07_Seleccion_en_Series_I

November 28, 2023

1 Indexado, Selección y Slicing en Series (I)

Hemos visto los métodos y herramientas para acceder, establecer y modificar valores en arrays de NumPy. Estos incluyen la indexación (por ejemplo, `arr[2, 1]`), el slicing (por ejemplo, `arr[:, 1:5]`), la mascara (por ejemplo, `arr[arr > 0]`) [la máscara es el acceso mediante la creación implicita de unal lista de valores booleanos] y además hemos visto combinaciones de los mismos.

Ahora veremos medios similares para acceder y modificar valores en los objetos Pandas `Series` y `DataFrame`. Si has utilizado los patrones de NumPy, los patrones correspondientes en Pandas te resultarán muy familiares, aunque hay algunas peculiaridades que debes tener en cuenta.

Empezaremos con el caso simple del objeto `Series`.

1.1 Introducción

Como estudiamos en las sesiones dedicadas a `Series`, un objeto **`Series` actúa en muchos aspectos como un array unidimensional de NumPy, y en muchos aspectos como un diccionario estándar de Python.** Si tenemos en cuenta estas dos analogías superpuestas, nos ayudará a entender los patrones de indexación y selección de datos en estos objetos.

1.1.1 Series como un diccionario: el objeto series contruye un mapeo de una colecion de claves a una coleccion de valores

Recordemos que al igual que un diccionario, el objeto `Series` proporciona un mapeo de una colección de claves a una colección de valores:

[2]:
```
import pandas as pd
```

```
data= pd.Series([1,2,4,5,6], index=["a","b","c","d","e"])
data
```

[2]:
```
a    1
b    2
c    4
d    5
e    6
dtype: int64
```

[3]:
```
data["c"]
```

[3]:
```
4
```

Como ya habíamos adelantado en las sesiones sobre `Series`, podemos utilizar expresiones y métodos de Python tipo diccionario para examinar las claves/índices y los valores:

[4]:
```
print("h" in data)
print ("a" in data)
```

```
False
True
```

En ese sentido, el index de una serie es como las keys de un diccionario

[5]:
```
data.keys()
```

[5]:
```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

[7]:
```
data.index
```

[7]:
```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

las series panda se pueden recorrer con un objeto items

[9]:
```
for indice, valor in data.items():
    print("index:",indice,"valor:", valor)
```

```
index: a valor: 1
index: b valor: 2
index: c valor: 4
index: d valor: 5
index: e valor: 6
```

Los objetos `Series` pueden incluso modificarse con una sintaxis similar a la de un diccionario. Al igual que se puede ampliar un diccionario asignando una nueva clave, se puede ampliar una `Serie` asignando un nuevo valor de índice:

[11]:
```
data ["z"] = 23
data["m"] = 22
data
```

[11]:
```
a     1
b     2
c     4
d     5
e     6
z    23
m    22
dtype: int64
```

OJO: fijate que el indice no esta ordenado, sino que tiene el orden como han sido insertados

1.1.2 Series como un array unidimensional

Una `Series` permite una interacción similar a un diccionario y proporciona además selección de elementos al estilo de un array de Numpy a través de los mismos mecanismos básicos que los arrays de NumPy – es decir, *cortes (slices)*, *enmascaramiento (masking)* y *indexación sofisticada (fancy indexing): puedo seleccionar los datos de mi serie indicandole una lista de indices. Ejemplos de estos son los siguientes, algunos ya visto en sesiones anteriores:

[17]:
```
# Corte o slicing por indice explícito (ya visto)
data["a":"d"] #ordenados por la forma que han ido entrando por el indice
```

[17]:
```
a    1
b    2
c    4
d    5
dtype: int64
```

[24]:
```
# Corte o slicing por indice implicito o posicional
data[-3:] # desde la antepenultima posicion hasta el final
data[0:1] # da como resultado el elemento cero pero no el elemento 1
```

[24]:
```
a    1
dtype: int64
```

la diferencia entre el slicing por indice explicito o por indice implicito o posicional es que en el primero pones inicio y fin, entrando dento de la serie los extremos inclusive. Sin embargo en las implicitas iniciamos y no le ponemos limite ym en caso de hacerlo, no llegara nunca a dar el valor del extremo de cierre

[18]:
```
# masking o enmascaramiento (Selección a través de crear booleanos)
data[( data >3) & (data < 8)]
```

[18]:
```
c    4
d    5
e    6
dtype: int64
```

[19]:
```
data
```

[19]:
```
a     1
b     2
c     4
d     5
e     6
z    23
m    22
dtype: int64
```

[21]:
```
condicion = ( data >3) & (data < 8) # sin vas a combinar condiciones es␣
 ↪necesario ponerlas entre parentesis antes de un "and"o u "or"
condicion # esto lo convierte indexando con cada valor correspòndiente, a un␣
 ↪valor booleano donde se ve si se cumple o no la condicion
```

[21]:
```
a    False
b    False
c     True
d     True
e     True
z    False
m    False
dtype: bool
```

[27]:
```
# fancy indexing
seleccion = ["a","m","z","d"]
data.loc[seleccion]# el loc en las series panda es opcional cuando haces making␣
 ↪pq actua como un diccionario pero es interesante ir acostumbradnose a ponerlo
```

[27]:
```
a     1
m    22
z    23
d     5
dtype: int64
```

Entre ellos, el slicing puede ser la fuente de mayor confusión. **Ten en cuenta que cuando se hace slicing con un índice explícito (es decir, `data['a':'c']`), el índice final se *incluye* en el slicing, mientras que cuando se hace slicing con un índice implícito (es decir, `datos[0:2]`), el índice final se *excluye* del slicing.**