03_Series_III

November 28, 2023

1 Pandas: Series (III)

Lo primero empieza importando pandas, esta vez sin numpy:

[6]:
```
import pandas as pd
```

1.0.1 Series como diccionario especializado

Puedes pensar en una `Series` de Pandas como una especialización de un diccionario de Python. Un diccionario es una estructura que asigna claves arbitrarias a un conjunto de valores arbitrarios, y una `Series` es una estructura que asigna claves tipificadas a un conjunto de valores tipificados. Esta tipificación es importante: al igual que el código compilado de tipo específico detrás de un array de NumPy lo hace más eficiente que una lista de Python para ciertas operaciones, la información de tipo de una `Serie` de Pandas la hace mucho más eficiente que los diccionarios de Python para ciertas operaciones, y nos permitira tratar los dataframe o tablas mas eficicentes

La analogía entre `Series` y diccionario se puede hacer aún más clara construyendo un objeto `Series` directamente desde un diccionario de Python:

[8]:
```
population_dict ={"California": 38332521,
                  "Texas": 26448123,
                  "New York": 19651127,
                  "Florida": 19552860,
                  "Illinois":22882135
                  }
population = pd.Series(population_dict)
population
```

[8]:
```
California    38332521
Texas         26448123
New York      19651127
Florida       19552860
Illinois      22882135
dtype: int64
```

[10]:
```
population.values
```

[10]:
```
array([38332521, 26448123, 19651127, 19552860, 22882135], dtype=int64)
```

[11]:
```
population.index
```

[11]:
```
Index(['California', 'Texas', 'New York', 'Florida', 'Illinois'],
dtype='object')
```

Por defecto, se creará una `Series` donde el índice se extrae de las claves ordenadas. A partir de aquí, se puede realizar el típico acceso a los elementos de estilo diccionario:

[12]:
```
population["Illinois"]
```

[12]:
```
22882135
```

Sin embargo, a diferencia de un diccionario, `Series` también admite operaciones de tipo matriz, como el corte o *slicing*: NOTA.- se puede hacer slicing en las series con las claves(index) y ademas resulta lamativo que al ejemeplo de mas abajo, incluye todos los valores entre California y Florida , ambos inclusive (que no ocurria ni en listas ni en arrays y no existia este metodo en dict)

[13]:
```
population["California":"Florida"]
```

[13]:
```
California    38332521
Texas         26448123
New York      19651127
Florida       19552860
dtype: int64
```

1.0.2 Construyendo objetos Series

Ya hemos visto algunas formas de construir una `Series` de Pandas desde cero; todas ellas son alguna versión de lo siguiente:

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

donde `índice` es un argumento opcional, y `datos` puede ser una de muchas entidades.

Por ejemplo, `data` puede ser una lista o un array de NumPy, en cuyo caso `index` es por defecto una secuencia de enteros:

[16]:
```
serie_ejemplo = pd.Series([2,4,5])
serie_ejemplo.index
```

[16]:
```
RangeIndex(start=0, stop=3, step=1)
```

`data` puede ser un escalar, que se repite para llenar el índice especificado:NOTA.- repite el valor tantas veces como el unico indice que tiene (index) , que tb se puede usar a partir de una funcion

[18]:
```
serie_repetida = pd.Series("Casa", index =[100,200,300])
serie_repetida
```

[18]:
```
100    Casa
200    Casa
300    Casa
dtype: object
```

como hemos visto en la parta inicial, `data` puede ser un diccionario, en el que `index` es por defecto las claves ordenadas del diccionario:

[19]:
```
serie_dict = pd.Series({2:"a", 4:"b",6:"c"}) #las claves del diccionario los␣
 ↪convierte en nuestro indice de series
serie_dict
```

[19]:
```
2    a
4    b
6    c
dtype: object
```

En cada caso, el índice puede establecerse explícitamente si se prefiere un resultado diferente:

[20]:
```
serie_dict_alt = pd.Series({2:"a", 4:"b",6:"c"},# aqui establecemos que nos␣
 ↪quedamos solo con una parte de los indices
                           index =[2,6])

serie_dict_alt
```

[20]:
```
2    a
6    c
dtype: object
```

Observe que, en este caso, la `Series` se rellena sólo con las claves identificadas explícitamente y en el orden especificado

[ ]:
```

```