Introducción a pandas

http://pandas.pydata.org/pandas-docs/stable

Pandas es una librería que proporciona estructuras de datos flexibles y permite trabajar con la información de forma eficiente (gran parte de Pandas está implementado usando C/Cython para obtener un buen rendimiento).

Funciona muy bien cuando nos toca trabajar con:

- Datos heterogéneos que pueden distribuirse de forma tabular.
- Series temporales
- Matrices

Pandas es una librería de python destinada al análisis de datos, que proporciona unas estructuras de datos flexibles y que permiten trabajar con ellos de forma muy eficiente. Pandas ofrece las siguientes estructuras de datos:

- **Series:** Son arrays unidimensionales con indexación (arrays con índice o etiquetados), similar a los diccionarios. Pueden generarse a partir de diccionarios o de listas.
- **DataFrame:** Son estructuras de datos similares a las tablas de bases de datos relacionales como SQL.
- **Panel, Panel4D y PanelND**: Estas estructuras de datos permiten trabajar con más de dos dimensiones.

Lo primero que debemos hacer es importar la librería de Pandas

import pandas as pd

Por convenio se pone " *pd* " como alias de la librería Pandas.El primer ejemplo que vamos a poner va a ser el de definir una estructura de datos " *Series* " que como ya comentamos es un array de datos unidimensional con indexación. Las "Series" se definen de la siguiente manera:

serie = pd.Series(data, index=index)

Es decir, que en el primer parámetro le indicamos los datos del array y en el segundo parámetro los índices.

Lectura de un fichero csv con pandas

Como alternativa al módulo csv, podemos utilizar la función read_csv de pandas. Especializada en la carga de archivos CSV, es parte de un amplio rango de funciones,Dedicado a la entrada / salida en diferentes formatos de archivo, según lo especificado por los pandas

Documentación en http://pandas.pydata.org/pandas-docs/stable/io.html

http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html#pandas.read_csv

pandas.read_csv

pandas.**read_csv**(*filepath_or_buffer, sep=', ', delimiter=None, header='infer', names=None, index_col=None, usecols=None, squeeze=False, prefix=None, mangle_dupe_cols=True, dtype=None, engine=None, converters=None, true_values=None, false_values=None, skipinitialspace=False, skiprows=None, nrows=None, na_values=None, keep_default_na=True, na_filter=True, verbose=False, skip_blank_lines=True, parse_dates=False, infer_datetime_format=False, keep_date_col=False, date_parser=None, dayfirst=False, iterator=False, chunksize=None, compression='infer', thousands=None, decimal='.', lineterminator=None, quotechar='"', quoting=0, escapechar=None, comment=None, encoding=None, dialect=None, tupleize_cols=False, error_bad_lines=True, warn_bad_lines=True, skipfooter=0, skip_footer=0, doublequote=True, delim_whitespace=False, as_recarray=False, compact_ints=False, use_unsigned=False, low_memory=True, buffer_lines=None, memory_map=False, float_precision=None*) [source]

Read CSV (comma-separated) file into DataFrame

Also supports optionally iterating or breaking of the file into chunks.

Additional help can be found in the online docs for IO Tools.

Parameters:

filepath_or_buffer : *str, pathlib.Path, py._path.local.LocalPath or any object with a read() method (such as a file handle or StringIO)*

The string could be a URL. Valid URL schemes include http, ftp, s3, and file. For file URLs, a host is expected. For instance, a local file could be file ://localhost/path/to/table.csv

sep : *str, default ','*

Delimiter to use. If sep is None, will try to automatically determine this. Separators longer than 1 character and different from `'\s+'` will be interpreted as regular expressions, will force use of the python parsing engine and will ignore quotes in the data. Regex example: `'\r\t'`

delimiter : *str, default None*

import pandas as pd
filename='file.csv'
data=pd.read_csv(filename)

type(data)
>>pandas.core.frame.DataFrame

data.keys()
>>devuelve los nombres de las columnas csv

data.shape
>>devuelve el número de líneas del fichero

data.head()
>>devuelve las primeras líneas

data.tail()
>>devuelve las últimas líneas

Estructuras de datos en Pandas

Para comenzar con los pandas, tendrá que sentirse cómodo con sus dos estructuras de datos utilizadas en toda la biblioteca: ***Series*** **y** ***DataFrame***

Una serie es un objeto de tipo matriz unidimensional que contiene una matriz de datos (de cualquier tipo de datos NumPy) y una matriz asociada de etiquetas de datos, denominada índice.

```
import pandas as pd
import numpy as np
pd.Series([1,3,5,np.nan,6,8])

# 0 1
# 1 3
# 2 5
# 3 NaN
# 4 6
# 5 8
```

Un **DataFrame** representa una estructura tabular de datos similar a una hoja de cálculo que contiene una colección ordenada de columnas, cada una de las cuales puede ser un tipo de valor diferente (numérico, cadena, booleano, etc.).

Como se puede ver, DataFrame tiene tanto un índice de filas como de columnas.

```
dates = pd.date_range('20170101',periods=6)
pd.DataFrame(np.random.randn(6,4),index=dates,columns=list('ABCD'))

# A B C D
# 2017-01-01 0.469112 -0.282863 -1.509059 -1.135632
# 2017-01-02 1.212112 -0.173215 0.119209 -1.044236
# 2017-01-03 -0.861849 -2.104569 -0.494929 1.071804
# 2017-01-04 0.721555 -0.706771 -1.039575 0.271860
# 2017-01-05 -0.424972 0.567020 0.276232 -1.087401
# 2017-01-06 -0.673690 0.113648 -1.478427 0.524988
```

Combinar dataframes

Pandas proporciona varias facilidades para combinar fácilmente objetos con varios tipos de lógica de conjuntos para los índices y la funcionalidad de álgebra relacional en el caso de operaciones de combinación / combinación.

```
key = ['foo', 'foo']
left = pd.DataFrame({'key': key, 'lval': [1, 2]})

# key lval
# 0 foo 1
# 1 foo 2
```

```
right = pd.DataFrame({'key': key, 'rval': [4, 5]})

# key rval
# 0 foo 4
# 1 foo 5
```

```
pd.concat([left,right])

# key lval rval
# 0 foo 1 NaN
# 1 foo 2 NaN
# 0 foo NaN 4
# 1 foo NaN 5
```

```
merged = pd.merge(left, right, on='key')

# key lval rval
# 0 foo 1 4
# 1 foo 1 5
# 2 foo 2 4
# 3 foo 2 5
```