Python para Análisis de Datos

Módulo 04

Pandas

Operaciones aritméticas

Cuando realizamos operaciones entre series o dataframes, todas las operaciones son alineadas según las etiquetas (recordemos que en numpy la alineación es por la posición).

Consideremos dos series con la misma cantidad de elementos pero que sólo comparten algunos índices. Si realizamos la suma de las series los elementos, se van a sumar sólo si comparten el mismo índice.

En el caso de que haya índices que estén presentes en una serie pero no en la otra, los elementos se van a completar con `NaN` de modo que la nueva serie tenga todos los índices de ambas series.

Operaciones

```
s1
```

```
0     9
1     4
2    10
3     8
4     0
5     1
dtype: int64
```

```
s2
```

```
3    17
4     4
5     1
6     0
7     0
8     0
dtype: int64
```

```
s1 + s2
```

```
0     NaN
1     NaN
2     NaN
3    25.0
4     4.0
5     2.0
6     NaN
7     NaN
8     NaN
dtype: float64
```

Alineación

Con dataframes la alineación sucede tanto en filas como en columnas.

```
df1
```

	A	B	C	D
0	9	11	2	10
1	4	4	18	17
2	10	12	0	0
3	8	8	0	2
4	0	3	18	3
5	1	1	10	2

```
df2
```

	C	D	E	F
3	11	5	17	8
4	12	13	4	2
5	19	4	1	9
6	19	5	0	12
7	1	12	0	0
8	1	0	0	8

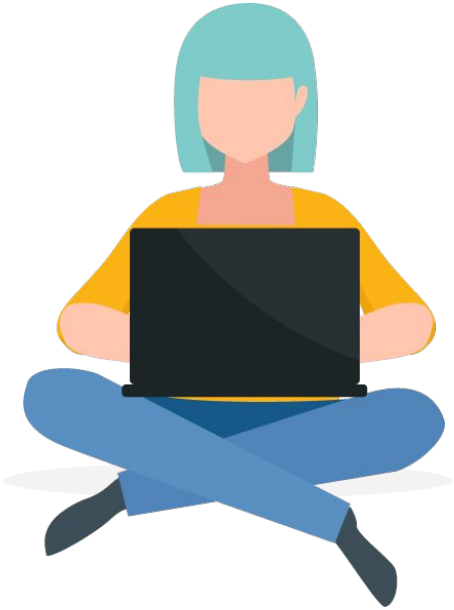
```
df1 + df2
```

	A	B	C	D	E	F
0	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	11.0	7.0	NaN	NaN
4	NaN	NaN	30.0	16.0	NaN	NaN
5	NaN	NaN	29.0	6.0	NaN	NaN
6	NaN	NaN	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN	NaN	NaN

Operaciones

Podemos operar entre dos columnas de un dataframe. En este caso se comparte el índice, por lo que todos los datos se encuentran alineados. Sólo si alguno de los valores es `NaN` el resultado correspondiente también lo será.

Estas operaciones generan una nueva serie, que se puede usar para crear una nueva columna en el dataframe o para reemplazar una existente. Esto se logra con una asignación directa.



Operaciones

```
df1
```

	A	B	C	D
0	9	11.0	2	10
1	4	4.0	18	17
2	10	12.0	0	0
3	8	8.0	0	2
4	0	NaN	18	3
5	1	NaN	10	2

```
df1["A"] + df1["B"]
```

```
0    20.0
1     8.0
2    22.0
3    16.0
4     NaN
5     NaN
dtype: float64
```

Operaciones

```
# creación de nueva columna
df1["E"] = df1["A"] + df1["B"]
df1
```

	A	B	C	D	E
0	9	11.0	2	10	20.0
1	4	4.0	18	17	8.0
2	10	12.0	0	0	22.0
3	8	8.0	0	2	16.0
4	0	NaN	18	3	NaN
5	1	NaN	10	2	NaN

```
# reasignación de columna existente
df1["E"] = df1["C"] + df1["D"]
df1
```

	A	B	C	D	E
0	9	11.0	2	10	12
1	4	4.0	18	17	35
2	10	12.0	0	0	0
3	8	8.0	0	2	2
4	0	NaN	18	3	21
5	1	NaN	10	2	12

Métodos asociados

Todos los operadores tienen métodos asociados que permiten realizar la misma operación. Sin embargo, los métodos ofrecen la posibilidad de controlar ciertos parámetros, dando más flexibilidad a la hora de hacer las operaciones.

Por ejemplo, el parámetro `fill_value` completa los `NaN` existentes en los objetos antes de la operación. También utiliza este valor para cualquier alineación que sea necesaria (en caso de que no exista la etiqueta en una de las estructuras).

Operador	+	-	*	/	//	%	**
Método	add	sub	mul	div	floordiv	mod	pow

Métodos asociados

```
s1.add(s2)
```

```
0     NaN
1     NaN
2     NaN
3    25.0
4     4.0
5     2.0
6     NaN
7     NaN
8     NaN
dtype: float64
```

```
s1.add(s2, fill_value = 0)
```

```
0     9.0
1     4.0
2    10.0
3    25.0
4     4.0
5     2.0
6     0.0
7     0.0
8     0.0
dtype: float64
```

Métodos asociados

Los métodos asociados también permiten controlar cómo se realiza el **broadcasting** entre dataframes y series.

En este caso la alineación por defecto es por columnas, es decir que al hacer la operación se van a comparar los índices de la serie con las columnas del dataframe. Si sumamos directamente un dataframe con una de sus columnas (y los índices de las filas son distintos de las columnas), ningún dato se va a encontrar alineado. Para controlar que la alineación se haga por filas podemos usar el método con el parámetro `axis = index.`

Métodos asociados

```
df1 + df1["A"]
# equivalente a
# df1.add(df1["A"], axis="columns")
```

	A	B	C	D	E	0	1	2	3	4	5
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
df1.add(df1["A"], axis="index")
```

	A	B	C	D	E
0	18	20.0	11	19	21
1	8	8.0	22	21	39
2	20	22.0	10	10	10
3	16	16.0	8	10	10
4	0	NaN	18	3	21
5	2	NaN	11	3	13

Comparaciones

También existen los **operadores de comparación** y sus métodos correspondientes. El comportamiento es similar al de los operadores y métodos aritméticos. Estas operaciones producen estructuras de booleanos que luego se pueden utilizar para hacer filtros.

Para producir filtros más complejos tenemos que ser capaces de combinar comparaciones. Los operadores `and`, `or` y `not` no funcionan con series o dataframes. Para hacer operaciones lógicas elemento a elemento hay que usar los operadores `&`, `|` y `~`.

Es importante usar paréntesis para agrupar correctamente las operaciones ya que no tienen el mismo orden de precedencia que los operadores habituales.

Comparaciones

```
df2
```

	C	D	E	F
3	11	5	17	8
4	12	13	4	2
5	19	4	1	9
6	19	5	0	12
7	1	12	0	0
8	1	0	0	8

```
df2[(df2["C"]>5) & (df2["E"]<15)]
```

	C	D	E	F
4	12	13	4	2
5	19	4	1	9
6	19	5	0	12

¡Muchas gracias!

¡Sigamos trabajando!