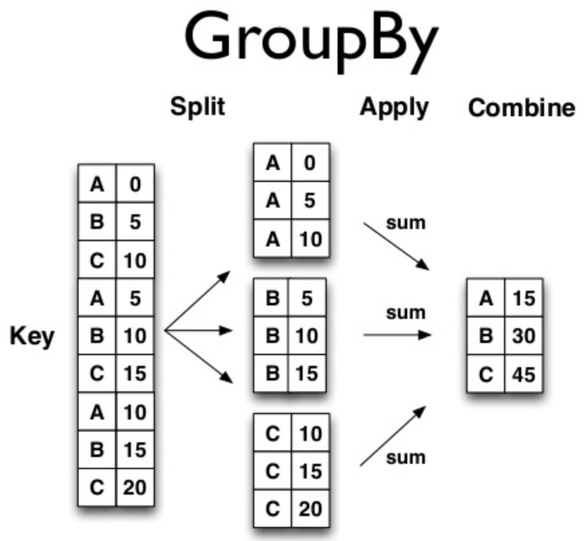
The *GroupBy* Pattern

The *GroupBy* Pattern

We have seen the ***GroupBy*** operator in ***Pandas***, but this is actually a more general ***design pattern*** that can be utilized in many data analyics frameworks and data access interfaces, e.g. in ***SQL***.

GroupBy: general Pattern



GroupBy in SQL:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;
```

GroupBy in MongoDB

```
db.BusinessProcess.aggregate({
    "$group": {
        _id: {
            status: "$status",
            type: "$type"
        },
        count: {
            $sum: 1
        }
    }
})
```

In [2]:
```
#setup example
import numpy as np
import pandas as pd
df = pd.DataFrame({'key1' : ['a', 'a', 'b', 'b', 'a'],
                   'key2' : ['one', 'two', 'one', 'two', 'one'],
                   'data1' : np.random.randn(5),
                   'data2' : np.random.randn(5)})
df
```

Out[2]:

	key1	key2	data1	data2
0	a	one	-1.116082	-0.618130
1	a	two	0.994237	0.514594
2	b	one	-0.161336	0.316494
3	b	two	-0.681649	0.755114
4	a	one	-1.446345	1.505941

In [3]:
```
#group by key1
grouped = df.groupby(df['key1'])
grouped #this is now a more complex group object
```

Out[3]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fc1dcc67310>
```

In [4]:
```
#and generates a table per group
for name, group in grouped:
    print ("name:", name, "\n",group)
```

```
name: a
   key1 key2     data1     data2
0    a  one -1.116082 -0.618130
1    a  two  0.994237  0.514594
4    a  one -1.446345  1.505941
name: b
   key1 key2     data1     data2
2    b  one -0.161336  0.316494
3    b  two -0.681649  0.755114
```

In [6]:
```
#access group table
grouped.get_group('b')
```

Out[6]:

	key1	key2	data1	data2
2	b	one	-0.161336	0.316494
3	b	two	-0.681649	0.755114

In [7]:
```
#get numper of entries (rows) per group
grouped.size()
```

Out[7]:
```
key1
a    3
b    2
dtype: int64
```

In [8]:
```
#get number of group entries by columns
grouped.count()
```

Out[8]:

	key2	data1	data2
key1			
a	3	3	3
b	2	2	2

Think of grouped DataFrames as 3d objects:

In [9]:
```
#accessing the "3d" group tables
grouped['data2'].get_group('a')
```

Out[9]:
```
0   -0.618130
1    0.514594
4    1.505941
Name: data2, dtype: float64
```

In [10]:
```
grouped.get_group('a')['data2']
```

Out[10]:
```
0   -0.618130
1    0.514594
4    1.505941
Name: data2, dtype: float64
```

Group by external keys

Group by external keys

In [11]:
```
#define external key years as numpy array
years = np.array([2005, 2005, 2006, 2005, 2006])
df['data1'].groupby([years]).mean()
```

Out[11]:
```
2005   -0.267831
2006   -0.803840
Name: data1, dtype: float64
```

Group by functions

Group by functions

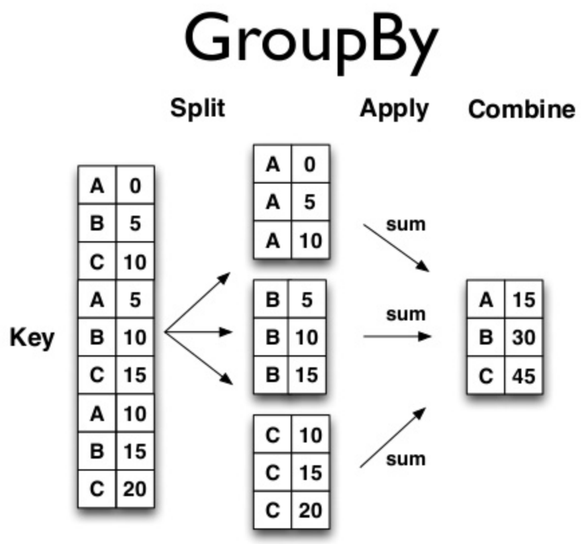
In [12]:
```
#sort by column and retun top n
def top(df, n=5, column='data1'):
    return df.sort_values(by=column)[-n:]

df.groupby(df['key1']).apply(top,n=5)
```

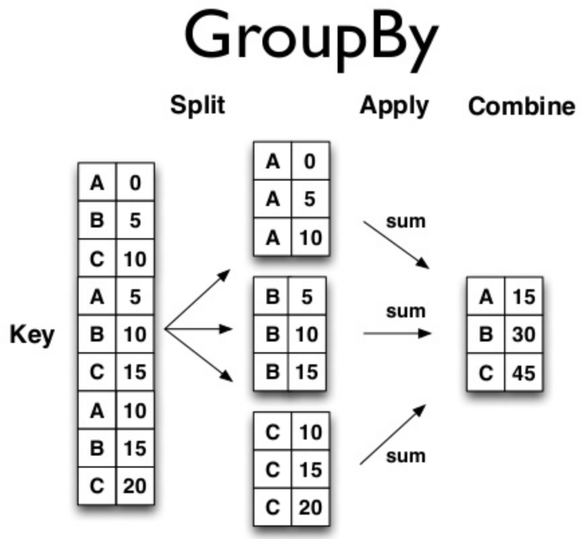
Out[12]:

		key1	key2	data1	data2
key1					
a	**4**	a	one	-1.446345	1.505941
	0	a	one	-1.116082	-0.618130
	1	a	two	0.994237	0.514594
b	**3**	b	two	-0.681649	0.755114
	2	b	one	-0.161336	0.316494

Group-wise aggregation (apply)

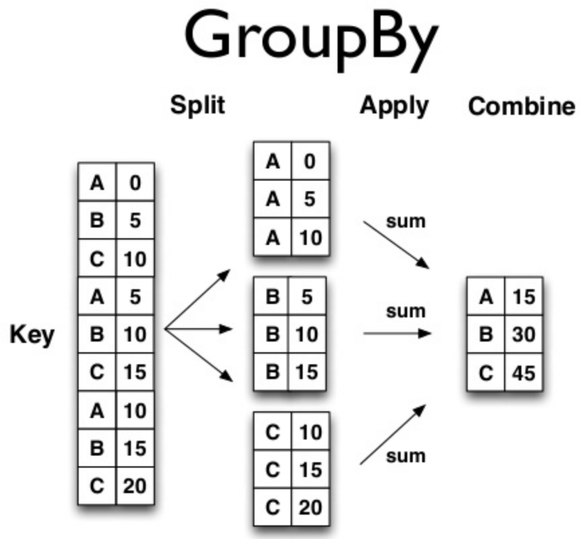


Group-wise aggregation (apply)



Typical build in aggregation functions:

Group-wise aggregation (apply)



Typical build in aggregation functions:

In [13]:
```
grouped.sum()
```

Out[13]:

	data1	data2
key1		
a	-1.568190	1.402405
b	-0.842985	1.071608

Custom Aggregation Functions

In [14]:
```
def peak_to_peak(arr):
    return arr.max() - arr.min()
grouped.agg(peak_to_peak)
```

Out[14]:

	data1	data2
key1		
a	2.440582	2.12407
b	0.520313	0.43862

Multiple aggregations

In [15]:
```
#just call a list of function
grouped.agg([peak_to_peak, 'mean', 'median'])
```

Out[15]:

	data1			data2		
	peak_to_peak	mean	median	peak_to_peak	mean	median
key1						
a	2.440582	-0.522730	-1.116082	2.12407	0.467468	0.514594
b	0.520313	-0.421492	-0.421492	0.43862	0.535804	0.535804

Suppressing the Group Keys

In [16]:
```
df.groupby(df['key1']).apply(top,n=2)
```

Out[16]:

		key1	key2	data1	data2
key1					
a	**0**	a	one	-1.116082	-0.618130
	1	a	two	0.994237	0.514594
b	**3**	b	two	-0.681649	0.755114
	2	b	one	-0.161336	0.316494

In [17]:
```
df.groupby(df['key1'], group_keys=False).apply(top,n=2)
```

Out[17]:

	key1	key2	data1	data2
0	a	one	-1.116082	-0.618130
1	a	two	0.994237	0.514594
3	b	two	-0.681649	0.755114
2	b	one	-0.161336	0.316494

More Exercises in the Lab session...