El futuro digital es de todos

MinTIC

Pandas – Selección de datos en Series

OPERADO POR:

Universidad Tecnológica de Pereira

Mision TIC 2022

RUTA DE APRENDIZAJE 1

Pandas - Selección de datos en Series

Creación de dataframes:

El constructor de dataframes es "pandas.DataFrame". Acepta cuatro parámetros principales:

- **data:** estructura de datos ndarray (array NumPy), diccionario u otro dataframe
- **index:** índice a aplicar a las filas. Si no se especifica, se asignará uno por defecto formado por números enteros entre 0 y n-1, siendo n el número de filas del dataframe.
- **columns:** etiquetas a aplicar a las columnas. Al igual que ocurre con el índice de filas, si no se añade se asignará uno automático formado por números enteros entre 0 y n-1, siendo n el número de columnas.
- **dtype:** tipo a aplicar a los datos. Solo se permite uno. Si no se especifica, se infiere el tipo de cada columna a partir de los datos que contengan.

Los valores de los índices de filas y columnas no tienen por qué ser necesariamente distintos.

Veamos algunas de las estructuras a partir de las que es posible construir un dataframe:

Utilizando un diccionario:

```
In [45]: elementos = {
             "Numero atómico":[1, 6, 47, 88],
             "Masa atómica":[1.008, 12.011, 107.87, 226],
             "Familia":["No metal", "No metal", "Metal", "Metal"]
         }
         elementos

Out[45]: {'Numero atómico': [1, 6, 47, 88],
          'Masa atómica': [1.008, 12.011, 107.87, 226],
          'Familia': ['No metal', 'No metal', 'Metal', 'Metal']}
```

Y creamos el **dataframe** con él como primer argumento:

```
In [46]: tabla_periodica = pd.DataFrame(elementos)
         tabla_periodica
```

Out[46]:

	Numero atómico	Masa atómica	Familia
0	1	1.008	No metal
1	6	12.011	No metal
2	47	107.870	Metal
3	88	226.000	Metal

El dataframe se ha creado situando las claves del diccionario como etiquetas de columnas y las listas asociadas a cada clave como columnas del dataframe. Al no haber especificado un índice de filas, éste ha tomado valores por defecto (0, 1, 2 y 3).

A continuación repetimos la misma operación especificando las etiquetas tanto para filas como para columnas, utilizando los parámetros index y columns, respectivamente:

```
In [50]: unidades = pd.DataFrame(unidades_Datos, index = [2015, 2016, 2017], columns = ["Ag", "Au", "Cu", "Pt"])
         unidades
```

Out[50]:

	Ag	Au	Cu	Pt
2015	2	5	3	2
2016	4	6	7	2
2017	3	2	4	1

Utilizando diferentes diccionarios

También podemos partir de un conjunto de diccionarios, cada uno definiendo el contenido de lo que será una fila del dataframe:

In [51]:
```
unidades_2015 = {"Ag":2, "Au":5, "Cu":3, "Pt":2}
unidades_2016 = {"Ag":4, "Au":6, "Cu":7, "Pt":2}
unidades_2017 = {"Ag":3, "Au":2, "Cu":4, "Pt":1}

unidades = pd.DataFrame([unidades_2015, unidades_2016, unidades_2017],
                        index = [2015, 2016, 2017])
unidades
```

Out[51]:

	Ag	Au	Cu	Pt
2015	2	5	3	2
2016	4	6	7	2
2017	3	2	4	1

Los diccionarios deberán compartir el mismo conjunto de claves que se interpretarán como etiquetas de columnas. Si las etiquetas no coinciden, se crearán todas las columnas pero se asignarán **NaN** a los valores desconocidos:

En este ejemplo, el año 2017 tiene una clave, Pb, que no existe en los otros dos diccionarios. Y este mismo año carece de la clave Au que sí se encuentra en los otros dos. Vemos cómo los datos no coincidentes se han rellenado con NaN.

Inspección de la información

Normalmente, una vez hemos cargado un bloque de datos en una serie o un dataframe, lo primero que haremos será inspeccionarlo para confirmar que los datos cargados son los esperados y que la lectura se ha realizado correctamente. Para esto tenemos los métodos **head, tail** y **sample**, con un comportamiento semejante en series y dataframes, que nos muestran un subconjunto de los datos cargados. Además, los métodos **describe** e **info** nos proporcionan información adicional sobre los datos. Veamos estos métodos por separado.

El método head

Este método, pandas.Series.head para series y pandas.DataFrame.head para dataframes, devuelve los primeros elementos de la estructura (los primeros valores en el caso de una serie y las primeras filas en el caso de un dataframe). Por defecto, se trata de los 5 primeros elementos, pero podemos especificar el número que deseamos como argumento de la función. Por ejemplo, partamos de las siguientes estructuras:

```
In [53]: import pandas as pd
         entradas = pd.Series([11, 18, 12, 16, 9, 16, 22, 28, 31, 29, 30, 12],
                              index = ["ene", "feb", "mar", "abr", "may", "jun", "jul", "ago",
                               "sep", "oct", "nov", "dic"])
         entradas
```

```
Out[53]: ene    11
         feb    18
         mar    12
         abr    16
         may     9
         jun    16
         jul    22
         ago    28
         sep    31
         oct    29
         nov    30
         dic    12
         dtype: int64
```

In [54]:
```
salidas = pd.Series([9, 26, 18, 15, 6, 22, 19, 25, 34, 22, 21, 14],
                    index = ["ene", "feb", "mar", "abr", "may", "jun", "jul", "ago",
                     "sep", "oct", "nov", "dic"])
salidas
```

Out[54]:
```
ene     9
feb    26
mar    18
abr    15
may     6
jun    22
jul    19
ago    25
sep    34
oct    22
nov    21
dic    14
dtype: int64
```

In [55]:
```
almacén = pd.DataFrame({"entradas": entradas, "salidas": salidas})
almacén["neto"] = almacén.entradas - almacén.salidas
almacén
```

Out[55]:

	entradas	salidas	neto
ene	11	9	2
feb	18	26	-8
mar	12	18	-6
abr	16	15	1
may	9	6	3
jun	16	22	-6
jul	22	19	3
ago	28	25	3
sep	31	34	-3
oct	29	22	7
nov	30	21	9
dic	12	14	-2

En este ejemplo estamos mostrando todos los elementos de la estructura pues son apenas 12. En un caso real podemos estar hablando de miles o de millones.

Ahora, para mostrar apenas los primeros elementos de la estructura, ejecutamos el método head:

```
In [56]: entradas.head()

Out[56]: ene    11
         feb    18
         mar    12
         abr    16
         may     9
         dtype: int64
```

```
In [57]: almacén.head()
```

Out[57]:

	entradas	salidas	neto
ene	11	9	2
feb	18	26	-8
mar	12	18	-6
abr	16	15	1
may	9	6	3

El método tail

Los métodos pandas.Series.tail (para series) y pandas.DataFrame.tail (para dataframes) son semejantes a los anteriores, pero muestran los últimos elementos de la estructura. Si no indicamos otra cosa como argumento, serán los 5 últimos elementos los que se muestren:

```
In [58]: entradas.tail()

Out[58]: ago    28
         sep    31
         oct    29
         nov    30
         dic    12
         dtype: int64
```

```
In [59]: almacén.tail()
```

Out[59]:

	entradas	salidas	neto
ago	28	25	3
sep	31	34	-3
oct	29	22	7
nov	30	21	9
dic	12	14	-2

El método sample:

Sin embargo, es frecuente que los datos que hayamos leído estén ordenados según algún criterio, y que el bloque de datos mostrado por los métodos head o tail estén formados por datos muy parecidos. Y en ocasiones nos puede convenir ver datos aleatorios de nuestra estructura. Para esto podemos utilizar los métodos "pandas.Series.sample" para series y "pandas.DataFrame.sample" para dataframes. Al contrario que head o tail, el número de elementos devueltos por defecto es uno, por lo que, si deseamos extraer una muestra mayor, tendremos que indicarlo como primer argumento:

```
In [60]: entradas.sample()
Out[60]: may    9
         dtype: int64

In [61]: almacén.sample(5)
Out[61]:
```

	entradas	salidas	neto
oct	29	22	7
feb	18	26	-8
dic	12	14	-2
jun	16	22	-6
may	9	6	3

El método describe:

El método describe devuelve información estadística de los datos del dataframe o de la serie (de hecho, este método devuelve un dataframe). Esta información incluye el número de muestras, el valor medio, la desviación estándar, el valor mínimo, máximo, la mediana y los valores correspondientes a los percentiles 25% y 75%.
Siguiendo con el ejemplo visto en la sección anterior:

```
In [62]: almacén.describe()
```

Out[62]:

	entradas	salidas	neto
count	12.00000	12.000000	12.000000
mean	19.50000	19.250000	0.250000
std	8.16311	7.641097	5.310795
min	9.00000	6.000000	-8.000000
25%	12.00000	14.750000	-3.750000
50%	17.00000	20.000000	1.500000
75%	28.25000	22.750000	3.000000
max	31.00000	34.000000	9.000000

El método acepta el parámetro percentiles conteniendo una lista (o semejante) de los percentiles a mostrar. También acepta los parámetros include y exclude para especificar los tipos de las características a incluir o excluir del resultado.

El método info:

El método info muestra un resumen de un dataframe, incluyendo información sobre el tipo de los índices de filas y columnas, los valores no nulos y la memoria usada:

```
In [63]: almacén.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 12 entries, ene to dic
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   entradas  12 non-null     int64
 1   salidas   12 non-null     int64
 2   neto      12 non-null     int64
dtypes: int64(3)
memory usage: 684.0+ bytes
```

El método value_counts:

Un método de las series pandas extremadamente útil es pandas.Series.value_counts. Este método devuelve una estructura conteniendo los valores presentes en la serie y el número de ocurrencias de cada uno. Estos valores se muestran en orden decreciente:

```
In [64]: import numpy as np
         s = pd.Series([3, 1, 2, 1, 1, 4, 1, 2, np.nan])
         print(s)
         s.value_counts()

         0    3.0
         1    1.0
         2    2.0
         3    1.0
         4    1.0
         5    4.0
         6    1.0
         7    2.0
         8    NaN
         dtype: float64

Out[64]: 1.0    4
         2.0    2
         3.0    1
         4.0    1
         dtype: int64
```

Como puede apreciarse, por defecto no se incluyen los valores nulos. Este comportamiento puede modificarse haciendo uso del parámetro **dropna**:

```
In [65]: s.value_counts(dropna = False)

Out[65]: 1.0    4
         2.0    2
         3.0    1
         4.0    1
         NaN    1
         dtype: int64
```

En lugar de devolver los valores distintos y el número de ocurrencias, este método también puede agrupar los datos en "bins" y devolver una lista de bins (indicando sus márgenes) con el número de valores en cada uno de ellos. Por ejemplo, si quisiéramos agrupar los valores de la serie anterior en dos bins podríamos hacerlo de la siguiente forma:

```
In [66]: s.value_counts(bins = 2)

Out[66]: (0.996, 2.5]    6
         (2.5, 4.0]      2
         dtype: int64
```

Vemos que se han creados los dos bins, el primero conteniendo los valores entre 0.996 y 2.5 (intervalo abierto por la izquierda y cerrado por la derecha), bin en el que hay 6 valores, y el segundo conteniendo los valores entre 2.5 y 4 (intervalo también abierto por la izquierda y cerrado por la derecha), bin en el que hay 2 valores.

Selección de datos en Series:

Ya se ha comentado que una serie pandas consta de un array de datos y un array de etiquetas (el índice o index). Si al crear la serie no se ha especificado el índice, ya sabemos que se asignará uno implícito por defecto:

```
In [67]: import pandas as pd

         s = pd.Series([10, 20, 30, 40])
         s

Out[67]: 0    10
         1    20
         2    30
         3    40
         dtype: int64
```

Podemos seleccionar los valores haciendo referencia al índice asignado con la misma notación que en un diccionario (la llamada "notación corchetes" o "square bracket notation"):

```
In [68]: print(s[0])
         print(s[2])
```

```
10
30
```

Usando esta sintaxis, si no se ha especificado un índice explícito, los índices negativos no están permitidos.

Si se asignan índices de forma explícita:

```
In [69]: s = pd.Series([10, 20, 30, 40], index = ["a", "b", "c", "d"])
         s
```

```
Out[69]: a    10
         b    20
         c    30
         d    40
         dtype: int64
```

Podemos seleccionar los elementos usando el índice explícito o el implícito:

```
In [70]: print(s["a"],s[0])
10 10

In [71]: print(s["d"],s[3])
40 40
```

Con esta sintaxis, sí está permitido hacer uso de índices negativos para referirnos a los elementos desde el final de la estructura.

Si los índices asignados son números enteros (al igual que las etiquetas del índice implícito), el índice implícito queda desactivado:

```
In [72]: s = pd.Series([10, 20, 30, 40], index = [3, 2, 1, 0])
         s

Out[72]: 3    10
         2    20
         1    30
         0    40
         dtype: int64

In [73]: s[0]

Out[73]: 40
```

Uso de rangos:

Es posible seleccionar rangos de valores. De esta forma, si usamos un rango numérico en una serie en la que hemos definido un índice explícito:

```
In [74]: s = pd.Series([10, 20, 30, 40], index = ["a", "b", "c", "d"])
         s

Out[74]: a    10
         b    20
         c    30
         d    40
         dtype: int64

In [75]: s[1:3]

Out[75]: b    20
         c    30
         dtype: int64
```

Observamos que el rango se interpreta como haciendo referencia al índice implícito, y se incluyen los valores desde el primer índice incluido, hasta el último sin incluir.

Si no se incluye alguno de los límites, el comportamiento es el estándar en Python (si no se incluye el primer valor, se consideran todos los elementos desde el principio, y si no se incluye el último valor, se consideran todos los elementos hasta el final):

```
In [76]: s[1:]
Out[76]: b    20
         c    30
         d    40
         dtype: int64

In [77]: s[:3]
Out[77]: a    10
         b    20
         c    30
         dtype: int64
```

```
In [78]: s["a":"c"]
Out[78]: a    10
         b    20
         c    30
         dtype: int64

In [79]: s[:"c"]
Out[79]: a    10
         b    20
         c    30
         dtype: int64

In [80]: s["b":]
Out[80]: b    20
         c    30
         d    40
         dtype: int64
```

Si se utilizan los índices explícitos en el rango, el comportamiento es ligeramente diferente:

Una posible fuente de confusión viene derivada del hecho de que, usando rangos, es posible hacer referencia tanto a las etiquetas como a los índices numéricos: si utilizamos etiquetas, hacemos referencia a las etiquetas (por supuesto):

```
In [81]: s = pd.Series([10, 20, 30, 40], index = ["a", "b", "c", "d"])

In [82]: s["b":"c"]

Out[82]: b    20
         c    30
         dtype: int64
```

Y, por tanto, si utilizamos números, hacemos referencia a los índices numéricos (¿por supuesto...?):

```
In [83]: s[1:3]

Out[83]: b    20
         c    30
         dtype: int64
```

Método iloc:

El método pandas.Series.iloc permite extraer datos de la serie a partir de los índices implícitos que éstos tienen asignados.

La opción más simple es utilizar como argumento un simple número entero (el primer elemento de la serie recibe el índice cero):

```
In [98]: s = pd.Series([10, 20, 30, 40], index = ["a", "b", "c", "d"])
         s

Out[98]: a    10
         b    20
         c    30
         d    40
         dtype: int64

In [99]: s.iloc[1]

Out[99]: 20

In [100]: s.iloc[0]

Out[100]: 10

In [101]: s.iloc[3]

Out[101]: 40
```

Si el número es negativo, hace referencia al final de la serie (en este caso, el último elemento recibe el índice -1) -y esto tanto si se ha especificado un índice explícito como si no-:

```
In [102]: s.iloc[-1]
Out[102]: 40

In [103]: s.iloc[-4]
Out[103]: 10
```

Una segunda opción es pasar como argumento una lista o array de números, en cuyo caso se devuelven los elementos que ocupan dichas posiciones en el orden indicado en la lista o array:

```
In [104]: s.iloc[[2, 0]]
Out[104]: c    30
          a    10
          dtype: int64
```

También podemos incluir en esta lista números negativos, con la funcionalidad ya comentada:

```
In [ ]: s.iloc[[-2, 0]]
```

Una tercera opción es usar como argumento un rango de números:

```
In [ ]: s.iloc[1:3]
```

Como vemos en el ejemplo anterior, si el rango tiene la forma a:b, se incluyen todos los elementos desde aquel cuyo índice es a (incluido) hasta el que tiene el índice b (sin incluir).

Si no se especifica el primer valor, se consideran todos los elementos desde el principio de la serie:

```
In [ ]: s.iloc[:3]
```

Y, si no se especifica el segundo valor, se consideran todos los elementos hasta el final de la serie:

```
In [ ]: s.iloc[2:]
```

También pueden usarse valores negativos para indicar el comienzo y/o el final del rango:

```
In [ ]: s.iloc[1:-1]
```

El futuro digital es de todos

MinTIC

GRACIAS

OPERADO POR:

Universidad Tecnológica de Pereira

MisiónTIC2022