Análisis y Manipulación de Datos

Data BootCamp
Cámara Comercio Bilbao

Germán Alonso Lascurain
germanalonso@opendeusto.es

Índice de contenidos

1. Introducción
2. Introducción a Estructuras de Datos
    1. Introducción
    2. Series
    3. DataFrames
3. Manipulación de Estructuras de Datos
4. Datos de tipo texto
5. Datos de tipo categórico
6. Datos de tipo temporal

1. Introducción

Introducción - Pandas

- En el año 2008 Wes McKinney empezó a desarrollar Pandas por la necesidad que tenía de una herramienta flexible de alto rendimiento para realizar análisis cuantitativo en datos financieros
- Pandas es una biblioteca de software lenguaje de programación Python escrita como extensión de NumPy, para manipulación y análisis de datos
- Ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales
- El nombre deriva del término "datos de panel", término de econometría que designa datos que combinan una dimensión temporal con otra dimensión transversal

Introducción - Características

- Tipo de datos: DataFrame para manipulación de datos con indexación integrada.
- Herramientas para leer y escribir datos entre estructuras de dato en-memoria y formatos de archivo variados.
- Alineación de dato y manejo integrado de datos faltantes.
- Reestructuración y segmentación de conjuntos de datos.
- Segmentación vertical basada en etiquetas, indexación elegante, y segmentación horizontal de grandes conjuntos de datos.
- Inserción y eliminación de columnas en estructuras de datos.
- Agrupación predefinida en la biblioteca lo que permite realizar cadenas de operaciones dividir-aplicar-combinar sobre conjuntos de datos.
- Mezcla y unión de datos.
- Indexación jerárquica de ejes para trabajar con datos de altas dimensiones en estructuras de datos de menor dimensión.
- Funcionalidad de series de tiempo: generación de rangos de fechas y conversión de frecuencias, desplazamiento de ventanas estadísticas y de regresiones lineales, desplazamiento de fechas y retrasos.

Introducción - Instalación y carga

- Para instalar Pandas, dependiendo del gestor de paquetes tenemos las siguientes órdenes:
  - PIP
    - `pip install pandas`
  - Anaconda
    - `conda install pandas`
- Para importar la biblioteca a nuestro código usaremos:
  - `import pandas`
- Aunque es más cómodo darle un alias
  - `import pandas as pd`

Introducción - Dependencias

Package	Minimum supported version
setuptools	24.2.0
NumPy	1.15.4
python-dateutil	2.7.3
pytz	2017.2

Introducción - Dependencias

BeautifulSoup4	4.6.0	HTML parser for read_html (see note)
Jinja2		Conditional formatting with DataFrame.style
PyQt4		Clipboard I/O
PyQt5		Clipboard I/O
PyTables	3.4.3	HDF5-based reading / writing
SQLAlchemy	1.1.4	SQL support for databases other than sqlite
SciPy	0.19.0	Miscellaneous statistical functions
XLsxWriter	0.9.8	Excel writing
blosc		Compression for HDF5
fsspec	0.7.4	Handling files aside from local and HTTP
fastparquet	0.3.2	Parquet reading / writing
gcsfs	0.6.0	Google Cloud Storage access
html5lib		HTML parser for read_html (see note)
lxml	3.8.0	HTML parser for read_html (see note)
matplotlib	2.2.2	Visualization
numba	0.46.0	Alternative execution engine for rolling operations
openpyxl	2.5.7	Reading / writing for xlsx files
pandas-gbq	0.12.0	Google Big Query access
psycopg2		PostgreSQL engine for sqlalchemy
pyarrow	0.12.0	Parquet, ORC (requires 0.13.0), and feather reading / writing
pymysql	0.7.11	MySQL engine for sqlalchemy
pyreadstat		SPSS files (.sav) reading
pytables	3.4.3	HDF5 reading / writing
pyxlsb	1.0.6	Reading for xlsb files
qtpy		Clipboard I/O
s3fs	0.4.0	Amazon S3 access
tabulate	0.8.3	Printing in Markdown-friendly format (see tabulate)
xarray	0.8.2	pandas-like API for N-dimensional data
xclip		Clipboard I/O on linux
xlrd	1.1.0	Excel reading
xlwt	1.2.0	Excel writing
xsel		Clipboard I/O on linux
zlib		Compression for HDF5

2. Introducción a Estructuras de Datos

Introducción a Estructuras de Datos - `Series`

- Las `Series` es una matriz unidimensional etiquetada capaz de contener cualquier tipo de datos (enteros, cadenas, números en coma flotante, objetos Python, etc.).
- Las etiquetas de los ejes se conocen colectivamente como el índice. El método básico para crear una `Serie` es llamar:
  - `serie = pd.Series(mis_datos, index=index)`
- En el ejemplo, `mis_datos` puede ser:
  - Un diccionario
  - Un `ndarray` de **NumPy**
  - Un valor escalar
- La variable `index` es una lista de etiquetas asociadas a cada elemento de la serie. Por lo tanto, `mis_datos` e `index` han de tener la misma longitud

Introducción a Estructuras de Datos - `Series` a partir de un diccionario

- El siguiente ejemplo crea una `Series` a partir de un diccionario:

```
Diccionario=   {'uno':1,
                'dos':2,
                'tres':3,
                'cuatro':4,
                'cinco':5}

mi_serie=pd.Series(diccionario)
mi_serie
```

Introducción a Estructuras de Datos - `Series` a partir de un `ndarray`

- El siguiente ejemplo crea una `Series` a partir de un `ndarray`:

```
array=np.array([1,2,3,4,5])

indice=['uno','dos','tres','cuatro','cinco']

mi_serie=pd.Series(array,index=indice)
mi_serie
```

Introducción a Estructuras de Datos - Series a partir de un escalar

- El siguiente ejemplo crea una `Series` a partir de un `ndarray`:

```
numero=1
indice=['uno','dos','tres','cuatro','cinco']

mi_serie=pd.Series(numero,index=indice)
mi_serie
```

Introducción a Estructuras de Datos - `Series` como `ndarray`

- Las `Series` actúa de manera muy similar a un `ndarray`, y pueden ser utilizadas en la mayoría de las funciones de NumPy:
  - ```
    serie=pd.Series(datos,index=index)
    serie[0]
    serie[:5]
    serie[:-1]
    serie[serie > 10]
    serie[[1,3,5]]
    np.log(serie)
    ```
    ```

Introducción a Estructuras de Datos - Series como diccionario

- Una serie es como un diccionario de tamaño fijo en el que puedes obtener y establecer valores por etiqueta de índice
  - ```
    serie=pd.Series(datos,index=index)
    serie['uno']
    serie['dos']=2
    'tres'in serie
    'trece' in serie
    ```
    ```

Introducción a Estructuras de Datos - Operaciones con `Series`

- Cuando se trabaja con vectores en **NumPy**, no suele ser necesario hacer un bucle de valor a valor. Lo mismo ocurre cuando se trabaja con Series en pandas:
  - `serie = pd.Series(datos, index=index)`
    `s + s`
    `s * 5`
- Una diferencia clave entre la `Series` y el `ndarray` es que las operaciones entre `Series` alinean automáticamente los datos a partir de la etiqueta. De esta forma, se pueden escribir cálculos sin tener en cuenta si las series implicadas tienen las mismas etiquetas*:
  - `serie[1:] + serie[:-1]`

*si la etiqueta de una serie no está en la otra serie, la operación devolverá Nan

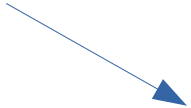
Introducción a Estructuras de Datos - DataFrame

- `DataFrame` es una estructura de datos etiquetada en 2 dimensiones con columnas de tipos comúnmente diferentes.
- Es similar a una **hoja de cálculo** o una **tabla SQL**, o un diccionario de objetos de tipo Series.
- Es el objeto de pandas más utilizado.
- Al igual que la serie, el `DataFrame` acepta muchos tipos diferentes de entrada:
  - Diccionario de 1D `ndarray`, listas, diccionarios, o `Series`
  - 2D `ndarray`
  - `ndarray` estructurado o de registro
  - Una `Series`
  - Otro `DataFrame`
- Junto con los datos, se puede pasar opcionalmente los argumentos de índice (etiquetas de filas) y de columnas (etiquetas de columnas).

Introducción a Estructuras de Datos - `DataFrame` a partir de un diccionario de `Series` o diccionarios

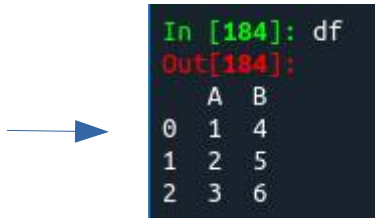
- El índice resultante será la unión de los índices de las distintas `Series`.
- Si hay algunos diccionarios anidados, estos se convertirán primero en `Series`.
- Si no se pasan columnas, las columnas serán la lista ordenada de claves de los diccionarios.
  - ```
    diccionario = {'uno': pd.Series(datos_1, index=index_1),
                   'dos': pd.Series(datos_2, index=index_2)}
    df = pd.DataFrame(diccionario)
    df
    ```
    ```

Introducción a Estructuras de Datos - `DataFrame` a partir de un diccionario de `ndarrays`

- Los `ndarrays` deben tener todos la misma longitud.
- Si se pasa un índice también debe ser de la misma longitud que los arrays.
- Si no se pasa ningún índice, el resultado será rango(n), donde n es la longitud del array.
  - ```
    diccionario = {'uno': [1,3,5,7,9],
                   'dos': [2,4,6,8,10]}
    df = pd.DataFrame(diccionario)
    df
    ```
    ```

Introducción a Estructuras de Datos - `DataFrame` a partir de un `ndarray` estructurado o de registro

- Es similar a un diccionario de arrays:
  - ```
    datos = np.zeros((2, ), dtype=[('A', 'i4'), ('B', 'f4'),
    ('C', 'a10')])
    datos[:] = [(1, 2., 'Hola'), (2, 3., 'Mundo')]
    df = pd.DataFrame(datos)
    df
    ```
    ```

Introducción a Estructuras de Datos - `DataFrame` a partir de una lista de diccionarios

- Es similar a un diccionario de arrays:
  - ```
    diccionario = [{'uno': 1, 'dos': 2}, {'uno': 1, 'dos': 2, 'tres': 3}]
    df = pd.DataFrame(diccionario)
    df
    ```

Ojo →

Índice	Tipo	Tamaño	
0	dict	2	{'uno':1, 'dos':2}
1	dict	3	{'uno':1, 'dos':2, 'tres':3}



Índice	uno	dos	tres
0	1	2	nan
1	1	2	3
    ```

Introducción a Estructuras de Datos - `DataFrame` a partir del método `from_dict`

- `from_dict` toma un diccionario de diccionarios o un diccionario de secuencias tipo array y devuelve un `DataFrame`.
- Funciona como el constructor del `DataFrame`, excepto por el parámetro de orientación que es "columnas" por defecto, pero que puede ser configurado como "índice" para usar las teclas de dictado como etiquetas de fila:
  - ```
    diccionario = {'A':[1, 2, 3], 'B':[4, 5, 6]}
    df = pd.DataFrame.from_dict(diccionario)
    df
    ```



- El argumento `orient` cambia la interpretación del diccionario (`orient = 'columns' u orient='index'`)
    ```

Introducción a Estructuras de Datos - DataFrame a partir del método `from_records`

- `from_records` toma una lista de tuplas o un `ndarray` con `dtype` estructurado
- Funciona de manera análoga al constructor normal de `DataFrame`, excepto que el índice de `DataFrame` resultante puede ser un campo específico del `dtype` estructurado.
  - ```
    datos = np.zeros((2, ), dtype=[('A', 'i4'), ('B', 'f4'),
    ('C', 'a10')])
    datos[:] = [(1, 2., 'Hola'), (2, 3., 'Mundo')]
    df = pd.DataFrame.from_records(datos, index='C')
    df
    ```
- Sólo admite `orient='columns'`
    ```

3. Manipulación de Estructuras de Datos

Manipulación de Estructuras de Datos - Introducción

- Pandas ahora soporta dos tipos de indexación multieje:
  - `.loc` se basa principalmente en la etiqueta, pero también puede utilizarse con una matriz booleana. .loc elevará el `KeyError` cuando los elementos no se encuentren. Las entradas permitidas son:
    - Una sola etiqueta
    - Una lista o conjunto de etiquetas
    - Un subconjunto de un objeto
    - Una matriz booleana
    - Una función con un argumento (la llamada `Series` o `DataFrame`) y que devuelve una salida válida para la indexación (una de las anteriores).
  - `.iloc` se basa principalmente en la posición entera (desde 0 hasta la longitud-1 del eje), pero también puede utilizarse con un arreglo booleano. .iloc aumentará el IndexError si un indexador solicitado está fuera de los límites, excepto los indexadores de corte que permiten la indexación fuera de los límites. (esto se ajusta a la semántica de rebanadas de Python/NumPy). Las entradas permitidas son:
    - Un número entero
    - Una lista o conjunto de números enteros
    - Un subconjunto de un objeto
    - Una matriz booleana
    - Una función con un argumento (la llamada `Series` o `DataFrame`) y que devuelve una salida válida para la indexación (una de las anteriores).

Manipulación de Estructuras de Datos - Operador []

- La selección más básica es el operador [ ]:
  - ```
    diccionario = [{'uno': 1, 'dos': 2}, {'uno': 1, 'dos': 2, 'tres': 3}]
    df = pd.DataFrame(diccionario)
    df['uno']
    ```
- Se puede pasar una lista de columnas a [ ] para seleccionar las columnas en ese orden. Si una columna no está contenida en el DataFrame, se planteará una excepción. También se pueden establecer múltiples columnas de esta manera:
  - ```
    diccionario = [{'uno': 1, 'dos': 2}, {'uno': 1, 'dos': 2, 'tres': 3}]
    df = pd.DataFrame(diccionario)
    df[['uno', 'dos']]
    ```

Manipulación de Estructuras de Datos - Operador [ ]

- Se puede acceder a un índice en una Serie o columna en un DataFrame directamente como un atributo:
  - ```
    datos = [1, 2, 3, 4, 5]
    indice = ['a','b','c','d','e']
    serie = pd.Series(datos , index=indice)
    serie.a
    ```
- También se puede modificar su valor
  - ```
    datos = [1, 2, 3, 4, 5]
    indice = ['a','b','c','d','e']
    serie = pd.Series(datos , index=indice)
    serie.a = 10
    ```

Manipulación de Estructuras de Datos - Operador [ ]

- Para obtener subconjuntos con Series se puede utilizar el operador [ ], la sintaxis funciona exactamente como con un ndarray, devolviendo el subconjunto de los valores y las etiquetas correspondientes:
  - ```
    datos = [1, 2, 3, 4, 5]
    indice = ['a','b','c','d','e']
    serie = pd.Series(datos , index=indice)
    serie[3:]
    serie[::2]
    ```
- Con los DataFrames, el operador [ ] corta las filas:
  - ```
    diccionario = [{'uno': 1, 'dos': 2},
                   {'uno': 1, 'dos': 2, 'tres': 3},
                   {'uno': 1, 'dos': 2, 'tres': 3}]
    df = pd.DataFrame(diccionario)
    df[:-1]
    df[::-1] # ordenar en sentido inverso
    ```

Manipulación de Estructuras de Datos - Indexación basada en etiquetas

- Pandas proporciona un conjunto de métodos para tener una indexación basada puramente en etiquetas
- Se trata de un protocolo basado en la inclusión estricta
- Cada etiqueta que se pida debe estar en el índice, o se levantará un KeyError
- Cuando se corta, se incluyen tanto el límite de inicio como el límite de parada, si está presente en el índice
- Los números enteros son etiquetas válidas, pero se refieren a la etiqueta y no a la posición
  - ```
    datos = [1, 2, 3, 4, 5]
    indice = ['a','b','c','d','e']
    serie = pd.Series(datos , index=indice)
    serie.loc['b']
    serie.loc['c'] = 13
    ```
    ```

Manipulación de Estructuras de Datos - Indexación basada en etiquetas

- Con un DataFrame
  - ```
    diccionario = [{'uno': 1, 'dos': 2},
                   {'uno': 1, 'dos': 2, 'tres': 3},
                   {'uno': 1, 'dos': 2, 'tres': 3}]
    indice = ['a','b','c']
    df = pd.DataFrame(diccionario, index=indice)
    df.loc['a']
    df.loc['a':]
    df.loc['a', 'uno']
    df.loc['b':, 'uno']
    ```
    ```

Manipulación de Estructuras de Datos - Indexación basada en etiquetas

- Mediante un array de booleanos
  - ```
    diccionario = [{'uno': 1, 'dos': 2},
                   {'uno': 1, 'dos': 2, 'tres': 3},
                   {'uno': 1, 'dos': 2, 'tres': 3}]
    indice = ['a','b','c']
    df = pd.DataFrame(diccionario, index=indice)
    df.loc['b'] > 1
    df.loc['b'][df.loc['b'] > 1]
    ```
    ```

Manipulación de Estructuras de Datos - Indexación basada en posición

- Pandas proporciona un conjunto de métodos para obtener una indexación basada puramente en números enteros
- La semántica es similar a la selección de subconjuntos de Python y NumPy
- Es una indexación basada en 0
- Al rebanar, se incluye el límite de inicio, mientras que el límite superior se excluye
  - ```
    datos = [1, 2, 3, 4, 5]
    indice = ['a','b','c','d','e']
    serie = pd.Series(datos , index=indice)
    serie.iloc[2]
    serie.iloc[2:]
    serie.iloc[2:] = 13
    ```
    ```

Manipulación de Estructuras de Datos - Indexación basada en posición

- Con un DataFrame:
  - ```
    diccionario = [{'uno': 1, 'dos': 2},
                   {'uno': 1, 'dos': 2, 'tres': 3},
                   {'uno': 1, 'dos': 2, 'tres': 3}]
    indice = ['a','b','c']
    df = pd.DataFrame(diccionario, index=indice)
    df.iloc[:1]
    df.iloc[:1, 1:]
    ```
    ```

Manipulación de Estructuras de Datos - Método sample()

- Para seleccionar de manera aleatoria de filas o columnas de una Serie o DataFrame se utiliza el método `sample()`
- El método muestrea las filas por defecto, y acepta un número específico de filas/columnas a devolver, o un porcentaje (fracción) de filas.
  - ```
    datos = [1, 2, 3, 4, 5]
    indice = ['a','b','c','d','e']
    serie = pd.Series(datos , index=indice)
    serie.sample()
    serie.sample(n=2)
    serie.sample(frac=.5)
    ```
    ```

Manipulación de Estructuras de Datos - Método where()

- A diferencia de la selección de valores de una Serie con un vector booleano, que generalmente devuelve un subconjunto de los datos, se puede utilizar el método where en Series y DataFrame para garantizar que la salida de la selección tiene la misma forma que los datos originales:
  - ```
    datos = [1, 2, 3, 4, 5]
    indice = indice = ['a','b','c','d','e']
    serie = pd.Series(datos , index=indice)
    serie[serie > 2]
    serie.where(serie > 2)
    ```
- Además, **where** toma un argumento opcional **other** para el reemplazo de los valores donde la condición es Falsa:
  - ```
    serie.where(serie > 2, serie + 2)
    ```

Manipulación de Estructuras de Datos - Método query()

- Los objetos DataFrame tienen un método `query()` que permite la selección mediante una expresión
  - ```
    diccionario = [{'uno': 1, 'dos': 2},
                   {'uno': 10, 'dos': 20, 'tres': 30},
                   {'uno': 100, 'dos': 200, 'tres': 300}]
    indice = ['a','b','c']
    df = pd.DataFrame(diccionario, index=indice)
    df.query('uno >= 10 and dos < 200')
    ```
    ```

4. Datos de tipo texto

Datos de tipo texto - Introducción

- En pandas hay dos formas de almacenar texto:
  - object-dtype NumPy array
  - StringDtype
- El tipo StringDtype ha de ser definido explicitamente, por defecto el tipo es object:
  - `pd.Series(list('abcde'), dtype='string')`
- O modificado después de creado:
  - ```
    serie = pd.Series(list('abcde'))
    serie.astype('string')
    ```
- Para acceder a las operaciones de cadenas de texto, utilizaremos el operador .str:
  - ```
    serie = pd.Series(list('abcde'), dtype='string')
    serie.str.upper()
    serie.str.len()
    serie.str.split('_')
    ```

Datos de tipo texto - Método extract()

- El método extract() acepta una expresión regular con al menos un grupo de captura
- La extracción de una expresión regular con más de un grupo devuelve un DataFrame con una columna por grupo.
- Los elementos que no coinciden regresan a una fila llena de NaN
- El dtype del resultado es siempre objeto, incluso si no se encuentra ninguna coincidencia y el resultado sólo contiene NaN
  - ```
    serie = pd.Series(['a1', 'b2', 'c3'], dtype="string")
    serie.str.extract(r'([ab])(\d)')
    ```
  - ```
    serie = pd.Series(['a1', 'b2', 'c3'], dtype="string")
    serie.str.extract(r'(?P<letra>[ab])(?P<digito>\d)', expand=False)
    ```

Datos de tipo texto - Método contains(), match(), fullmatch()

- En Pandas, en una columna de tipo string se puede comprobar si contiene un patrón con el método contains:
  - ```
    patron = r'[ab][\d]'
    serie = pd.Series(['a1', 'b2', 'c3'], dtype="string")
    serie.str.contains(patron)
    ```
- También si coincide un determinado patrón:
  - ```
    patron = r'[ab][\d]'
    serie = pd.Series(['a1', 'b2', 'c3'], dtype="string")
    serie.str.match(patron)
    ```
- O si coincide de manera completa:
  - ```
    patron = r'[ab][\d]'
    serie = pd.Series(['a1', 'b2', 'c3'], dtype="string")
    serie.str.fullmatch(patron)
    ```
    ```

5. Datos de tipo categórico

Manipulación de Estructuras de Datos - Introducción

- En pandas las categorías son de tipo CategoricalDtype
- El tipo de una categoría se describe completamente por
  - categories: una secuencia de valores únicos y sin valores perdidos
  - ordered: un booleano
- En Pandas se puede definir una Serie o la columna de un DataFrame de tipo categórico.
- Se puede realizar especificando el tipo en la construcción:
  - `pd.Series(list('abcde'), dtype='category')`
- O modificado después de creado:
  - ```
    serie = pd.Series(list('abcde'))
    serie.astype('category')
    ```
- Para acceder a los métodos de las categorías se utiliza el operador .cat
  - ```
    serie = pd.Series(list('abcde'), dtype='category')
    serie.cat.categories
    ```

6. Datos de tipo temporal

Datos de tipo temporal - Introducción

- En pandas las fechas y horas son, principalmente, de tipo Timestamp
- Las colecciones de Timestamp son de tipo DatetimeIndex
- La definición de una serie de Timestamp se realiza de la siguiente forma
  - ```
    fechas = [pd.Timestamp('2020-09-01'),
              pd.Timestamp('2020-09-02'),
              pd.Timestamp('2020-09-03')]
    serie = pd.Series(fechas)
    serie
    ```
  - ```
    serie = pd.Series(np.random.rand(3), index=fechas)
    serie
    ```
- Para acceder a las funciones específicas de los Timestamp utilizamos el operador .dt
  - ```
    fechas = [pd.Timestamp('2020-09-01'),
              pd.Timestamp('2020-09-02'),
              pd.Timestamp('2020-09-03')]
    serie = pd.Series(fechas)
    ```
  - ```
    serie.dt.year
    ```

Germán Alonso Lascurain
germanalonso@opendeusto.es