



Introducción a pandas

Departamento de Estadística, Informática y Matemáticas
UPNA



Índice



- Estructuras de datos
 - Series
 - DataFrame
- Indexación
 - Acceso a índices
 - Acceso a elementos
- Operaciones básicas
- Lectura/Escritura de ficheros
- Combinación o unión de datos
- Agregación de datos y operaciones de agrupamiento



Introducción



- Librería que ofrece
 - estructuras de datos de alto nivel
 - Series
 - DataFrame
 - herramientas para su manipulación
- Objetivo de pandas: análisis de datos de manera **sencilla** y **eficiente** en Python
- pandas está construido sobre NumPy
 - nos permite utilizar muchas funcionalidades de NumPy
- Característica principal: indexación
 - Permite el agrupamiento de forma eficiente

Estructuras de datos: Series

- Estructuras de datos de alto nivel
 - Series
 - Un objeto (tipo array) unidimensional que contiene
 - un conjunto de datos
 - un conjunto de índices asociados a los datos

index		values
A	→	5
B	→	6
C	→	12
D	→	-5
E	→	6.7

Estructuras de datos: Series

- Creación de series

- A partir de listas en Python

- `serie = Series([4, -5, 7, 2])`

- Al no haber especificado los índices, por defecto se asignan 0, 1, 2, ..., N-1 (siendo N el número de elementos de la serie)

```
serie = Series([1, -4, 5, 7])
serie
0      1
1     -4
2      5
3      7
dtype: int64
```

Estructuras de datos: Series

► Creación de series

► A partir de listas en Python

► `serie = Series([4, -5, 7, 2], index=['a', 'b', 'c', 'd'])`

```
serie = Series([4, -5, 7, 2], index=['a', 'b', 'c', 'd'])  
serie
```

```
a    4  
b   -5  
c    7  
d    2  
dtype: int64
```

► Ahora se parece más a un diccionario...

► `serie['a'] -> 4; serie['d'] -> 2`

► `serie[['a', 'd']] -> Series con los índices/valores asociados a 'a' y 'd'`

Estructuras de datos: Series

- Creación de series
 - A partir de diccionarios

```
#A partir de diccionarios  
d = {'Pamplona':195853, 'Tudela': 35388, 'Estella':13702, 'Alsasua':7490}  
serie = pd.Series(d)  
serie.name = 'Poblacion'  
print (serie)
```

```
Pamplona    195853  
Tudela      35388  
Estella     13702  
Alsasua     7490  
Name: Poblacion, dtype: int64
```

Estructuras de datos: Series

- Creación de series
 - Las series tienen asociado un atributo name
 - `serie.name = 'Población'`

```
#A partir de diccionarios  
d = {'Pamplona':195853, 'Tudela': 35388, 'Estella':13702, 'Alsasua':7490}  
serie = pd.Series(d)  
serie.name = 'Poblacion'  
print (serie)
```

```
Pamplona    195853  
Tudela      35388  
Estella     13702  
Alsasua      7490  
Name: Poblacion, dtype: int64
```


Estructuras de datos: Series

► Algunas operaciones

```
print (serie>10000)
```

```
Pamplona      True
Tudela        True
Estella       True
Alsasua       False
Name: Poblacion, dtype: bool
```

```
print ('Pamplona' in serie)
print ('Barañain' in serie)
```

```
print (serie[serie>10000])
```

```
Pamplona      195853
Tudela         35388
Estella        13702
Name: Poblacion, dtype: int64
```



Estructuras de datos: DataFrame

- Estructuras de datos de alto nivel
 - DataFrame
 - Estructura de datos tabular que contiene
 - Una colección ordenada de columnas
 - Cada columna puede ser de un tipo diferente
 - Tanto las filas como las columnas tienen índice
 - Generalmente llamamos índice a las filas
 - Y columnas a las columnas

Estructuras de datos: DataFrame

columns		foo	bar	baz	qux
index					
A	→	0	x	2.7	True
B	→	4	y	6	True
C	→	8	z	10	False
D	→	-12	w	NA	False
E	→	16	a	18	False

Estructuras de datos: DataFrame

► Creación de DataFrame

- A partir de diccionarios cuyos valores son listas de la misma dimensión

```
d = {'localidad': ['Pamplona', 'Pamplona', 'Pamplona', 'Tudela', 'Estella', 'Alsasua'],  
     'poblacion': [183964, 196166, 195853, 35388, 13702, 7490],  
     'año': [2001, 2014, 2015, 2015, 2015, 2015]}  
data = pd.DataFrame(d)  
print (data)
```

	localidad	poblacion	año
0	Pamplona	183964	2001
1	Pamplona	196166	2014
2	Pamplona	195853	2015
3	Tudela	35388	2015
4	Estella	13702	2015
5	Alsasua	7490	2015

Estructuras de datos: DataFrame

- Creación de DataFrame
 - Podemos cambiar el orden de las columnas
 - E incluso crear columnas nuevas

```
#Reordenando columnas
data = pd.DataFrame(d, columns = ['localidad', 'poblacion', 'año', 'densidad'])
print (data)
```

	localidad	poblacion	año	densidad
0	Pamplona	183964	2001	NaN
1	Pamplona	196166	2014	NaN
2	Pamplona	195853	2015	NaN
3	Tudela	35388	2015	NaN
4	Estella	13702	2015	NaN
5	Alsasua	7490	2015	NaN

Estructuras de datos: DataFrame

- Creación de DataFrame: acceso a columnas

```
#Acceso a columnas
print (data['localidad'])
print (type(data['localidad']))
```

0	Pamplona
1	Pamplona
2	Pamplona
3	Tudela
4	Estella
5	Alsasua

```
Name: localidad, dtype: object
<class 'pandas.core.series.Series'>
```

```
print (data.poblacion)
print (type(data.poblacion)) #Cada una
```

0	183964
1	196166
2	195853
3	35388
4	13702
5	7490

```
Name: poblacion, dtype: int64
<class 'pandas.core.series.Series'>
```

Indexación

- Acceso a índices
 - Devuelve un objeto Index con elementos de la indexación
 - serie.index
 - data.index
 - Los índices son inmutables

```
#Acceso a colección de índices y columnas
print (data.index)
data.index[0] = 6 #Esto nos devuelve un fallo. El i
```

```
RangeIndex(start=0, stop=6, step=1)
```

```
-----
TypeError                                Traceback
<ipython-input-64-7f3c62d51652> in <module>
      1 #Acceso a colección de índices y columnas
      2 print (data.index)
```

Indexación

- ¿Y no podemos reindexar una vez creado el DataFrame?

```
#Reindexación (reordenar las filas)
data = pd.DataFrame(d, columns = ['localidad', 'poblacion', 'año', 'densidad'])
print (data)
data = data.reindex([3, 4, 5, 1, 2, 3, 6])
print (data)
```

	localidad	poblacion	año	densidad
0	Pamplona	183964	2001	NaN
1	Pamplona	196166	2014	NaN
2	Pamplona	195853	2015	NaN
3	Tudela	35388	2015	NaN
4	Estella	13702	2015	NaN
5	Alsasua	7490	2015	NaN

	localidad	poblacion	año	densidad
3	Tudela	35388.0	2015.0	NaN
4	Estella	13702.0	2015.0	NaN
5	Alsasua	7490.0	2015.0	NaN
1	Pamplona	196166.0	2014.0	NaN
2	Pamplona	195853.0	2015.0	NaN
3	Tudela	35388.0	2015.0	NaN
6	NaN	NaN	NaN	NaN

Indexación

- ¿Y cómo obtener las columnas?
 - `dataframe.columns`
 - Vuelve a ser inmutable

```
#Acceso a la colección de columnas
columnas = data.columns
print (columnas)
columnas[0] = 'poblacion' #Esto nos da fallo
```

```
Index(['localidad', 'poblacion', 'año', 'densidad'], dtype='object')
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-68-de5aaa641d2e> in <module>
      2 columnas = data.columns
      3 print (columnas)
```



Indexación



- Iteración sobre los elementos de una Series
 - `serie.iteritems()`: devuelve un conjunto de tuplas (índice, valor)
- Iteración sobre los elementos de un DataFrame
 - `dataframe.iterrows()`: iteración por filas
 - Devuelve un conjunto de tuplas (índice, Series)
 - `dataframe.iteritems()`: iteración por columnas
 - Devuelve un conjunto de tuplas (columna, Series)

Indexación

```
In [96]: df
Out[96]:
```

	dos	uno
a	4	1
b	5	2
c	6	3
d	7	NaN

```
In [97]: for column, values in df.iteritems():
...:     print column
...:     print list(values)
...:
dos
[4, 5, 6, 7]
uno
[1.0, 2.0, 3.0, nan]

In [98]: for row, values in df.iterrows():
...:     print row
...:     print list(values)
...:
a
[4.0, 1.0]
b
[5.0, 2.0]
c
[6.0, 3.0]
d
[7.0, nan]
```

Indexación

► Acceso a elementos

- Series: igual que ndarray de NumPy, excepto que podemos utilizar el índice de la serie en lugar de solo números

```
s = pd.Series([0, 1, 2, 3], index=['a', 'b', 'c', 'd'])  
print (s)
```

```
a    0  
b    1  
c    2  
d    3  
dtype: int64
```

```
print (s['b'])  
print (s[1])  
print (s[0:3])  
print (s['a':'d'])
```

```
1  
1  
a    0  
b    1  
c    2  
dtype: int64  
a    0  
b    1  
c    2  
d    3  
dtype: int64
```

Indexación

- Acceso a elementos
 - DataFrames: obtener una o más columnas

```
print(data)
print (data['localidad'])
print (data[['localidad', 'año']])
```

	localidad	poblacion	año	densidad
3	Tudela	35388.0	2015.0	NaN
4	Estella	13702.0	2015.0	NaN
5	Alsasua	7490.0	2015.0	NaN
1	Pamplona	196166.0	2014.0	NaN
2	Pamplona	195853.0	2015.0	NaN
3	Tudela	35388.0	2015.0	NaN
6	NaN	NaN	NaN	NaN

	localidad	año
3	Tudela	2015.0
4	Estella	2015.0
5	Alsasua	2015.0
1	Pamplona	2014.0
2	Pamplona	2015.0
3	Tudela	2015.0
6	NaN	NaN

Name: localidad, dtype: object

Indexación

- Acceso a elementos
 - DataFrames: obtener una o más filas

```
In [29]: try:
          print data[1]
        except KeyError:
          print "Fallo al intentar indexar data[0]"
          print data[0:1]
          print data[2:]
```

```
Fallo al intentar indexar data[0]
  localidad poblacion  año densidad
1 Pamplona    183964  2001      NaN
  localidad poblacion  año densidad
3 Pamplona    195853  2015      NaN
4 Tudela      35388   2015      NaN
5 Estella     13702   2015      NaN
6 Alsasua     7490    2015      NaN
```

- A veces se hace un poco complicado, e incluso podemos llegar a ciertos errores...



Indexación



- Acceso a elementos: 2 maneras de acceder en DataFrames
 - loc: principalmente para indexar filas/columnas a través del nombre de las etiquetas
 - iloc: principalmente para indexar filas/columnas a través de índices numéricos
- Un DataFrame puede devolver
 - Otro DataFrame si se indexa una combinación de filas/columnas
 - Un Series si se indexa una única fila/columna
 - Un valor si se indexa una fila/columna específica

Indexación

```
In [38]: d = {'localidad': ['Pamplona', 'Pamplona', 'Pamplona', 'Tudela', 'Estella', 'Alsasua'],
            'poblacion': [183964, 196166, 195853, 35388, 13702, 7490],
            'año': [2001, 2014, 2015, 2015, 2015, 2015]}
data = DataFrame(d, columns = ['localidad', 'poblacion', 'año', 'densidad'], index=[1, 2, 3, 4, 5, 6])
print data
print data.loc[4, 'localidad']
print data.loc[1, ['localidad', 'poblacion']]
print data.loc[1:3, 'localidad': 'año']
try:
    data.loc[0, 'localidad']
except KeyError:
    print "No existe la etiqueta 0"
```

	localidad	poblacion	año	densidad
1	Pamplona	183964	2001	NaN
2	Pamplona	196166	2014	NaN
3	Pamplona	195853	2015	NaN
4	Tudela	35388	2015	NaN
5	Estella	13702	2015	NaN
6	Alsasua	7490	2015	NaN

Tudela

localidad Pamplona

poblacion 183964

Name: 1, dtype: object

	localidad	poblacion	año
--	-----------	-----------	-----

1	Pamplona	183964	2001
---	----------	--------	------

2	Pamplona	196166	2014
---	----------	--------	------

3	Pamplona	195853	2015
---	----------	--------	------

No existe la etiqueta 0

Indexación

```
In [52]: d = {'localidad':['Pamplona', 'Pamplona', 'Pamplona', 'Tudela', 'Estella', 'Alsasua'],
              'poblacion':[183964, 196166, 195853, 35388, 13702, 7490],
              'año': [2001, 2014, 2015, 2015, 2015, 2015]}
data = DataFrame(d, columns = ['localidad', 'poblacion', 'año', 'densidad'], index=[1, 2, 3, 4, 5, 6])
print data
print data.iloc[0,2]
print data.iloc[0,[0, 1]]
print data.iloc[0:2,0:3]
```

	localidad	poblacion	año	densidad
1	Pamplona	183964	2001	NaN
2	Pamplona	196166	2014	NaN
3	Pamplona	195853	2015	NaN
4	Tudela	35388	2015	NaN
5	Estella	13702	2015	NaN
6	Alsasua	7490	2015	NaN

2001

localidad Pamplona

poblacion 183964

Name: 1, dtype: object

	localidad	poblacion	año
--	-----------	-----------	-----

1	Pamplona	183964	2001
---	----------	--------	------

2	Pamplona	196166	2014
---	----------	--------	------

Indexación

- Indexación por booleanos
 - Igual que en NumPy

```
In [136]: frameDD
Out[136]:
```

	FL	GA
2008	NaN	9.7
2010	18.8	9.7
2011	19.1	9.8

```
In [137]: frameDD < 9.8
Out[137]:
```

	FL	GA
2008	False	True
2010	False	True
2011	False	False

```
In [138]: frameDD[frameDD < 9.8] = 0

In [139]: frameDD
Out[139]:
```

	FL	GA
2008	NaN	0.0
2010	18.8	0.0
2011	19.1	9.8

```
In [171]: f
Out[171]:
```

	one	two
0	2	2
1	4	4
2	6	6

```
In [172]: aux = f['one'] > 3

In [173]: f[aux]
Out[173]:
```

	one	two
1	4	4
2	6	6

Operaciones básicas

► Operaciones básicas

- Una de las potencias de pandas es la posibilidad de realizar operaciones aritméticas entre objetos de diferentes índices

- $a+b$ `a.add(b)`

- $a-b$ `a.sub(b)`

- $a*b$ `a.mul(b)`

- a/b `a.div(b)`

Operaciones básicas

```
s1 = Series([7.3, -2.5, 3.4, 1.5], index = ['a', 'c', 'd', 'e'])  
s2 = Series([-2.1, 3.6, -1.5, 4, 3.1], index = ['a', 'b', 'c', 'd', 'e'])  
print s1+s2  
print s1.add(s2, fill_value=0)
```

```
a    5.2  
b    NaN  
c   -4.0  
d    7.4  
e    4.6  
dtype: float64  
a    5.2  
b    3.6  
c   -4.0  
d    7.4  
e    4.6  
dtype: float64
```

Operaciones básicas

► Operaciones básicas

- El broadcasting (propagación) vista en NumPy también se aplica

```
#Propagación
df1 = pd.DataFrame(np.arange(12).reshape(3,4) , columns=['a', 'b', 'c', 'd'])
serie = df1.iloc[0]
print df1
print serie
print df1-serie
```

```
   a  b  c  d
0  0  1  2  3
1  4  5  6  7
2  8  9 10 11
a    0
b    1
c    2
d    3
Name: 0, dtype: int32
   a  b  c  d
0  0  0  0  0
1  4  4  4  4
2  8  8  8  8
```

Operaciones básicas

► Operaciones básicas

- Todas las funciones que se podían aplicar en NumPy también pueden hacerse sobre un Series/DataFrame

```
In [65]: #Aplicación de funciones
frame = DataFrame(np.random.randn(4,3), columns=list('abc'), index = list('wxyz'))
print frame
print np.abs(frame)
print frame**2
```

	a	b	c
w	2.566959	-0.517779	-0.185874
x	1.418474	-1.555041	-0.849757
y	-0.037238	0.393216	0.399346
z	0.954156	0.459932	0.951014

	a	b	c
w	2.566959	0.517779	0.185874
x	1.418474	1.555041	0.849757
y	0.037238	0.393216	0.399346
z	0.954156	0.459932	0.951014

	a	b	c
w	6.589278	0.268095	0.034549
x	2.012068	2.418153	0.722087
y	0.001387	0.154619	0.159477
z	0.910414	0.211537	0.904427



Operaciones básicas

- Operaciones básicas

- `Series.map(función)`

- Se aplica dicha función sobre cada elemento de la Series

- `DataFrame.apply(función)`

- Se aplica dicha función sobre cada columna (o fila) del DataFrame. Depende del axis que especifiquemos (por defecto axis = 0)

- `DataFrame.applymap(función)`

- Se aplica dicha función sobre cada elemento del DataFrame

Operaciones básicas

```
print frame  
frame.apply(lambda x: (x-x.min())/(x.max()-x.min()))
```

	a	b	c
w	2.566959	-0.517779	-0.185874
x	1.418474	-1.555041	-0.849757
y	-0.037238	0.393216	0.399346
z	0.954156	0.459932	0.951014

	a	b	c
w	1.000000	0.514777	0.368666
x	0.558987	0.000000	0.000000
y	0.000000	0.966890	0.693649
z	0.380691	1.000000	1.000000

Operaciones básicas

```
In [84]: print frame  
         print frame.applymap(lambda x:int(x)*10%4)
```

	a	b	c
w	2.566959	-0.517779	-0.185874
x	1.418474	-1.555041	-0.849757
y	-0.037238	0.393216	0.399346
z	0.954156	0.459932	0.951014

	a	b	c
w	0	0	0
x	2	2	0
y	0	0	0
z	0	0	0



Operaciones básicas

- Operaciones básicas: visualización
 - `head()`: muestra las primeros 5 líneas del dataset
 - `head(n)`: muestra las n primer líneas
 - `tail()`: muestra las últimas 5 líneas del dataset
 - `tail(n)`: muestra las n últimas líneas
- En todas las opciones anteriores podemos acceder a datos concretos
 - `dataframe.head(). 'localidad'`
 - `dataframe.localidad.head()`




Operaciones básicas: estadísticos

- Estadísticos: permiten extraer agregaciones o reducciones
 - un único valor de una Series
 - Una Series de valores de un DataFrame
- Opciones
 - axis
 - Permite realizar por filas/columnas

Operaciones básicas: estadísticos

Method	Description
count	Number of non-NA values
describe	Compute set of summary statistics for Series or each DataFrame column
min, max	Compute minimum and maximum values
argmin, argmax	Compute index locations (integers) at which minimum or maximum value obtained, respectively
idxmin, idxmax	Compute index values at which minimum or maximum value obtained, respectively
quantile	Compute sample quantile ranging from 0 to 1
sum	Sum of values
mean	Mean of values
median	Arithmetic median (50% quantile) of values
mad	Mean absolute deviation from mean value
var	Sample variance of values
std	Sample standard deviation of values
skew	Sample skewness (3rd moment) of values
kurt	Sample kurtosis (4th moment) of values
cumsum	Cumulative sum of values
cummin, cummax	Cumulative minimum or maximum of values, respectively
cumprod	Cumulative product of values
diff	Compute 1st arithmetic difference (useful for time series)
pct_change	Compute percent changes



Operaciones básicas: ordenación y ranking

- Ordenación
 - Ordenar una Serie por índice
 - `s.sort_index()`
 - Ordenar una Serie por valor
 - `s.order_values()`
 - Ordenar un DataFrame por índice
 - `f.sort_index()`
 - Ordenar un DataFrame por columna/s
 - `f.sort_values(by='columna')`
 - `f.sort_values(by=['columna1', 'columna2'])`



Lectura / escritura de ficheros

- Lectura de datos: `read_csv`
 - `path` del archivo (o URL)
 - `sep=','` (sirve para especificar el separador)
 - `header = 0` (número de la fila donde el nombre de las columnas está especificado)
 - `names` -> lista de los nombres de las columnas
 - `nrows` -> número de filas que se quieren leer
- Escritura de datos
 - `dataframe.to_csv('archivo.csv')`



Combinación o unión de datos

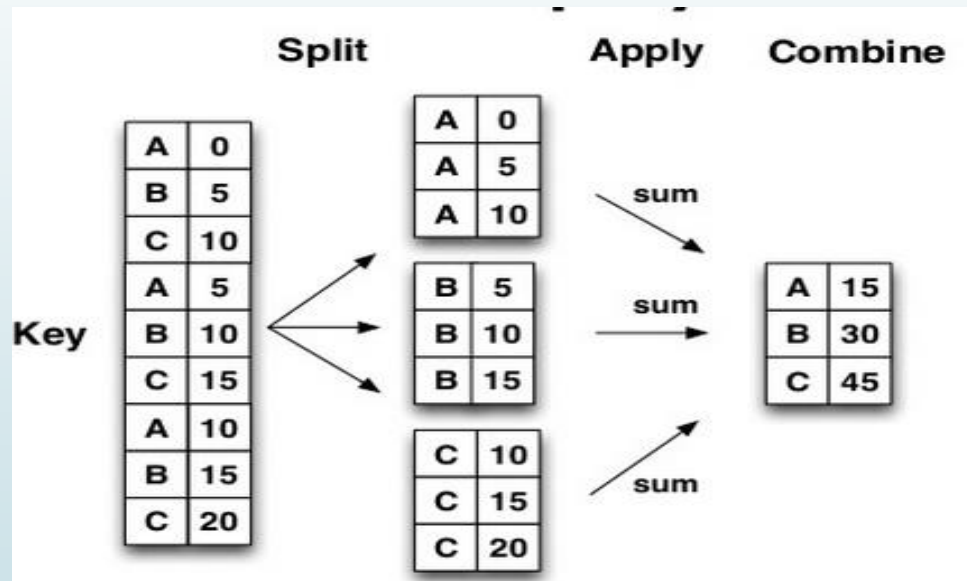
- `pd.merge(df1, df2)`
 - Permite la unión de dos DataFrames mediante una o más claves (al estilo d SQL JOIN)
 - Por defecto, merge aplica “inner join”
 - El conjunto de claves es la intersección de las claves de ambas instancias
 - Si incorporamos `how={'inner', 'left', 'right', 'outer'}`
 - left: el conjunto de claves es igual a las claves del primer DataFrame (aparecerá NaN si una clave no aparece en df2)
 - right: al revés de left
 - outer: el conjunto de claves es la unión de las claves



Agregación de datos y operaciones de agrupamiento

- Operaciones usuales
 - calcular estadísticos agrupados alrededor de una o varias claves
 - Posibilidad de calcular pivot tables mediante la agrupación
 - ...
- Todas las operaciones de agrupamiento
 - Separar los datos por clave
 - Realizar una operación sobre los datos con una misma clave

Agregación de datos y operaciones de agrupamiento





Agregación de datos y operaciones de agrupamiento

- La función GroupBy separa los datos en diferentes grupos dependiendo en una clave
 - Devuelve un objeto groupby, que
 - contiene datos intermedios alrededor de la clave utilizada
 - permite aplicar una serie de operaciones sobre dichos datos
 - mean
 - min
 - max
 - ...

Agregación de datos y operaciones de agrupamiento

➡ Dev

```
df = pd.DataFrame({'clave1':list('aabba'),  
                  'clave2':['uno', 'dos', 'uno', 'dos', 'uno'],  
                  'd1': np.random.randn(5),  
                  'd2': np.random.randn(5)})  
  
print df  
obj = df.groupby('clave1')  
print obj.mean()  
print obj.mean().d1  
print obj.mean().d2
```

```
clave1 clave2      d1      d2  
0      a     uno  0.737720  0.859498  
1      a     dos -0.236364 -0.738238  
2      b     uno  0.641429  1.235890  
3      b     dos  0.118674 -0.255720  
4      a     uno  0.644284 -2.071992  
      d1      d2
```

```
clave1  
a      0.381880 -0.650244  
b      0.380051  0.490085
```

```
clave1  
a      0.381880  
b      0.380051  
Name: d1, dtype: float64
```

```
clave1  
a      -0.650244  
b       0.490085  
Name: d2, dtype: float64
```

Agregación de datos y operaciones de agrupamiento

- Incluso podemos especificar una operación definida por el usuario o una lista de operaciones mediante la función `agg`

```
print obj.agg(lambda x: 0.5*min(x)+0.5*max(x))
d = {'d1': ['mean', 'prod'],
     'd2': ['min', 'max']}
print obj.agg(d)
```

	d1	d2
clave1		
a	-0.375649	0.229567
b	-0.286299	0.533919

	d2		d1	
	min	max	mean	prod
clave1				
a	-0.557564	1.016698	-0.465662	0.037384
b	0.373989	0.693850	-0.286299	-0.562142



Agregación de datos y operaciones de agrupamiento

- Pivot_tables: cuando lo que queremos es visualizar el resultado de un agrupamiento en forma de tabla, podemos crear una pivot_table
- `data.pivot_table(values=None, index=None, columns=None, aggfunc='mean', fill_value=None)`
 - values: la columna utilizada para realizar la agregación
 - Index: los campos que se utilizan como índice en la pivot table
 - Columns: los campos que se utilizan como columnas en la pivot table
 - Aggfunc: la agregación a aplicar
 - Fill_value: el valor usado para rellenar campos vacíos