unique.

In [21]:
```
df8 = pd.DataFrame({'name': ['Bob', 'Jake', 'Lisa', 'Sue'],
                    'rank': [1, 2, 3, 4]})

df9 = pd.DataFrame({'name': ['Bob', 'Jake', 'Lisa', 'Sue'],
                    'rank': [3, 1, 4, 2]})

display('df8', 'df9', 'pd.merge(df8, df9, on="name")')
```

Out[21]:

df8

	name	rank
0	Bob	1
1	Jake	2
2	Lisa	3
3	Sue	4

df9

	name	rank
0	Bob	3
1	Jake	1
2	Lisa	4
3	Sue	2

pd.merge(df8, df9, on="name")

	name	rank_x	rank_y
0	Bob	1	3
1	Jake	2	1
2	Lisa	3	4
3	Sue	4	2

- specify custom suffixes

In [22]:
```
df8 = pd.DataFrame({'name': ['Bob', 'Jake', 'Lisa', 'Sue'],
                    'rank': [1, 2, 3, 4]})

df9 = pd.DataFrame({'name': ['Bob', 'Jake', 'Lisa', 'Sue'],
                    'rank': [3, 1, 4, 2]})

display('df8', 'df9',
        'pd.merge(df8, df9, on="name", suffixes=["_L", "_R"])')
```

Out[22]:

df8

	name	rank
0	Bob	1
1	Jake	2
2	Lisa	3
3	Sue	4

df9

	name	rank
0	Bob	3
1	Jake	1
2	Lisa	4
3	Sue	2

pd.merge(df8, df9, on="name", suffixes=["_L", "_R"])

	name	rank_L	rank_R
0	Bob	1	3
1	Jake	2	1
2	Lisa	3	4
3	Sue	4	2

Example: US States Data

- Rank states by their total population density for the year 2012

In [23]:
```
pop = pd.read_csv('http://people.bu.edu/kalathur/datasets/state-population.csv')
areas = pd.read_csv('http://people.bu.edu/kalathur/datasets/state-areas.csv')
abbrevs = pd.read_csv('http://people.bu.edu/kalathur/datasets/state-abbrevs.csv')

display('pop.head()', 'areas.head()', 'abbrevs.head()')
```

Out[23]:

pop.head()

	state/region	ages	year	population
0	AL	under18	2012	1117489.0
1	AL	total	2012	4817528.0
2	AL	under18	2010	1130966.0
3	AL	total	2010	4785570.0
4	AL	under18	2011	1125763.0

areas.head()

	state	area (sq. mi)
0	Alabama	52423
1	Alaska	656425
2	Arizona	114006
3	Arkansas	53182
4	California	163707

abbrevs.head()

	state	abbreviation
0	Alabama	AL
1	Alaska	AK
2	Arizona	AZ
3	Arkansas	AR
4	California	CA

In [24]:
```
areas.state.unique()
```

Out[24]:
```
array(['Alabama', 'Alaska', 'Arizona', 'Arkansas', 'California',
       'Colorado', 'Connecticut', 'Delaware', 'Florida', 'Georgia',
       'Hawaii', 'Idaho', 'Illinois', 'Indiana', 'Iowa', 'Kansas',
       'Kentucky', 'Louisiana', 'Maine', 'Maryland', 'Massachusetts',
       'Michigan', 'Minnesota', 'Mississippi', 'Missouri', 'Montana',
       'Nebraska', 'Nevada', 'New Hampshire', 'New Jersey', 'New Mexico',
       'New York', 'North Carolina', 'North Dakota', 'Ohio', 'Oklahoma',
       'Oregon', 'Pennsylvania', 'Rhode Island', 'South Carolina',
       'South Dakota', 'Tennessee', 'Texas', 'Utah', 'Vermont',
       'Virginia', 'Washington', 'West Virginia', 'Wisconsin', 'Wyoming',
       'District of Columbia', 'Puerto Rico'], dtype=object)
```

In [25]:
```
abbrevs.state.unique()
```

Out[25]:
```
array(['Alabama', 'Alaska', 'Arizona', 'Arkansas', 'California',
       'Colorado', 'Connecticut', 'Delaware', 'District of Columbia',
       'Florida', 'Georgia', 'Hawaii', 'Idaho', 'Illinois', 'Indiana',
       'Iowa', 'Kansas', 'Kentucky', 'Louisiana', 'Maine', 'Montana',
       'Nebraska', 'Nevada', 'New Hampshire', 'New Jersey', 'New Mexico',
       'New York', 'North Carolina', 'North Dakota', 'Ohio', 'Oklahoma',
       'Oregon', 'Maryland', 'Massachusetts', 'Michigan', 'Minnesota',
       'Mississippi', 'Missouri', 'Pennsylvania', 'Rhode Island',
       'South Carolina', 'South Dakota', 'Tennessee', 'Texas', 'Utah',
       'Vermont', 'Virginia', 'Washington', 'West Virginia', 'Wisconsin',
       'Wyoming'], dtype=object)
```

In [26]:
```
pop['state/region'].unique()
```

Out[26]:
```
array(['AL', 'AK', 'AZ', 'AR', 'CA', 'CO', 'CT', 'DE', 'DC', 'FL', 'GA',
       'HI', 'ID', 'IL', 'IN', 'IA', 'KS', 'KY', 'LA', 'ME', 'MD', 'MA',
       'MI', 'MN', 'MS', 'MO', 'MT', 'NE', 'NV', 'NH', 'NJ', 'NM', 'NY',
       'NC', 'ND', 'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX',
       'UT', 'VT', 'VA', 'WA', 'WV', 'WI', 'WY', 'PR', 'USA'],
      dtype=object)
```

In [27]:
```
merged = pd.merge(pop, abbrevs, left_on='state/region', right_on='abbreviation')
merged.head()
```

Out[27]:

	state/region	ages	year	population	state	abbreviation
0	AL	under18	2012	1117489.0	Alabama	AL
1	AL	total	2012	4817528.0	Alabama	AL
2	AL	under18	2010	1130966.0	Alabama	AL
3	AL	total	2010	4785570.0	Alabama	AL
4	AL	under18	2011	1125763.0	Alabama	AL

In [28]:
```
merged = merged.drop('abbreviation', axis=1)
merged.head()
```

Out[28]:

	state/region	ages	year	population	state
0	AL	under18	2012	1117489.0	Alabama
1	AL	total	2012	4817528.0	Alabama
2	AL	under18	2010	1130966.0	Alabama
3	AL	total	2010	4785570.0	Alabama
4	AL	under18	2011	1125763.0	Alabama

In [29]:
```
merged.isnull().any()
```

Out[29]:
```
state/region    False
ages            False
year            False
population      False
state           False
dtype: bool
```

In [30]:
```
merged = pd.merge(merged, areas, on='state', how='left')
merged.head(20)
```

Out[30]:

	state/region	ages	year	population	state	area (sq. mi)
0	AL	under18	2012	1117489.0	Alabama	52423
1	AL	total	2012	4817528.0	Alabama	52423
2	AL	under18	2010	1130966.0	Alabama	52423
3	AL	total	2010	4785570.0	Alabama	52423
4	AL	under18	2011	1125763.0	Alabama	52423
5	AL	total	2011	4801627.0	Alabama	52423
6	AL	total	2009	4757938.0	Alabama	52423
7	AL	under18	2009	1134192.0	Alabama	52423
8	AL	under18	2013	1111481.0	Alabama	52423
9	AL	total	2013	4833722.0	Alabama	52423
10	AL	total	2007	4672840.0	Alabama	52423
11	AL	under18	2007	1132296.0	Alabama	52423
12	AL	total	2008	4718206.0	Alabama	52423
13	AL	under18	2008	1134927.0	Alabama	52423
14	AL	total	2005	4569805.0	Alabama	52423
15	AL	under18	2005	1117229.0	Alabama	52423
16	AL	total	2006	4628981.0	Alabama	52423
17	AL	under18	2006	1126798.0	Alabama	52423
18	AL	total	2004	4530729.0	Alabama	52423
19	AL	under18	2004	1113662.0	Alabama	52423

In [31]:
```
merged.isnull().any()
```

Out[31]:
```
state/region     False
ages             False
year             False
population       False
state            False
area (sq. mi)    False
dtype: bool
```

In [32]:
```
merged.year.unique()
```

Out[32]:
```
array([2012, 2010, 2011, 2009, 2013, 2007, 2008, 2005, 2006, 2004, 2003,
       2001, 2002, 1999, 2000, 1998, 1997, 1996, 1995, 1994, 1993, 1992,
       1991, 1990])
```

In [33]:
```
merged.ages.unique()
```

Out[33]:
```
array(['under18', 'total'], dtype=object)
```

In [ ]:

In [34]:
```
data_2012 = merged.query("year == 2012 & ages == 'total'")

data_2012.head()
```

Out[34]:

	state/region	ages	year	population	state	area (sq. mi)
1	AL	total	2012	4817528.0	Alabama	52423
95	AK	total	2012	730307.0	Alaska	656425
97	AZ	total	2012	6551149.0	Arizona	114006
191	AR	total	2012	2949828.0	Arkansas	53182
193	CA	total	2012	37999878.0	California	163707

In [35]:
```
data_2012.set_index('state', inplace=True)

data_2012.head()
```

Out[35]:

state	state/region	ages	year	population	area (sq. mi)
Alabama	AL	total	2012	4817528.0	52423
Alaska	AK	total	2012	730307.0	656425
Arizona	AZ	total	2012	6551149.0	114006
Arkansas	AR	total	2012	2949828.0	53182
California	CA	total	2012	37999878.0	163707

In [36]:
```
density = data_2012['population'] / data_2012['area (sq. mi)']

density.sort_values(ascending=False, inplace=True)
```

In [37]:
```
density.head()
```

Out[37]:
```
state
District of Columbia    9315.102941
New Jersey              1016.710502
Rhode Island             679.808414
Connecticut              647.865260
Massachusetts            629.588157
dtype: float64
```

In [38]:
```
density.tail()
```

Out[38]:
```
state
South Dakota    10.814785
North Dakota     9.919453
Montana          6.837955
Wyoming          5.894886
Alaska           1.112552
dtype: float64
```

In [ ]:
```

```

Concat and Append

In [39]:
```
def make_df(cols, ind):
    data = {c: [str(c) + str(i) for i in ind]
            for c in cols}
    return pd.DataFrame(data, ind)
```

In [40]:
```
# example DataFrame
make_df('ABC', range(5))
```

Out[40]:

	A	B	C
0	A0	B0	C0
1	A1	B1	C1
2	A2	B2	C2
3	A3	B3	C3
4	A4	B4	C4

In [41]:
```
df1 = make_df('AB', [1, 2, 3])
df2 = make_df('AB', [4, 5])

display('df1', 'df2', 'pd.concat([df1, df2])')

# default concatenation is row wise (axis = 0)
```

Out[41]:

df1

	A	B
1	A1	B1
2	A2	B2
3	A3	B3

df2

	A	B
4	A4	B4
5	A5	B5

pd.concat([df1, df2])

	A	B
1	A1	B1
2	A2	B2
3	A3	B3
4	A4	B4
5	A5	B5

In [42]:
```
df1 = make_df('AB', [1, 2])
df2 = make_df('CD', [1, 2])


display('df1', 'df2', 'pd.concat([df1, df2], axis=1)')
```

Out[42]:

df1

	A	B
1	A1	B1
2	A2	B2

df2

	C	D
1	C1	D1
2	C2	D2

pd.concat([df1, df2], axis=1)

	A	B	C	D
1	A1	B1	C1	D1
2	A2	B2	C2	D2

Duplicate Indices

- Pandas preserves indices with concatenation

In [43]:
```
df1 = make_df('AB', [1, 2])
df2 = make_df('AB', [3, 4])
df2.index = df1.index

display('df1', 'df2', 'pd.concat([df1, df2])')
```

Out[43]:

df1

	A	B
1	A1	B1
2	A2	B2

df2

	A	B
1	A3	B3
2	A4	B4

pd.concat([df1, df2])

	A	B
1	A1	B1
2	A2	B2
1	A3	B3
2	A4	B4

In [44]:
```
df3 = pd.concat([df1, df2])
df3.loc[1]
```

Out[44]:

	A	B
1	A1	B1
1	A3	B3

In [45]:
```
display('df1', 'df2', 'pd.concat([df1, df2], ignore_index=True)')
```

Out[45]:

df1

	A	B
1	A1	B1
2	A2	B2

df2

	A	B
1	A3	B3
2	A4	B4

pd.concat([df1, df2], ignore_index=True)

	A	B
0	A1	B1
1	A2	B2
2	A3	B3
3	A4	B4

Concatenation with joins

In [46]:
```
df5 = make_df('ABC', [1, 2])
df6 = make_df('BCD', [3, 4])

display('df5', 'df6', 'pd.concat([df5, df6], sort=False)')
```

Out[46]:

df5

	A	B	C
1	A1	B1	C1
2	A2	B2	C2

df6

	B	C	D
3	B3	C3	D3
4	B4	C4	D4

pd.concat([df5, df6], sort=False)

	A	B	C	D
1	A1	B1	C1	NaN
2	A2	B2	C2	NaN
3	NaN	B3	C3	D3
4	NaN	B4	C4	D4

In [47]:
```
# Default join is outer, the union of the input columns
```

In [48]:
```
display('df5', 'df6', 'pd.concat([df5, df6], join="inner")')
```

Out[48]:

df5

	A	B	C
1	A1	B1	C1
2	A2	B2	C2

df6

	B	C	D
3	B3	C3	D3
4	B4	C4	D4

pd.concat([df5, df6], join="inner")

	B	C
1	B1	C1
2	B2	C2
3	B3	C3
4	B4	C4

append()

- Data frame method for concatenation

In [49]:
```
df1 = make_df('AB', [1, 2])
df2 = make_df('AB', [3, 4])
df2.index = df1.index

display('df1', 'df2', 'df1.append(df2)')
```

Out[49]:

df1

	A	B
1	A1	B1
2	A2	B2

df2

	A	B
1	A3	B3
2	A4	B4

df1.append(df2)

	A	B
1	A1	B1
2	A2	B2
1	A3	B3
2	A4	B4

In [50]:
```
display('df1', 'df2', 'df1.append(df2, ignore_index=True)')
```

Out[50]:

df1

	A	B
1	A1	B1
2	A2	B2

df2

	A	B
1	A3	B3
2	A4	B4

df1.append(df2, ignore_index=True)

	A	B
0	A1	B1
1	A2	B2
2	A3	B3
3	A4	B4

In [51]:
```
df5 = make_df('ABC', [1, 2])
df6 = make_df('BCD', [3, 4])

display('df5', 'df6', 'df5.append(df6, sort=True)')
```

Out[51]:

df5

	A	B	C
1	A1	B1	C1
2	A2	B2	C2

df6

	B	C	D
3	B3	C3	D3
4	B4	C4	D4

df5.append(df6, sort=True)

	A	B	C	D
1	A1	B1	C1	NaN
2	A2	B2	C2	NaN
3	NaN	B3	C3	D3
4	NaN	B4	C4	D4

In [52]:
```
display('df6', 'df5', 'df6.append(df5, sort=False)')
```

Out[52]:

df6

	B	C	D
3	B3	C3	D3
4	B4	C4	D4

df5

	A	B	C
1	A1	B1	C1
2	A2	B2	C2

df6.append(df5, sort=False)

	B	C	D	A
3	B3	C3	D3	NaN
4	B4	C4	D4	NaN
1	B1	C1	NaN	A1
2	B2	C2	NaN	A2

query()

In [ ]:
```
```

In [53]:
```
np.random.seed(231)

df = pd.DataFrame(np.random.randn(10, 2), columns=list('ab'))
df
```

Out[53]:

	a	b
0	0.417943	1.397100
1	-1.785904	-0.708828
2	-0.074725	-0.775017
3	-0.149798	1.861729
4	-1.425529	-0.376357
5	-0.342275	0.294908
6	-0.837324	0.952188
7	1.329317	0.524652
8	-0.148100	0.889532
9	0.124447	0.991093

In [54]:
```
df[df['a'] > df['b']]

# or

df[df.a > df.b]
```

Out[54]:

	a	b
2	-0.074725	-0.775017
7	1.329317	0.524652

In [55]:
```
df.query('a > b')
```

Out[55]:

	a	b
2	-0.074725	-0.775017
7	1.329317	0.524652

In [56]:
```
df.eval('a > b')
```

Out[56]:
```
0    False
1    False
2     True
3    False
4    False
5    False
6    False
7     True
8    False
9    False
dtype: bool
```

In [57]:
```
df[df.eval('a > b')]
```

Out[57]:

	a	b
2	-0.074725	-0.775017
7	1.329317	0.524652

In [58]:
```
df
```

Out[58]:

	a	b
0	0.417943	1.397100
1	-1.785904	-0.708828
2	-0.074725	-0.775017
3	-0.149798	1.861729
4	-1.425529	-0.376357
5	-0.342275	0.294908
6	-0.837324	0.952188
7	1.329317	0.524652
8	-0.148100	0.889532
9	0.124447	0.991093

In [59]:
```
df.query('a > b', inplace=True)
```

In [60]:
```
df
```

Out[60]:

	a	b
2	-0.074725	-0.775017
7	1.329317	0.524652

In [ ]:
```

```

eval()

- uses string expressions to efficiently compute operations using DataFrames

In [61]:
```
nrows, ncols = 10000, 10
rng = np.random.RandomState(42)
df1, df2, df3, df4 = (pd.DataFrame(rng.rand(nrows, ncols))
                      for i in range(4))
```

In [62]:
```
df1.head()
```

Out[62]:

	0	1	2	3	4	5	6	7	8	9
0	0.374540	0.950714	0.731994	0.598658	0.156019	0.155995	0.058084	0.866176	0.601115	0.708073
1	0.020584	0.969910	0.832443	0.212339	0.181825	0.183405	0.304242	0.524756	0.431945	0.291229
2	0.611853	0.139494	0.292145	0.366362	0.456070	0.785176	0.199674	0.514234	0.592415	0.046450
3	0.607545	0.170524	0.065052	0.948886	0.965632	0.808397	0.304614	0.097672	0.684233	0.440152
4	0.122038	0.495177	0.034389	0.909320	0.258780	0.662522	0.311711	0.520068	0.546710	0.184854

In [63]:
```
np.allclose(df1 + df2 + df3 + df4,
            pd.eval('df1 + df2 + df3 + df4'))
```

Out[63]:
```
True
```

In [64]:
```
df1, df2, df3, df4, df5 = (pd.DataFrame(rng.randint(0, 1000, (100, 3)))
                           for i in range(5))

df1.head()
```

Out[64]:

	0	1	2
0	411	124	78
1	241	41	151
2	70	745	355
3	550	0	327
4	544	30	555

In [65]:
```
result1 = -df1 * df2 / (df3 + df4) - df5

result2 = pd.eval('-df1 * df2 / (df3 + df4) - df5')

np.allclose(result1, result2)
```

Out[65]: True

In [66]:
```
# pd.eval() supports all comparison operators, including chained expressions

result1 = (df1 < df2) & (df2 <= df3) & (df3 != df4)

result2 = pd.eval('df1 < df2 <= df3 != df4')

np.allclose(result1, result2)
```

Out[66]: True

In [67]:
```
df = pd.DataFrame(rng.rand(1000, 3), columns=['A', 'B', 'C'])
df.head()
```

Out[67]:

	A	B	C
0	0.261220	0.022824	0.820763
1	0.126501	0.074407	0.392106
2	0.188662	0.214910	0.362007
3	0.153719	0.397591	0.074359
4	0.316478	0.640667	0.406242

In [68]:
```
result1 = (df['A'] + df['B']) / (df['C'] - 1)

result2 = pd.eval("(df.A + df.B) / (df.C - 1)")

np.allclose(result1, result2)
```

Out[68]: True

In [69]:
```
# Using DataFrame.eval

result3 = df.eval('(A + B) / (C - 1)')

np.allclose(result1, result3)
```

Out[69]:
```
True
```

In [70]:
```
# Assignment of new column

df.eval('D = (A + B) / C', inplace=True)

df.head()
```

Out[70]:

	A	B	C	D
0	0.261220	0.022824	0.820763	0.346073
1	0.126501	0.074407	0.392106	0.512381
2	0.188662	0.214910	0.362007	1.114818
3	0.153719	0.397591	0.074359	7.414149
4	0.316478	0.640667	0.406242	2.356098

In [71]:
```
# Modification of existing column

df.eval('D = (A - B) / C', inplace=True)

df.head()
```

Out[71]:

	A	B	C	D
0	0.261220	0.022824	0.820763	0.290456
1	0.126501	0.074407	0.392106	0.132858
2	0.188662	0.214910	0.362007	-0.072509
3	0.153719	0.397591	0.074359	-3.279654
4	0.316478	0.640667	0.406242	-0.798022

In [72]:
```
df.mean(axis=1).head()
```

Out[72]:
```
0    0.348816
1    0.181468
2    0.173268
3   -0.663496
4    0.141341
dtype: float64
```

In [73]:
```
# Local variables

column_mean = df.mean(1)

result1 = df['A'] + column_mean

result2 = df.eval('A + @column_mean')

np.allclose(result1, result2)
```

Out[73]:
```
True
```

In [ ]:

In [ ]: