Pandas: cómo acceder a columnas o filas de un DataFrame, seleccionar subconjuntos

Mathieu Kessler

Departamento de Matemática Aplicada y Estadística
Universidad Politécnica de Cartagena

@kessler_mathieu

Las operaciones más frecuentes implican trabajar con subconjuntos

Esos subconjuntos de un DataFrame pueden ser de diferentes tipos:

- Extraigo un conjunto de columnas relevantes, descartando otras.

```
>>> datos
    x1   x2   x3  y
0  4.2  2.1  2.5  a
1  0.7 -3.7 -2.3  b
2 -2.4 -2.1  4.0  b
3  0.3 -1.5  1.9  c
```

- Me quedo con un subconjunto de filas que cumplen un determinado criterio

```
>>> datos
    x1   x2   x3  y
0  4.2  2.1  2.5  a
1  0.7 -3.7 -2.3  b
2 -2.4 -2.1  4.0  b
3  0.3 -1.5  1.9  c
```

- Una combinación de las dos opciones anteriores

```
>>> datos
    x1   x2   x3  y
0  4.2  2.1  2.5  a
1  0.7 -3.7 -2.3  b
2 -2.4 -2.1  4.0  b
3  0.3 -1.5  1.9  c
```

Extraer columnas

Para extraer un conjunto de columnas relevantes

La manera más sencilla es indicar entre llaves una etiqueta o una lista de las etiquetas de columnas que deseo extraer

Extraer columnas

Para extraer un conjunto de columnas relevantes

La manera más sencilla es indicar entre llaves una etiqueta o una lista de las etiquetas de columnas que deseo extraer

- Para seleccionar solamente una columna:

```
>>> datos['x2']
0    2.1
1   -3.7
2   -2.1
3   -1.5
Name: x2, dtype: float64
```

Extraer columnas

Para extraer un conjunto de columnas relevantes

La manera más sencilla es indicar entre llaves una etiqueta o una lista de las etiquetas de columnas que deseo extraer

- Para seleccionar solamente una columna:

```
>>> datos['x2']
0    2.1
1   -3.7
2   -2.1
3   -1.5
Name: x2, dtype: float64
```

- Para seleccionar varias columnas

```
>>> datos[['x1', 'x2']]
    x1   x2
0  4.2  2.1
1  0.7 -3.7
2 -2.4 -2.1
3  0.3 -1.5
```

Extraer columnas

Para extraer un conjunto de columnas relevantes

La manera más sencilla es indicar entre llaves una etiqueta o una lista de las etiquetas de columnas que deseo extraer

- Para seleccionar solamente una columna:

```
>>> datos['x2']
0    2.1
1   -3.7
2   -2.1
3   -1.5
Name: x2, dtype: float64
```

NOTA: el resultado es un Series

- Para seleccionar varias columnas

```
>>> datos[['x1', 'x2']]
    x1   x2
0  4.2  2.1
1  0.7 -3.7
2 -2.4 -2.1
3  0.3 -1.5
```

Extraer columnas

Para extraer un conjunto de columnas relevantes

La manera más sencilla es indicar entre llaves una etiqueta o una lista de las etiquetas de columnas que deseo extraer

- Para seleccionar solamente una columna:

```
>>> datos['x2']
0    2.1
1   -3.7
2   -2.1
3   -1.5
Name: x2, dtype: float64
```

NOTA: el resultado es un Series

- Para seleccionar varias columnas

```
>>> datos[['x1', 'x2']]
    x1   x2
0  4.2  2.1
1  0.7 -3.7
2 -2.4 -2.1
3  0.3 -1.5
```

NOTA: el resultado es un DataFrame

Extraer filas basándose en un criterio

Para seleccionar filas basándose en un criterio lógico

La manera más sencilla es indicar entre llaves un objeto de tipo vector (lista, array numpy, Series) booleano (True o False)

Extraer filas basándose en un criterio

Para seleccionar filas basándose en un criterio lógico

La manera más sencilla es indicar entre llaves un objeto de tipo vector (lista, array numpy, Series) booleano (True o False)

- Por ejemplo:

```
>>> datos[[False, True, True, False]]
    x1   x2   x3  y
1  0.7 -3.7 -2.3  b
2 -2.4 -2.1  4.0  b
```

Extraer filas basándose en un criterio

Para seleccionar filas basándose en un criterio lógico

La manera más sencilla es indicar entre llaves un objeto de tipo vector (lista, array numpy, Series) booleano (True o False)

- Por ejemplo:

```
>>> datos[[False, True, True, False]]
    x1   x2   x3  y
1  0.7 -3.7 -2.3  b
2 -2.4 -2.1  4.0  b
```

- Podemos proporcionar un Series calculado

```
>>> datos[datos['y'] == 'b']
    x1   x2   x3  y
1  0.7 -3.7 -2.3  b
2 -2.4 -2.1  4.0  b
```

Extraer filas basándose en un criterio

Para seleccionar filas basándose en un criterio lógico

La manera más sencilla es indicar entre llaves un objeto de tipo vector (lista, array numpy, Series) booleano (True o False)

- Por ejemplo:

```
>>> datos[[False, True, True, False]]
    x1   x2   x3  y
1  0.7 -3.7 -2.3  b
2 -2.4 -2.1  4.0  b
```

- Podemos proporcionar un Series calculado

```
>>> datos[datos['y'] == 'b']
    x1   x2   x3  y
1  0.7 -3.7 -2.3  b
2 -2.4 -2.1  4.0  b
```

NOTA: se preservan las etiquetas (index) de las filas

La manera más general de seleccionar filas y/o columnas

Los dos métodos más flexibles y generales para seleccionar filas y/o columnas

- El método `loc` permite seleccionar filas y/o columnas a través de las etiquetas (index o nombres de columnas) o usando vectores booleanos
- El método `iloc` permite seleccionar filas y/o columnas a través de sus posiciones (filas o columnas) o usando vectores booleanos

El método `loc[]`

Se aplica con llaves, separamos la especificación de filas y columnas por una coma. `df.loc[especificacion_filas, especificacion_columnas]`

El método `loc[]`

Se aplica con llaves, separamos la especificación de filas y columnas por una coma. `df.loc[especificacion_filas, especificacion_columnas]`

Recordad el DataFrame `datos`:

```
    x1   x2   x3  y
0  4.2  2.1  2.5  a
1  0.7 -3.7 -2.3  b
2 -2.4 -2.1  4.0  b
3  0.3 -1.5  1.9  c
```

El método `loc[]`

Se aplica con llaves, separamos la especificación de filas y columnas por una coma. `df.loc[especificacion_filas, especificacion_columnas]`

Recordad el DataFrame `datos`:

```
    x1   x2   x3  y
0  4.2  2.1  2.5  a
1  0.7 -3.7 -2.3  b
2 -2.4 -2.1  4.0  b
3  0.3 -1.5  1.9  c
```

- Si no hay especificación para filas, usamos ":" para indicar que las seleccionamos todas:

```
>>> datos.loc[:, ['x1', 'x2', 'x3']]
    x1   x2   x3
0  4.2  2.1  2.5
1  0.7 -3.7 -2.3
2 -2.4 -2.1  4.0
3  0.3 -1.5  1.9
```

El método `loc[]`

- Podemos usar un vector booleano, tanto para filas como para columnas.

```
>>> datos.loc[:, [True, True, True, False]]
    x1   x2   x3
0  4.2  2.1  2.5
1  0.7 -3.7 -2.3
2 -2.4 -2.1  4.0
3  0.3 -1.5  1.9
```

El método `loc[]`

- Podemos usar un vector booleano, tanto para filas como para columnas.

```
>>> datos.loc[:, [True, True, True, False]]
    x1   x2   x3
0  4.2  2.1  2.5
1  0.7 -3.7 -2.3
2 -2.4 -2.1  4.0
3  0.3 -1.5  1.9
```

- O combinar vectores booleanos con especificación de etiquetas

```
>>> datos.loc[datos['y'] == 'b', ['x1', 'x2', 'x3']]
    x1   x2   x3
1  0.7 -3.7 -2.3
2 -2.4 -2.1  4.0
```

El método `iloc[]`

Se aplica con llaves, separamos la especificación de filas y columnas por una coma. Utilizamos sus posiciones para seleccionar filas o columnas. No admite vectores booleanos.

El método `iloc[]`

Se aplica con llaves, separamos la especificación de filas y columnas por una coma. Utilizamos sus posiciones para seleccionar filas o columnas. No admite vectores booleanos.

Recordad el DataFrame `datos`:

```
    x1   x2   x3  y
0  4.2  2.1  2.5  a
1  0.7 -3.7 -2.3  b
2 -2.4 -2.1  4.0  b
3  0.3 -1.5  1.9  c
```

El método `iloc[]`

Se aplica con llaves, separamos la especificación de filas y columnas por una coma. Utilizamos sus posiciones para seleccionar filas o columnas. No admite vectores booleanos.

Recordad el DataFrame `datos`:

```
    x1   x2   x3  y
0  4.2  2.1  2.5  a
1  0.7 -3.7 -2.3  b
2 -2.4 -2.1  4.0  b
3  0.3 -1.5  1.9  c
```

- `>>> datos.iloc[:, [0, 1, 2]]`

```
    x1   x2   x3
0  4.2  2.1  2.5
1  0.7 -3.7 -2.3
2 -2.4 -2.1  4.0
3  0.3 -1.5  1.9
```

El método `iloc[]`

Se aplica con llaves, separamos la especificación de filas y columnas por una coma. Utilizamos sus posiciones para seleccionar filas o columnas. No admite vectores booleanos.

Recordad el DataFrame `datos`:

```
    x1   x2   x3  y
0  4.2  2.1  2.5  a
1  0.7 -3.7 -2.3  b
2 -2.4 -2.1  4.0  b
3  0.3 -1.5  1.9  c
```

- `>>> datos.iloc[:, [0, 1, 2]]`

```
    x1   x2   x3
0  4.2  2.1  2.5
1  0.7 -3.7 -2.3
2 -2.4 -2.1  4.0
3  0.3 -1.5  1.9
```

- `>>> datos.iloc[[1, 2], [0, 1, 2]]`

```
    x1   x2   x3
1  0.7 -3.7 -2.3
2 -2.4 -2.1  4.0
```

El método `loc[]`

El método `loc` trabaja con etiquetas, por lo que es necesario ser cautos si el index de nuestro DataFrame son los enteros. Considerad el DataFrame `datos_2`:

```
    x1   x2   x3  y
3  0.3 -1.5  1.9  c
2 -2.4 -2.1  4.0  b
1  0.7 -3.7 -2.3  b
0  4.2  2.1  2.5  a
```

El método `loc[]`

El método `loc` trabaja con etiquetas, por lo que es necesario ser cautos si el index de nuestro DataFrame son los enteros. Considerad el DataFrame `datos_2`:

```
    x1   x2   x3  y
3  0.3 -1.5  1.9  c
2 -2.4 -2.1  4.0  b
1  0.7 -3.7 -2.3  b
0  4.2  2.1  2.5  a
```

- `datos_2.loc[0, :]` no nos devuelve la primera fila (posición 0) sino la cuarta, que es la que tiene etiqueta "0".

```
>>> datos_2.loc[0,:]
x1    4.2
x2    2.1
x3    2.5
y       a
Name: 0, dtype: object
```

El método `loc[]`

El método `loc` trabaja con etiquetas, por lo que es necesario ser cautos si el index de nuestro DataFrame son los enteros. Considerad el DataFrame `datos_2`:

```
    x1   x2   x3  y
3  0.3 -1.5  1.9  c
2 -2.4 -2.1  4.0  b
1  0.7 -3.7 -2.3  b
0  4.2  2.1  2.5  a
```

- `datos_2.loc[0, :]` no nos devuelve la primera fila (posición 0) sino la cuarta, que es la que tiene etiqueta "0".

```
>>> datos_2.loc[0,:]
x1    4.2
x2    2.1
x3    2.5
y       a
Name: 0, dtype: object
```

- `datos_2.iloc[0,:]` sí nos devuelve la primera fila:

```
>>> datos_2.iloc[0,:]
x1    0.3
x2   -1.5
x3    1.9
y       c
Name: 3, dtype: object
```

Cortes (slices) en DataFrames

Al igual que para listas, es posible especificar cortes (slices) en DataFrames o Series, usando ":". El corte incluye las columnas o filas comprendidas entre los dos extremos de la especificación.

Cortes (slices) en DataFrames

Al igual que para listas, es posible especificar cortes (slices) en DataFrames o Series, usando ":". El corte incluye las columnas o filas comprendidas entre los dos extremos de la especificación.

Considerad el DataFrame datos donde hemos añadido un índice:

```
     x1   x2   x3  y
a1  4.2  2.1  2.5  a
a2  0.7 -3.7 -2.3  b
a3 -2.4 -2.1  4.0  b
a4  0.3 -1.5  1.9  c
```

Cortes (slices) en DataFrames

Al igual que para listas, es posible especificar cortes (slices) en DataFrames o Series, usando ":". El corte incluye las columnas o filas comprendidas entre los dos extremos de la especificación.

Considerad el DataFrame datos donde hemos añadido un índice:

```
     x1   x2   x3  y
a1  4.2  2.1  2.5  a
a2  0.7 -3.7 -2.3  b
a3 -2.4 -2.1  4.0  b
a4  0.3 -1.5  1.9  c
```

- Para `loc`, usamos especificación por etiquetas:

```
>>> datos.loc['a2':'a4', :]
     x1   x2   x3  y
a2  0.7 -3.7 -2.3  b
a3 -2.4 -2.1  4.0  b
a4  0.3 -1.5  1.9  c
```

Cortes (slices) en DataFrames

Al igual que para listas, es posible especificar cortes (slices) en DataFrames o Series, usando ":". El corte incluye las columnas o filas comprendidas entre los dos extremos de la especificación.

Considerad el DataFrame datos donde hemos añadido un índice:

```
     x1   x2   x3  y
a1  4.2  2.1  2.5  a
a2  0.7 -3.7 -2.3  b
a3 -2.4 -2.1  4.0  b
a4  0.3 -1.5  1.9  c
```

- Para `loc`, usamos especificación por etiquetas:

```
>>> datos.loc['a2':'a4', :]
     x1   x2   x3  y
a2  0.7 -3.7 -2.3  b
a3 -2.4 -2.1  4.0  b
a4  0.3 -1.5  1.9  c
```

- Para `iloc`, usamos posiciones

```
>>> datos.iloc[1:2, 0:2]
     x1   x2
a2  0.7 -3.7
```

Cortes (slices) en DataFrames

Al igual que para listas, es posible especificar cortes (slices) en DataFrames o Series, usando ":". El corte incluye las columnas o filas comprendidas entre los dos extremos de la especificación.

Considerad el DataFrame datos donde hemos añadido un índice:

```
     x1   x2   x3  y
a1  4.2  2.1  2.5  a
a2  0.7 -3.7 -2.3  b
a3 -2.4 -2.1  4.0  b
a4  0.3 -1.5  1.9  c
```

- Para `loc`, usamos especificación por etiquetas:

```
>>> datos.loc['a2':'a4', :]
     x1   x2   x3  y
a2  0.7 -3.7 -2.3  b
a3 -2.4 -2.1  4.0  b
a4  0.3 -1.5  1.9  c
```

- Para `iloc`, usamos posiciones

```
>>> datos.iloc[1:2, 0:2]
     x1   x2
a2  0.7 -3.7
```

Para `loc` los dos extremos están incluidos, para `iloc` no está incluido el extremo de la derecha.

Cambiar valores en subconjuntos de un DataFrame

Usando `loc` o `iloc` es posible cambiar los valores en un subconjunto del DataFrame.

Cambiar valores en subconjuntos de un DataFrame

Usando `loc` o `iloc` es posible cambiar los valores en un subconjunto del DataFrame.

Considerad el DataFrame datos:

```
    x1   x2   x3  y
0  4.2  2.1  2.5  a
1  0.7 -3.7 -2.3  b
2 -2.4 -2.1  4.0  b
3  0.3 -1.5  1.9  c
```

Cambiar valores en subconjuntos de un DataFrame

Usando `loc` o `iloc` es posible cambiar los valores en un subconjunto del DataFrame.

Considerad el DataFrame datos:

```
    x1   x2   x3  y
0  4.2  2.1  2.5  a
1  0.7 -3.7 -2.3  b
2 -2.4 -2.1  4.0  b
3  0.3 -1.5  1.9  c
```

Podemos fijar el subconjunto a un único valor:

```
>>> datos.loc[['a2', 'a4'], 'y'] = np.NaN
>>> datos
     x1   x2   x3    y
a1  4.2  2.1  2.5    a
a2  0.7 -3.7 -2.3  NaN
a3 -2.4 -2.1  4.0    b
a4  0.3 -1.5  1.9  NaN
```

Cambiar valores en subconjuntos de un DataFrame

También podemos asignar a un subconjunto un objeto de tipo vector (lista, numpy array, Series o DataFrame) de dimensiones compatibles con el subconjunto.

Recordad el DataFrame datos:

```
    x1   x2   x3  y
0  4.2  2.1  2.5  a
1  0.7 -3.7 -2.3  b
2 -2.4 -2.1  4.0  b
3  0.3 -1.5  1.9  c
```

Cambiar valores en subconjuntos de un DataFrame

También podemos asignar a un subconjunto un objeto de tipo vector (lista, numpy array, Series o DataFrame) de dimensiones compatibles con el subconjunto.

Recordad el DataFrame datos:

```
   x1   x2   x3  y
0  4.2  2.1  2.5  a
1  0.7 -3.7 -2.3  b
2 -2.4 -2.1  4.0  b
3  0.3 -1.5  1.9  c
```

```
>>> datos.loc[['a2', 'a4'], 'y'] = ['YES', 'NO']
>>> datos
     x1   x2   x3    y
a1  4.2  2.1  2.5    a
a2  0.7 -3.7 -2.3  YES
a3 -2.4 -2.1  4.0    b
a4  0.3 -1.5  1.9   NO
```

Cambiar valores en subconjuntos de un DataFrame

Si usamos un Series o DataFrame, los índices deben coincidir

Recordad el DataFrame datos:

```
    x1   x2   x3  y
0  4.2  2.1  2.5  a
1  0.7 -3.7 -2.3  b
2 -2.4 -2.1  4.0  b
3  0.3 -1.5  1.9  c
```

Cambiar valores en subconjuntos de un DataFrame

Si usamos un Series o DataFrame, los índices deben coincidir

Recordad el DataFrame datos:

```
    x1   x2   x3  y
0  4.2  2.1  2.5  a
1  0.7 -3.7 -2.3  b
2 -2.4 -2.1  4.0  b
3  0.3 -1.5  1.9  c
```

```
>>> datos.loc[['a2', 'a4'], 'y'] = pd.Series(['YES', 'NO'])
>>> datos
     x1   x2   x3    y
a1  4.2  2.1  2.5    a
a2  0.7 -3.7 -2.3  NaN
a3 -2.4 -2.1  4.0    b
a4  0.3 -1.5  1.9  NaN
```

Cambiar valores en subconjuntos de un DataFrame

Si usamos un Series o DataFrame, los índices deben coincidir

Recordad el DataFrame datos:

```
    x1   x2   x3  y
0  4.2  2.1  2.5  a
1  0.7 -3.7 -2.3  b
2 -2.4 -2.1  4.0  b
3  0.3 -1.5  1.9  c
```

```
>>> datos.loc[['a2', 'a4'], 'y'] = pd.Series(['YES', 'NO'])
>>> datos
     x1   x2   x3    y
a1  4.2  2.1  2.5    a
a2  0.7 -3.7 -2.3  NaN
a3 -2.4 -2.1  4.0    b
a4  0.3 -1.5  1.9  NaN
```

El Series de la derecha tiene los índices por defecto (0, 1) que no están en el DataFrame de la derecha

Cambiar valores en subconjuntos de un DataFrame

Si usamos un Series o DataFrame, los índices deben coincidir

Recordad el DataFrame datos:

```
    x1   x2   x3  y
0  4.2  2.1  2.5  a
1  0.7 -3.7 -2.3  b
2 -2.4 -2.1  4.0  b
3  0.3 -1.5  1.9  c
```

```
>>> datos.loc[['a2', 'a4'], 'y'] = pd.Series(['YES', 'NO'])
>>> datos
     x1   x2   x3    y
a1  4.2  2.1  2.5    a
a2  0.7 -3.7 -2.3  NaN
a3 -2.4 -2.1  4.0    b
a4  0.3 -1.5  1.9  NaN
```

El Series de la derecha tiene los índices por defecto (0, 1) que no están en el DataFrame de la derecha

```
>>> datos.loc[['a2', 'a4'], 'y'] = pd.Series(['Y', 'N'], index=['a2', 'a4'])
>>> datos
     x1   x2   x3  y
a1  4.2  2.1  2.5  a
a2  0.7 -3.7 -2.3  Y
a3 -2.4 -2.1  4.0  b
a4  0.3 -1.5  1.9  N
```

Cambiar valores en subconjuntos de un DataFrame

A veces se usa indexación en cadena en lugar de usar `loc`: Recordad el DataFrame datos:

```
     x1   x2   x3  y
a1  4.2  2.1  2.5  a
a2  0.7 -3.7 -2.3  b
a3 -2.4 -2.1  4.0  b
a4  0.3 -1.5  1.9  c
```

Cambiar valores en subconjuntos de un DataFrame

A veces se usa indexación en cadena en lugar de usar `loc`: Recordad el DataFrame datos:

```
     x1   x2   x3  y
a1  4.2  2.1  2.5  a
a2  0.7 -3.7 -2.3  b
a3 -2.4 -2.1  4.0  b
a4  0.3 -1.5  1.9  c
```

```
>>> datos[~(datos['y'] == 'b')]['y']
a1    a
a4    c
Name: y, dtype: object
```

Cambiar valores en subconjuntos de un DataFrame

A veces se usa indexación en cadena en lugar de usar `loc`: Recordad el DataFrame datos:

```
     x1   x2   x3  y
a1  4.2  2.1  2.5  a
a2  0.7 -3.7 -2.3  b
a3 -2.4 -2.1  4.0  b
a4  0.3 -1.5  1.9  c
```

```
>>> datos[~(datos['y'] == 'b')]['y']
a1    a
a4    c
Name: y, dtype: object
```

devuelve el mismo objeto que

```
>>> datos.loc[~(datos['y'] == 'b'), 'y']
a1    a
a4    c
Name: y, dtype: object
```

Cambiar valores en subconjuntos de un DataFrame

A veces se usa indexación en cadena en lugar de usar `loc`: Recordad el DataFrame datos:

```
     x1   x2   x3  y
a1  4.2  2.1  2.5  a
a2  0.7 -3.7 -2.3  b
a3 -2.4 -2.1  4.0  b
a4  0.3 -1.5  1.9  c
```

```
>>> datos[~(datos['y'] == 'b')]['y']
a1    a
a4    c
Name: y, dtype: object
```

devuelve el mismo objeto que

```
>>> datos.loc[~(datos['y'] == 'b'), 'y']
a1    a
a4    c
Name: y, dtype: object
```

AVISO

Sin embargo, NO se debe usar la primera forma para asignarle un valor

```
datos[~(datos['y'] == 'b')]['y'] = 'b'
```

Es de mucho preferible usar `loc` para ello

```
datos.loc[~(datos['y'] == 'b'), 'y'] = 'b'
```