Introduction to Pandas* (in Tulip):

*Panel Data Structures

Jocelyn Forest, Benjamin Renoust, Guy Mélançon,
LaBRI, Université Bordeaux I and CNRS UMR 5800

What is pandas?

- Recent API based on Numpy
- Devised by Wes McKinney
- Fast and intuitive data structures
- Easy to work with messy and irregularly indexed data
- Optimized for performance, with critical code paths compiled to C
- Adopts concepts of R language

Main focus

- The two basics structures of pandas
  - Series 1d array
  - DataFrame 2d array
  - Panel nd array (n>2)
- Filtering, selecting data
- Aggregating, transforming data
- Joining, concatenating, merging data
- Descriptive basics statistics

Installing pandas

- Version python 2.6 or 2.7
- Dependencies:
  - NumPy 1.6.1 or higher
- Optional dependencies:
  - Matplotlib to plot
  - SciPy for statistical functions

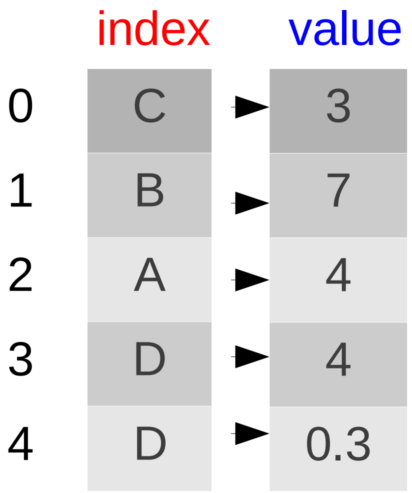
Exercise

-
  ```
  > sudo apt-get install python-pandas
  ```

-
  ```
  > git clone git://github.com/pydata/pandas.git
  > cd pandas
  > python setup.py install
  ```
- Header:

  ```
  import pandas as pd
  ```

Series

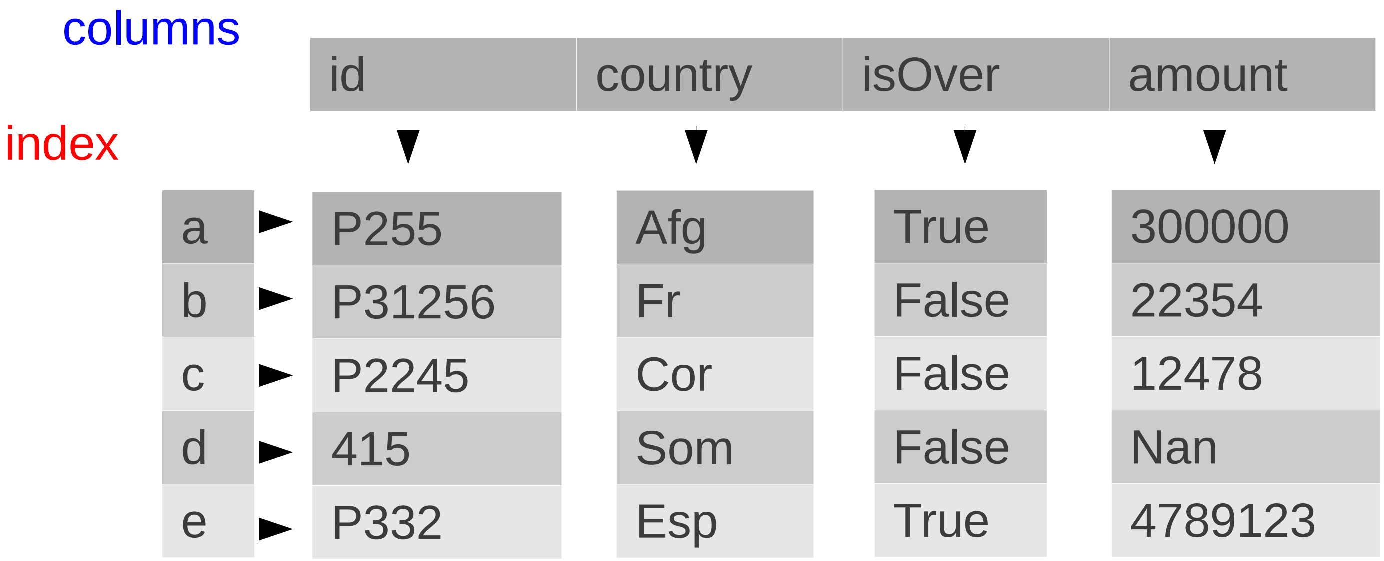


	index	value
0	C	3
1	B	7
2	A	4
3	D	4
4	D	0.3

- Subclass from numpy.ndarray
- Any type of data (numeric, string, boolean...)
- Index need not to be ordered
- Duplicated index are possible

Some vocabulary:
- `Series.index`: list of indices
- `Series.values`: list of values

DataFrame



columns

index

	id	country	isOver	amount
a	P255	Afg	True	300000
b	P31256	Fr	False	22354
c	P2245	Cor	False	12478
d	415	Som	False	Nan
e	P332	Esp	True	4789123

- ndarray-like
- 2D data structure (for *n*D data structures see Panel)
- Dictionary of series
- Row and column index
- Size mutable: insert or delete columns

DataFrame

- Some vocabulary
  - `DataFrame.index`: list of DataFrame indices
  - `DataFrame.values`: 2D array of all values contained in the DataFrame
  - `DataFrame.columns`: list of columns labels
  - axis: indicates the axis index for rows (axis = 0), columns (axis = 1),

    or even nth axis in panels

Panel

- Container for three or more dimensional data
- Dictionary of DataFrame objects
- Less used than Series and DataFrame objects
- DataFrame methods not all available for Panel objects
- **Unnecessary** in a lot of cases :

  → *Hierarchical indexing*

Construction of Series and DataFrame

Exercise

- Directly editing

```
 s = pd.Series([3,7,4,4,0.3] ,
              [index = ['a','b','c','d','e']])

df = pd.DataFrame(np.arange(9).reshape(3,3),
                   [index = ['b','a','c'],
                    columns=['Paris','Berlin','Madrid']])
```

- From a python dict

```
data = {'Paris': [0,3,6,999999999],'Berlin': [1,4,7], 'Madrid': [2,5,8]}

df = pd.DataFrame(data,
                 [index = ['b','a','c','d'],
                  Columns = ['Paris', 'Berlin', 'Madrid'] ])
```

Warning: index array size >= max element array size

- Several methods in the API to import from databases

```
df = pd.read_csv(path/fichier.csv,
               [index_col = [...]])

df = pd.read_table(path/fichier.txt,
                 [sep = ','])
```

Selection of data

- Selection on series

```
In:s
Out:
 a     3.0
 b     7.0
 c     4.0
 d     4.0
 d     0.3
```

```
In:s['b']
Out:
7.0
```

```
In:s['a':'c']
Out:
a     3.0
b     7.0
c     4.0
```

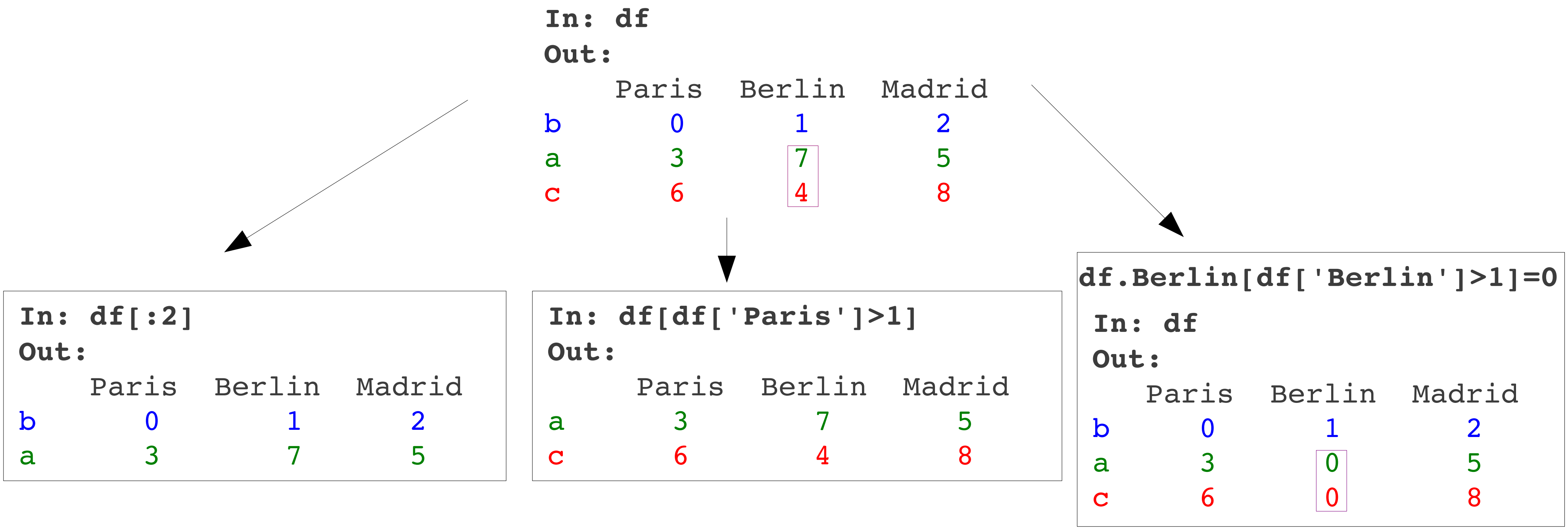
```
In:s['d']
Out:
d    4.0
d    0.3
```

```
In:s[1]
Out:
7.0
```

- The returned object is either a value, or a subset of the initial series ***s***
- Select some data with integer index OR index label
  - **Warning: Work only if the index type is not numeric**

Selection of data

- Filter on DataFrame



```
In: df
Out:
     Paris  Berlin  Madrid
b        0       1       2
a        3       7       5
c        6       4       8
```

```
In: df[:2]
Out:
     Paris  Berlin  Madrid
b        0       1       2
a        3       7       5
```

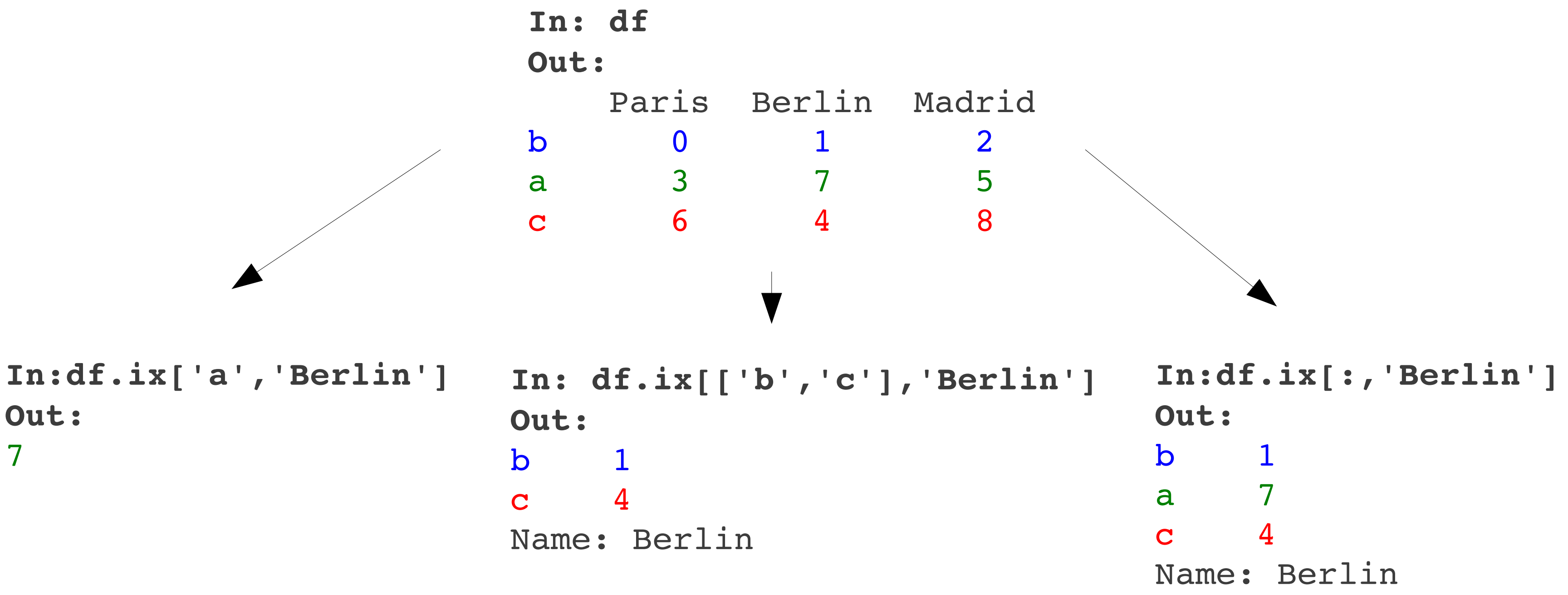
```
In: df[df['Paris']>1]
Out:
     Paris  Berlin  Madrid
a        3       7       5
c        6       4       8
```

```
df.Berlin[df['Berlin']>1]=0
 In: df
 Out:
     Paris  Berlin  Madrid
b        0       1       2
a        3       0       5
c        6       0       8
```

- Output Object: subset of the initial DataFrame

Selection of data

- The indexing field ***ix*** enables to select a subset of the rows and columns from a DataFrame.



```
In: df
Out:
    Paris  Berlin  Madrid
b       0       1       2
a       3       7       5
c       6       4       8
```

```
In:df.ix['a','Berlin']
Out:
7
```

```
In: df.ix[['b','c'],'Berlin']
Out:
b     1
c     4
Name: Berlin
```

```
In:df.ix[:,'Berlin']
Out:
b     1
a     7
c     4
Name: Berlin
```

- Output Object: a value OR a Series subset of the DataFrame

Exercise

Select the rows where `'Rank'=0`

Dropping entries from an axis

- On series or DataFrame, drop a row by his index

```
In: s
Out:
a     3.0
b     7.0
c     4.0
d     4.0
d     0.3
```

```
In: s.drop('d')
Out:
a     3.0
b     7.0
c     4.0
```

```
In: s.drop_duplicates()
Out:
a     3.0
b     7.0
c     4.0
d     0.3
```

- In DataFrame, (default) 'axis=0' refers to (row) index and axis=1 to columns

```
In: df
Out:
   Paris   Berlin   Madrid
b      0        1        2
a      3        7        5
c      6        4        8
```

```
In: df.drop('c')
Out:
   Paris   Berlin   Madrid
b      0        1        2
a      3        7        5
```

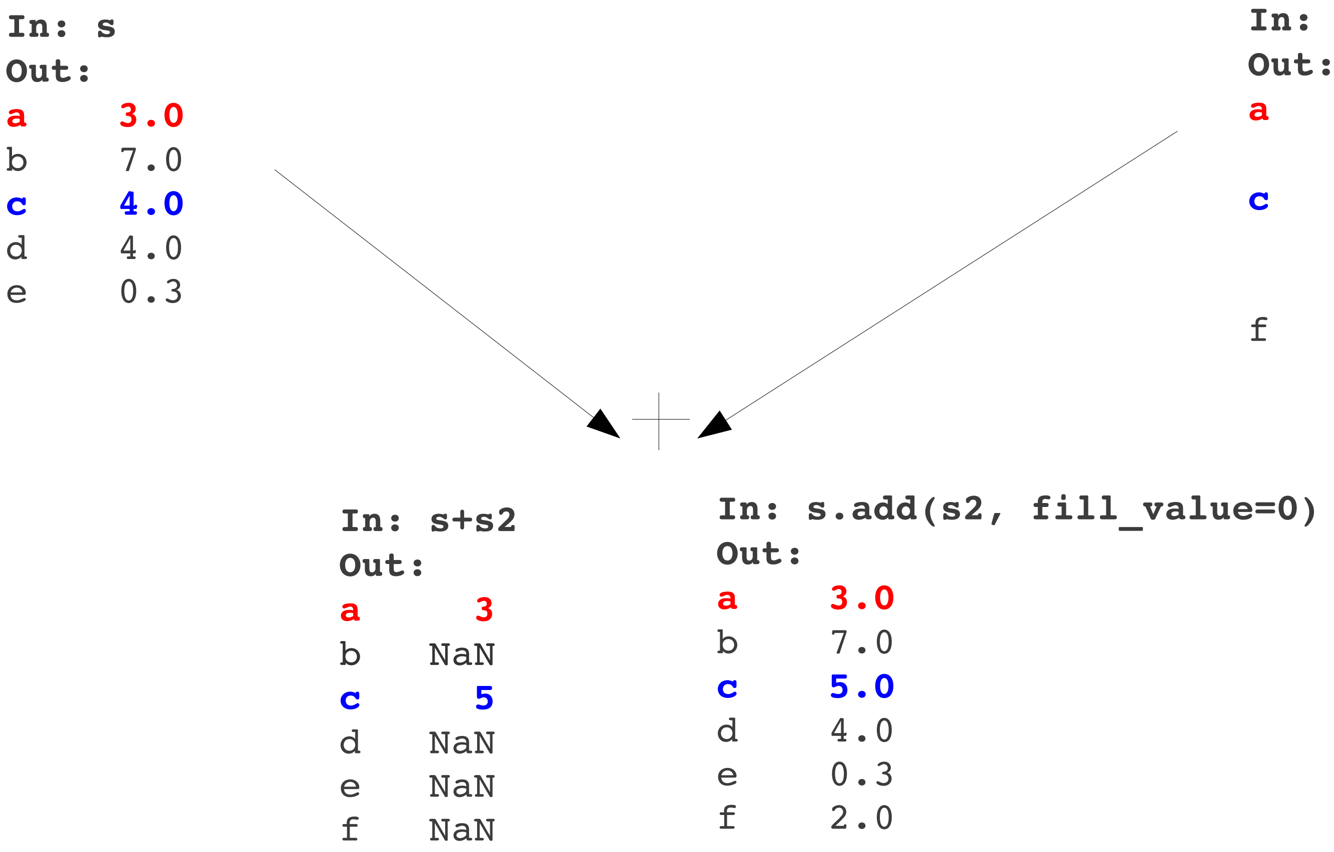
```
In :df.drop('Berlin', axis=1)
Out:
   Paris   Madrid
b      0        2
a      3        5
c      6        8
```

Exercise

Drop rows containing `'Rank' = 0`

Data alignment

- Series join and align axis to do operations



```
In: s
Out:
a     3.0
b     7.0
c     4.0
d     4.0
e     0.3
```

```
In: s2
Out:
a     0

c     1


f     2
```

```
In: s+s2
Out:
a      3
b    NaN
c      5
d    NaN
e    NaN
f    NaN
```

```
In: s.add(s2, fill_value=0)
Out:
a     3.0
b     7.0
c     5.0
d     4.0
e     0.3
f     2.0
```

Data alignment

- DataFrame join and align on both axes

```
In: df
Out:
    Paris  Berlin  Madrid
b       0       1       2
a       3       7       5
c       6       4       8
```

```
In:df2
Out:
     Paris  Lisbonne  Madrid
b        0         1       2
e        3         4       5
c        6         7       8
a        9        10      11
```

```
In: df+df2
Out:
    Berlin  Lisbonne  Madrid  Paris
a   NaN        NaN      16      12
b   NaN        NaN      4        0
c   NaN        NaN      16      12
e   NaN        NaN     NaN     NaN
```

```
In: df.add(df2, fill_value=0)
Out:
         Berlin  Lisbonne  Madrid    Paris
a          7        10       16        12
b          1         1        4         0
c          4         7       16        12
e         NaN        4        5         3
```

Exercise

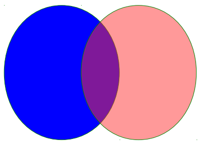
- Compute the total amount between the two DataFrame information ('Technical budget' and 'Amount')

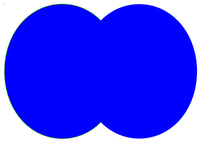
Merge, join, concatenate

- Many to one:

```
In: df1
Out:
   data1  keyLeft
0      0        b
1      1        b
2      2        a
3      3        c
4      4        a
5      5        a
6      6        b
```

```
In: df2
Out:
   data2  key
0      0    a
1      1    b
2      2    d
```



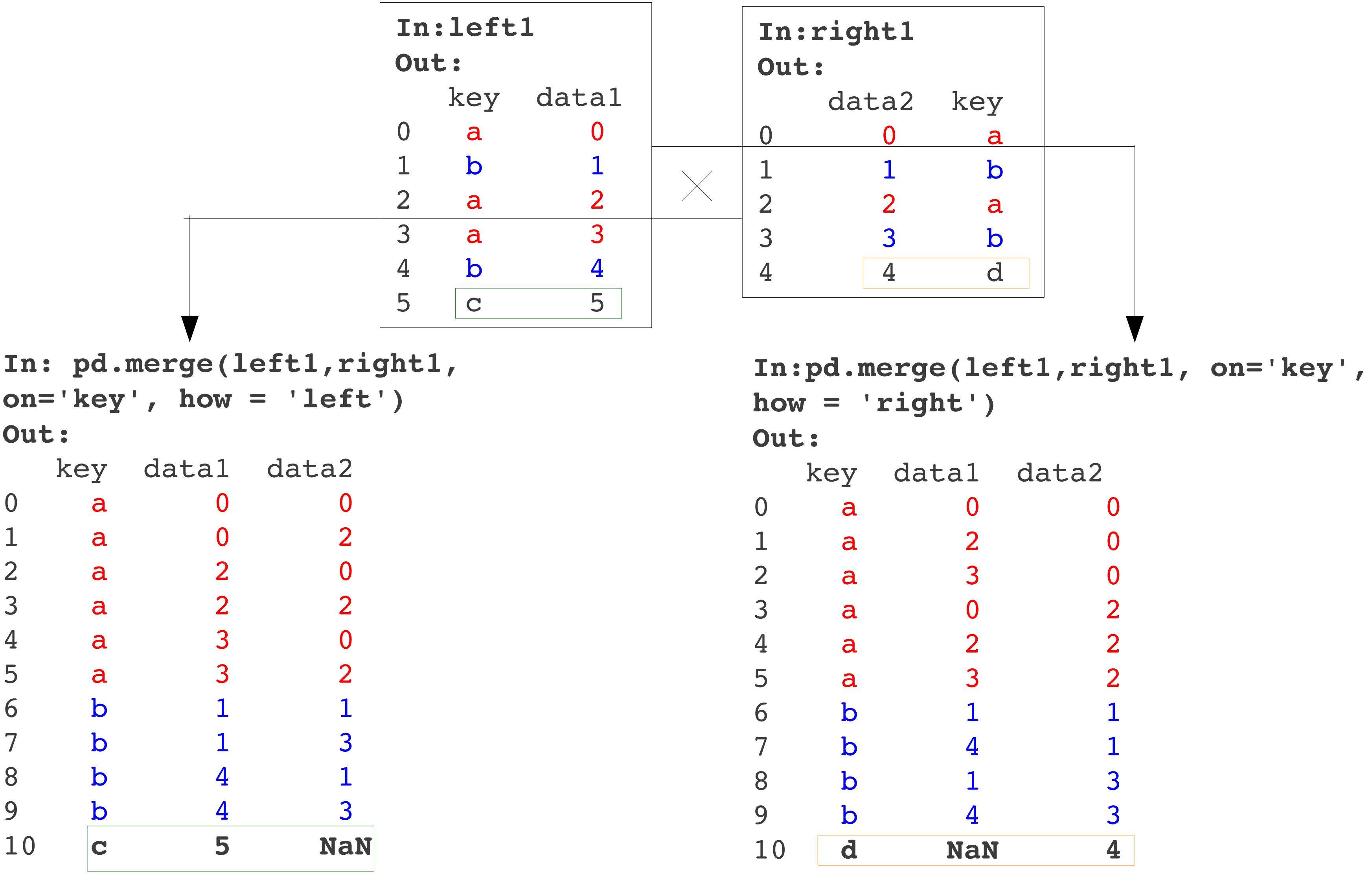


```
In: pd.merge(df1,df2, left_on = 'keyLeft',
right_on='key', how = 'inner')
Out:
   data1 keyLeft  data2  key
0      0       b      1    b
1      1       b      1    b
2      6       b      1    b
3      2       a      0    a
4      4       a      0    a
5      5       a      0    a
```

```
In:pd.merge(df1,df2, left_on =
'keyLeft', right_on='key', how =
'outer')
Out:
   data1    keyLeft      data2     key
0      0          b          1       b
1      1          b          1       b
2      6          b          1       b
3      2          a          0       a
4      4          a          0       a
5      5          a          0       a
6      3          c        NaN     NaN
7    NaN        NaN          2       d
```

Merge, join, concatenate

- Many to many: cartesian product of the rows given a common key



```
In:left1
Out:
    key  data1
0    a     0
1    b     1
2    a     2
3    a     3
4    b     4
5    c     5
```

```
In:right1
Out:
     data2   key
0      0      a
1      1      b
2      2      a
3      3      b
4      4      d
```

```
In: pd.merge(left1,right1,
on='key', how = 'left')
Out:
   key  data1  data2
0   a     0      0
1   a     0      2
2   a     2      0
3   a     2      2
4   a     3      0
5   a     3      2
6   b     1      1
7   b     1      3
8   b     4      1
9   b     4      3
10  c     5    NaN
```

```
In:pd.merge(left1,right1, on='key',
how = 'right')
Out:
   key  data1  data2
0   a     0      0
1   a     2      0
2   a     3      0
3   a     0      2
4   a     2      2
5   a     3      2
6   b     1      1
7   b     4      1
8   b     1      3
9   b     4      3
10  d    NaN     4
```

Merge, join, concatenate

Exercise

- Merge the two CSV among the keys [Id, Project] :

- Make the joint considering the intersection

Ranking

- Rank methods on Series and DataFrame among several methods

```
In: s
Out:
a    3.0
b    7.0
c    4.0
d    4.0
d    0.3
```

```
In: s.rank([ascending = True])
Out:
a    2.0
b    5.0
c    3.5
d    3.5
d    1.0
```

```
In: s.rank(method='first')
Out:
a    2
b    5
c    3
d    4
d    1
```

```
In: s.rank(method='max', ascending=False)
Out:
a    4
b    1
c    3
d    3
d    5
```

Ranking

- Rank methods on Series and DataFrame

```
In: df
Out:
   Paris  Berlin  Madrid
b      0     1       2
a      3     7       5
c      6     4       8
```

```
In: df.rank()
Out:
   Paris  Berlin  Madrid
b      1     1       1
a      2     3       2
c      3     2       3
```

```
In: df.rank(axis=1)
Out:
   Paris  Berlin  Madrid
b      1     2       3
a      1     3       2
c      2     1       3
```

- Value = rank in the specified axis

Series ordering/sorting

- Order method: only on Series

```
In: s
Out:
a     3.0
b     7.0
c     4.0
d     4.0
d     0.3
```

```
In: s.order([ascending=True])
Out:
d     0.3
a     3.0
c     4.0
d     4.0
b     7.0
```

- Sort method by index

```
In:
s.sort_index(ascending=False)
Out:
d     0.3
d     4.0
c     4.0
b     7.0
a     3.0
```

DataFrame ordering/sorting

- No order method for DataFrame: specify the axis

```
In: df
Out:
    Paris  Berlin  Madrid
b     0      1       2
a     3      7       5
c     6      4       8
```

```
In:df.sort_index([ascending=True])
Out:
    Paris  Berlin  Madrid
a     3      7       5
b     0      1       2
c     6      4       8
```

```
In:df.sort_index(by = 'Berlin')
Out:
        Paris  Berlin  Madrid
b         0      1       2
c         6      4       8
a         3      7       5
```

```
In: df.sort_index(axis=1)
Out:

      Berlin  Madrid   Paris
a       7       5       3
b       1       2       0
c       4       8       6
```

Function application

- Basics operations on Series and DataFrame values

```
In: df
Out:
     Paris  Berlin  Madrid
b      0       1       2
a      3       7       5
c      6       4       8
```

```
In:df.max()
Out:
Paris     6
Berlin    7
Madrid    8
```

```
In: df + df.max()
Out:
     Paris  Berlin  Madrid
b      6       8      10
a      9      11      13
c     12      14      16
```

- **Warning: operations are applied among 1D arrays --> the output object is a serie**

```
f = lambda x: math.sqrt(x)


In: df.apply(f)
Out:
TypeError: ('only length-1 arrays can be converted to Python
scalars', u'occurred at index Paris')
```

Function application

- Apply mathematical functions directly on values

```
In: df
Out :
    Paris  Berlin  Madrid
b      0      1       2
a      3      7       5
c      6      4       8
```

```
f = lambda x: math.sqrt(x)
In: df.applymap(f)
Out:
      Paris     Berlin     Madrid
b  0.000000  1.000000  1.414214
a  1.732051  2.645751  2.236068
c  2.449490  2.000000  2.828427
```

```
df.Berlin = df['Berlin'].map(f)

In: df
Out:
   Paris   Berlin   Madrid
b      0   1.000000     2
a      3   2.645751     5
c      6   2.000000     8
```

Exercise

Assign in a new column 'Total' the sum of the others columns amount values applied with the function f(x) = x+ 0.2*x and sort the table by total value

Computing Descriptive Statistics

- Objects are equipped with a set of common statistical methods.

```
In: df
Out:
    Paris  Berlin  Madrid
b       0       1       2
a       3       7       5
c       6       4       8
```

```
In: df.sum(axis=1)
Out:
b     3
a    15
c    18
```

```
In: df.describe()
Out:
            Paris  Berlin   Madrid
count       3.0     3.0      3.0
mean        3.0     4.0      5.0
std         3.0     3.0      3.0
min         0.0     1.0      2.0
25%         1.5     2.5      3.5
50%         3.0     4.0      5.0
75%         4.5     5.5      6.5
max         6.0     7.0      8.0
```

- Covariance and correlation

```
In: df.cov()
Out:
        Paris  Berlin  Madrid
Paris     9.0     4.5     9.0
Berlin    4.5     9.0     4.5
Madrid    9.0     4.5     9.0
```

```
In: df.corr()
Out:
        Paris  Berlin  Madrid
Paris     1.0     0.5     1.0
Berlin    0.5     1.0     0.5
Madrid    1.0     0.5     1.0
```

Working on index

- Reindex Series and DataFrame

```
In: df
Out:
   Paris  Berlin  Madrid
b      0       1       2
a      3       7       5
c      6       4       8
```

```
In:df.reindex(['c','b','a','g'])
Out:
    Paris    Berlin   Madrid
c     6         4        8
b     0         1        2
a     3         7        5
g    NaN       NaN      NaN
```

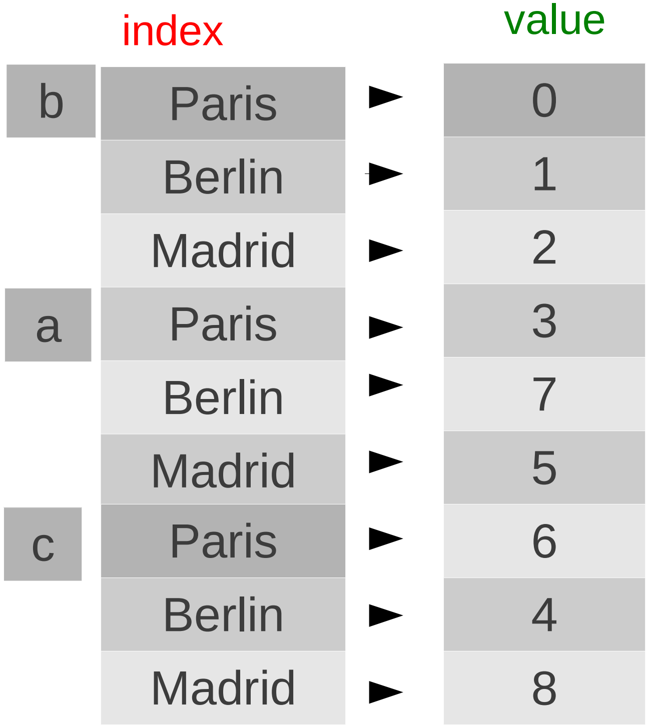
```
In:df.reindex(['c','b','a','g'],
fill_value = 14)
Out:
   Paris  Berlin  Madrid
c      6       4       8
b      0       1       2
a      3       7       5
g     14      14      14
```

```
In: df.reindex(columns = ['Varsovie','Paris','Madrid'])
Out:
    Varsovie  Paris  Madrid
b        NaN      0       2
a        NaN      3       5
c        NaN      6       8
```

Warning: be aware if no duplicate index: `df.index.is_unique`

Hierarchical indexing

- Indices are n-dimensional tables (n>1)
- Easy to build complex datasets



index			value
b	Paris	►	0
	Berlin	►	1
	Madrid	►	2
a	Paris	►	3
	Berlin	►	7
	Madrid	►	5
c	Paris	►	6
	Berlin	►	4
	Madrid	►	8

```
In: df.index
Out: MultiIndex
[(b, Paris), (b, Berlin), (b, Madrid),
 (a, Paris), (a, Berlin), (a, Madrid),
 (c, Paris), (c, Berlin), (c, Madrid)]
```

- Index are `MultiIndex` objects

Hierarchical indexing

- Build a hierarchical index from DataFrame columns

```
In: df
Out:
   Paris  Berlin  Madrid
b      0       1       2
a      3       7       5
c      6       4       8
```

```
df2 = df.set_index(['Berlin','Madrid'])
In: df2
Out:
                      Paris
Berlin Madrid
 1          2             0
 7          5             3
 4          8             6
```

- The field ***xs*** enables to select values from any index level

```
In: df2.xs(7, level = 0)
Out:
             Paris
Madrid
5                3
```

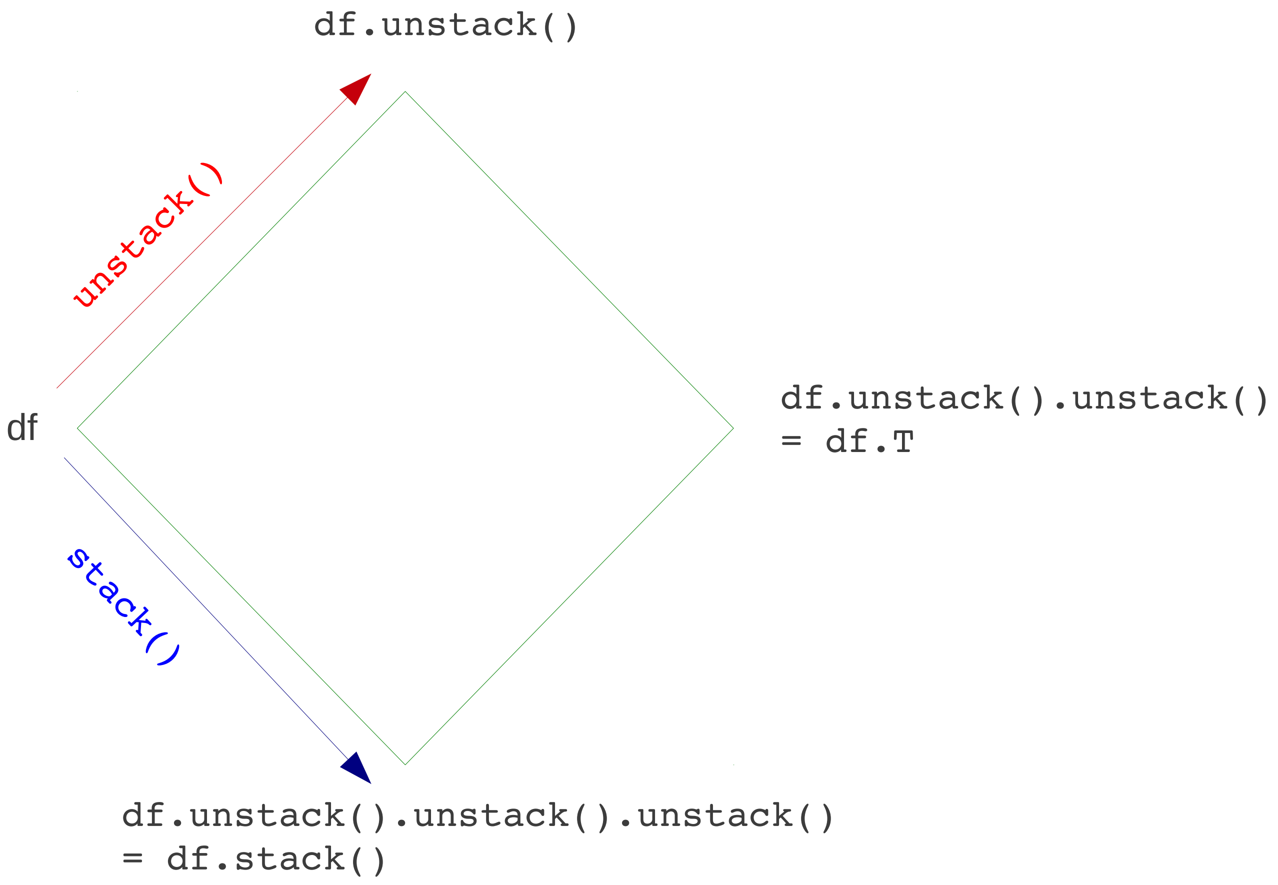
```
In: df2.xs(8, level = 1)
Out:
            Paris
Berlin
4               6
```

Exercise

Transform the DataFrame `dfTot` with a hierarchical index: `['Country', 'Id']`

Hierarchical indexing

- Conversion in Series/DataFrame with methods `stack()`/`unstack()`



`df.unstack()`

df

`df.unstack().unstack()`
`= df.T`

`df.unstack().unstack().unstack()`
`= df.stack()`

Hierarchical indexing

- Conversion in Series/DataFrame with methods `stack()/unstack()`

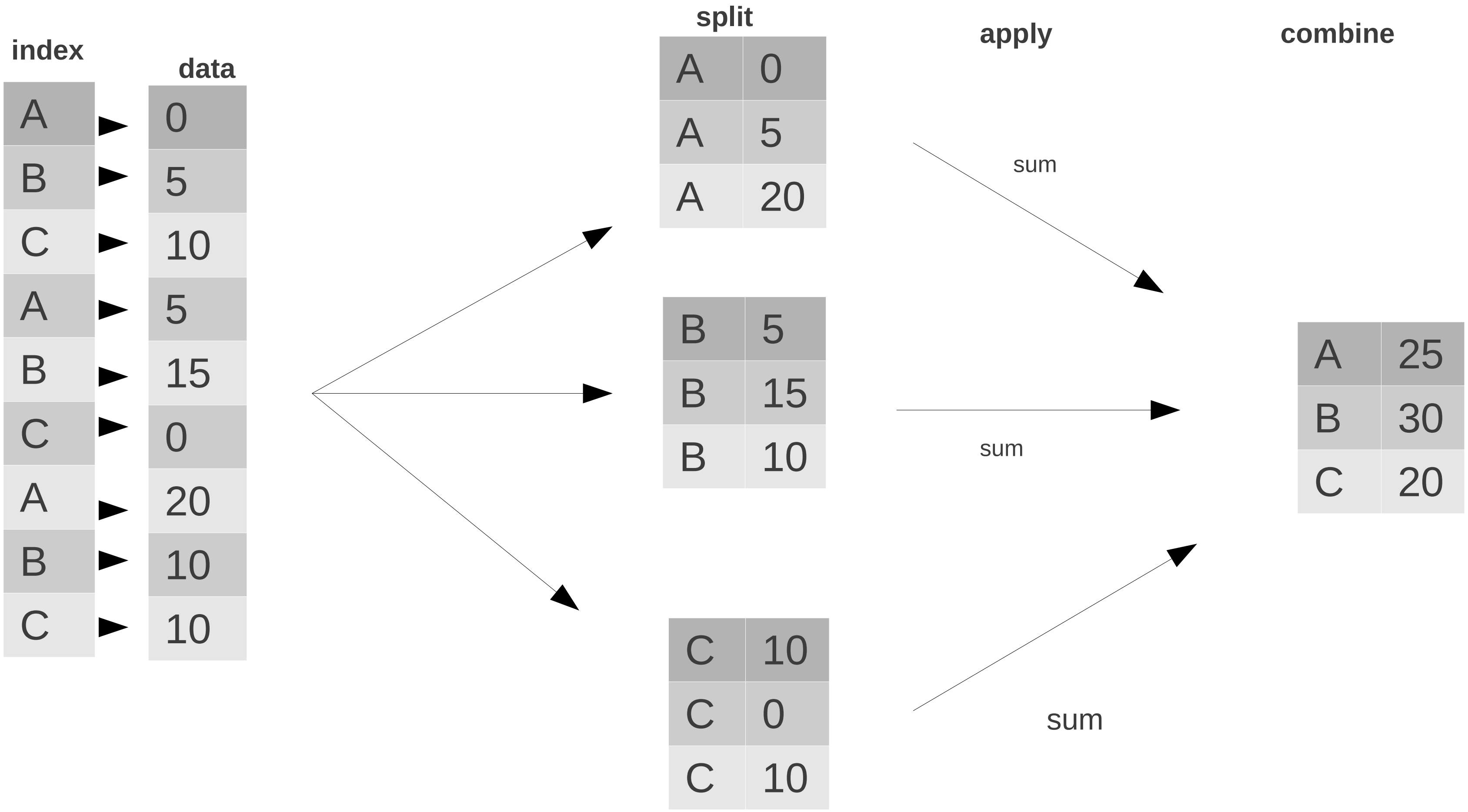
```
In: df
Out:
      Paris  Berlin  Madrid
b       0       1        2
a       3       7        5
c       6       4        8
```

```
In : df.unstack()
Out :
Paris      b      0
           a      3
           c      6
Berlin     b      1
           a      7
           c      4
Madrid     b      2
           a      5
           c      8
```

```
In: df.stack()
Out:
b    Paris       0
     Berlin      1
     Madrid      2
a    Paris       3
     Berlin      7
     Madrid      5
c    Paris       6
     Berlin      4
     Madrid      8
```

```
In: df.unstack().unstack()
Out:
           b  a  c
Paris      0  3  6
Berlin     1  7  4
Madrid     2  5  8
```

The *groupby* Object



index	data
A	0
B	5
C	10
A	5
B	15
C	0
A	20
B	10
C	10

split

A	0
A	5
A	20

B	5
B	15
B	10

C	10
C	0
C	10

apply

sum

sum

sum

combine

A	25
B	30
C	20

Illustration of a *groupby* process

groupby: a concrete example

```
In: dfG
Out:
      data1     data2     key1   key2
0 -0.822677  0.120968     a       one
1  0.199444  0.713446     a       two
2  0.054523 -0.530082     b       one
3 -1.087544 -1.952220     b       two
4  0.591362 -0.446848     a       one
```

```
In: group = dfG.groupby(['key1', 'key2'])
    group
Out:
<pandas.core.groupby.DataFrameGroupBy
object at 0x3960f90>
```

- All operations are possible from the groupby

```
In: group[data1].mean()
Out:
                 data1
key1 key2
a       one    -0.115657
        two     0.199444
b       one     0.054523
        two    -1.087544
```

```
In: group.size()
Out:
key1   key2
a       one         2
        two         1
b       one         1
        two         1
```

Exercise

Give the mean 'Rank' by 'Id' using the groupby object
Drop the rows which contains duplicated Id

Draw the graph

Exercise

- Our DataFrame is now cleaner and well ordered
  - Draw the associated graph:
    - Nodes: Project item
    - Edges: same country between two nodes
  - Set the DataFrame index with the referenced `tlp.node` object
  - Create a new DataFrame with all the graphic properties (at least viewLayout) of each node
    - using this DataFrame, draw the nodes on a line starting at the first node's position

Conclusion

- Manipulate data easily and fastly
- Intuitive representation
- N-hierarchical index and groupby: most powerful tool of pandas
- Statistics methods and calculs based on R language
- Nearly impossible to combine or adjust misaligns data