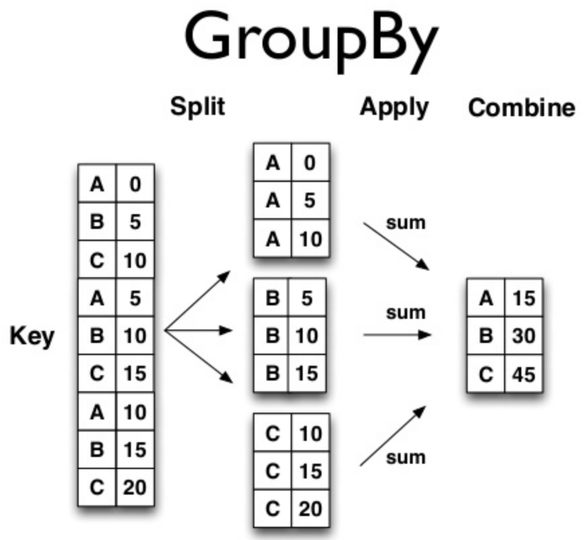
The *GroupBy* Pattern

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 [1]:
```
#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[1]:

	key1	key2	data1	data2
0	a	one	-2.246449	0.256898
1	a	two	-0.872856	-0.768457
2	b	one	2.017690	1.027994
3	b	two	-0.633363	0.217852
4	a	one	0.289376	-0.717704

In [2]:
```
#group by key1
grouped = df.groupby(df['key1'])
grouped #this is now a more complex group object
```

Out[2]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fc9641db210>
```

In [2]:
```
#group by key1
grouped = df.groupby(df['key1'])
grouped #this is now a more complex group object
```

Out[2]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fc9641db210>
```

In [3]:
```
#group by creates a group for every unique key
df['key1'].unique()
```

Out[3]:
```
array(['a', 'b'], dtype=object)
```

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 -2.246449  0.256898
1    a  two -0.872856 -0.768457
4    a  one  0.289376 -0.717704
name: b
   key1 key2     data1     data2
2    b  one  2.017690  1.027994
3    b  two -0.633363  0.217852
```

In [5]:
```
#access group table
grouped.get_group('a')
```

Out[5]:

	key1	key2	data1	data2
0	a	one	-2.246449	0.256898
1	a	two	-0.872856	-0.768457
4	a	one	0.289376	-0.717704

In [6]:
```
#get numper of entries (rows) per group
grouped.size()
```

Out[6]:
```
key1
a    3
b    2
dtype: int64
```

In [7]:
```
#get number of group entries by columns
grouped.count()
```

Out[7]:

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

Think of grouped DataFrames as 3d objects:

In [8]:
```
#accessing the "3d" group tables
grouped['data2'].get_group('a')
```

Out[8]:
```
0    0.256898
1   -0.768457
4   -0.717704
Name: data2, dtype: float64
```

In [9]:
```
grouped.get_group('a')['data2']
```

Out[9]:
```
0    0.256898
1   -0.768457
4   -0.717704
Name: data2, dtype: float64
```

Group by external keys

Group by external keys

In [10]:
```
#define external key years as numpy array
years = np.array([2005, 2005, 2006, 2005, 2006])
df['data1'].groupby([years]).mean()
```

Out[10]:
```
2005   -1.250890
2006    1.153533
Name: data1, dtype: float64
```

Group by functions

Group by functions

In [11]:
```
#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[11]:

		key1	key2	data1	data2
key1					
a	**0**	a	one	-2.246449	0.256898
	1	a	two	-0.872856	-0.768457
	4	a	one	0.289376	-0.717704
b	**3**	b	two	-0.633363	0.217852
	2	b	one	2.017690	1.027994

Group-wise aggregation

Group-wise aggregation

Typical build in aggregation functions:

- sum
- mean
- max / min
- quantile
- ...

Group-wise aggregation

Typical build in aggregation functions:

- sum
- mean
- max / min
- quantile
- ...

In [12]:
```
#aggregate over the groups
grouped.mean()
```

Out[12]:

	data1	data2
key1		
a	-0.943310	-0.409754
b	0.692163	0.622923

Custom Aggregation Functions

In [13]:
```
def peak_to_peak(arr):
    return arr.max() - arr.min()
grouped.agg(peak_to_peak)
```

Out[13]:

	data1	data2
key1		
a	2.535826	1.025355
b	2.651054	0.810142

Multiple aggregations

In [14]:
```
#just call a list of function
grouped.agg([peak_to_peak, 'mean', 'median'])
```

Out[14]:

	data1			data2		
	peak_to_peak	mean	median	peak_to_peak	mean	median
key1						
a	2.535826	-0.943310	-0.872856	1.025355	-0.409754	-0.717704
b	2.651054	0.692163	0.692163	0.810142	0.622923	0.622923

Suppressing the Group Keys

In [15]:
```
df.groupby(df['key1']).apply(top,n=2)
```

Out[15]:

		key1	key2	data1	data2
key1					
a	**1**	a	two	-0.872856	-0.768457
	4	a	one	0.289376	-0.717704
b	**3**	b	two	-0.633363	0.217852
	2	b	one	2.017690	1.027994

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

Out[16]:

	key1	key2	data1	data2
1	a	two	-0.872856	-0.768457
4	a	one	0.289376	-0.717704
3	b	two	-0.633363	0.217852
2	b	one	2.017690	1.027994

More Exercises in the Lab session...