10_Seleccion_en_DataFrame_II

November 28, 2023

0.1 Selección en DataFrame (II)

0.1.1 DataFrame como un array bidimensional mejorado

Como se ha mencionado en otras sesiones, también podemos ver el `DataFrame` como un array bidimensional mejorado. Podemos examinar la matriz de datos subyacente (que tambien es un arry de numpay) utilizando el atributo `values`: En esta sesion tb expliaremos como hacer filas y columnas partiendo de condiciones . RECORDAR = las series pandas eran vistas como array unidimensional

[1]:
```
import pandas as pd

superficie = pd.Series({'California': 423967, 'Texas': 695662,
                  'New York': 141297, 'Florida': 170312,
                  'Illinois': 149995})
poblacion = pd.Series({'California': 38332521, 'Texas': 26448193,
                 'New York': 19651127, 'Florida': 19552860,
                 'Illinois': 12882135})

data = pd.DataFrame({'area':superficie, 'pob' :poblacion})
data["densidad"]  = data.pob/data.area # Si desaconsejamos este uso, pero a␣
 ↪veces es más rápido ;-) mejor data["pob"]/data["area"]
data
```

[1]:

	area	pob	densidad
California	423967	38332521	90.413926
Texas	695662	26448193	38.018740

```
New York    141297  19651127  139.076746
Florida     170312  19552860  114.806121
Illinois    149995  12882135   85.883763
```

ahora vamos a covertirlo en un array bidimensional con el atributo values

[2]:
```
data.values
```

[2]:
```
array([[4.23967000e+05, 3.83325210e+07, 9.04139261e+01],
       [6.95662000e+05, 2.64481930e+07, 3.80187404e+01],
       [1.41297000e+05, 1.96511270e+07, 1.39076746e+02],
       [1.70312000e+05, 1.95528600e+07, 1.14806121e+02],
       [1.49995000e+05, 1.28821350e+07, 8.58837628e+01]])
```

Se pueden hacer muchas operaciones del tipo array numpy en el propio `DataFrame`. **Por ejemplo, podemos transponer el `DataFrame` completo para intercambiar filas y columnas:**

[3]:
```
data.T#  la inversa (el nombre de las columnas pasa a los indices y al reves)
```

[3]:

	California	Texas	New York	Florida	Illinois
area	4.239670e+05	6.956620e+05	1.412970e+05	1.703120e+05	1.499950e+05
pob	3.833252e+07	2.644819e+07	1.965113e+07	1.955286e+07	1.288214e+07
densidad	9.041393e+01	3.801874e+01	1.390767e+02	1.148061e+02	8.588376e+01

Sin embargo, cuando se trata de indexar objetos `DataFrame`, está claro que la indexación de las columnas al estilo de un diccionario impide nuestra **capacidad de tratarlo simplemente como un array de NumPy.** En particular, al pasar un solo índice a un array se accede a una fila:

podemos acceder con elementos posicionales implicitos si vermos el dataframe con un array con el atributo values, no pudiendo hacerlo tal cuela sobre el dataframe directamente, porque lo que esta esperando DF es el nombre de una columna. Entonces para acceder eleemnto a elemento en un DF necesitaremos indexadores de las series panda (loc e iloc) pero con un funcionamiento mas particular

[5]:
```
data.values[2,2]# me dara el elemento 2 dela fila 2 ( es la densidad de␣
↪poblacion de nueva york del data frame)
```

[5]:
```
139.07674614464568
```

[ ]:
```

```

y pasando un único ”índice” a un `DataFrame` se accede a una columna:

[10]:
```
data[2:2]# no devuelve nada
data["area"]
```

[10]:
```
California    423967
Texas         695662
New York      141297
Florida       170312
```

```
Illinois      149995
Name: area, dtype: int64
```

Por lo tanto, para la indexación estilo array, necesitamos otra convención. Aquí Pandas vuelve a utilizar los indexadores `loc` e `iloc` mencionados al hablar de la selección e indexado en Series.

Usando el indexador `iloc`, podemos indexar el array subyacente como si fuera un simple array de NumPy (usando el índice implícito(slicing) de estilo Python, es decir usando los indices posicionales viendolo como una matriz con filas y columnas), pero el índice `DataFrame` y las etiquetas de las columnas se mantienen en el resultado:

[11]:
```
data
```

[11]:

	area	pob	densidad
California	423967	38332521	90.413926
Texas	695662	26448193	38.018740
New York	141297	19651127	139.076746
Florida	170312	19552860	114.806121
Illinois	149995	12882135	85.883763

[13]:
```
# desnidad de tejas seria el elemento [0,1][0,1,2]
data.iloc[1,2] # te da lel valor del extremo
```

[13]:
```
38.01874042279153
```

[14]:
```
data.iloc[:3,0:2]# 3 filas y 3 primeras columnas
```

[14]:

	area	pob
California	423967	38332521
Texas	695662	26448193
New York	141297	19651127

Formula == data.iloc[selctor posicional de filas, selector posicional de columna]

[16]:
```
data.iloc[-1,1]# ultima fila y ultima columna
```

[16]:
```
12882135
```

Del mismo modo, utilizando el indexador `loc` podemos indexar los datos subyacentes en un estilo similar al de los arrays pero utilizando el índice explícito y los nombres de las columnas: Recuerda que en un dataframe existen 2 indices explicitos : el index, el indice del dataframe y nombre de las columnas y con loc jugaremos con esto

FORMULA: data.loc[selector o filtro de filas(porque tb se puede enmascarar las filas), selector de columnas]

[18]:
```
data.loc["Texas":"Florida"]# me dara las filas entre texas y␣
↪florida(inlusives)(indice) y todas sus columnas
```

[18]:

	area	pob	densidad
Texas	695662	26448193	38.018740
New York	141297	19651127	139.076746
Florida	170312	19552860	114.806121

[21]:
```
data.loc ["Texas":"Florida", "pob":"densidad"] # cojo las filas entre texas y␣
↪florida,ambas inclusive, pero con las columnas pob y densidad (area no)
```

[21]:

	pob	densidad
Texas	26448193	38.018740
New York	19651127	139.076746
Florida	19552860	114.806121

Como quedarme con todas las filas y solo algunas columnas. EN el anteriro no hacia falat decirle las columnas pero aqui esa el truco

[23]:
```
data.loc["pob":"densidad"]# da error porque el DF busca primero en las filas y␣
↪despues en las columnas, por lo que el truco es(siguiente celda)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File c:
↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\indexes\
↪py:3790, in Index.get_loc(self, key)
   3789 try:
-> 3790     return self._engine.get_loc(casted_key)
   3791 except KeyError as err:

File index.pyx:152, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:181, in pandas._libs.index.IndexEngine.get_loc()

File pandas\_libs\hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.
↪PyObjectHashTable.get_item()

File pandas\_libs\hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.
↪PyObjectHashTable.get_item()

KeyError: 'pob'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
```

```
e:\Cursos\BC_Data_Science\Repositorio\ONLINE_DS_THEBRIDGE_V\SPRING 4\UNIT␣
 ↪1\10_Seleccion_en_DataFrame_II.ipynb Celda 27 line 1

----> <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%201/10_Seleccion_en_DataFrame_II.
 ↪ipynb#X43sZmlsZQ%3D%3D?line=0'>1</a> data.loc["pob":"densidad"]# da error␣
 ↪porque el DF busca primero en las filas y despues en las columnas, por lo que␣
 ↪el truco es(siguiente celda)


File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\indexing
 ↪py:1153, in _LocationIndexer.__getitem__(self, key)
   1150 axis = self.axis or 0
   1152 maybe_callable = com.apply_if_callable(key, self.obj)
-> 1153 return self._getitem_axis(maybe_callable, axis=axis)


File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\indexing
 ↪py:1373, in _LocIndexer._getitem_axis(self, key, axis)
   1371 if isinstance(key, slice):
   1372     self._validate_key(key, axis)
-> 1373     return self._get_slice_axis(key, axis=axis)
   1374 elif com.is_bool_indexer(key):
   1375     return self._getbool_axis(key, axis=axis)


File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\indexing
 ↪py:1405, in _LocIndexer._get_slice_axis(self, slice_obj, axis)
   1402     return obj.copy(deep=False)
   1404 labels = obj._get_axis(axis)
-> 1405 indexer = labels.slice_indexer(slice_obj.start, slice_obj.stop,␣
 ↪slice_obj.step)
   1407 if isinstance(indexer, slice):
   1408     return self.obj._slice(indexer, axis=axis)


File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\indexes
 ↪py:6601, in Index.slice_indexer(self, start, end, step)
   6557 def slice_indexer(
   6558     self,
   6559     start: Hashable | None = None,
   6560     end: Hashable | None = None,
   6561     step: int | None = None,
   6562 ) -> slice:
   6563     """
   6564     Compute the slice indexer for input labels and step.
   6565
   (…)
   6599     slice(1, 3, None)
   6600     """
-> 6601     start_slice, end_slice = self.slice_locs(start, end, step=step)
```

```
   6603     # return a slice
   6604     if not is_scalar(start_slice):

File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\indexes\
 ↪py:6818, in Index.slice_locs(self, start, end, step)
   6816 start_slice = None
   6817 if start is not None:
-> 6818     start_slice = self.get_slice_bound(start, "left")
   6819 if start_slice is None:
   6820     start_slice = 0

File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\indexes\
 ↪py:6743, in Index.get_slice_bound(self, label, side)
   6740         return self._searchsorted_monotonic(label, side)
   6741     except ValueError:
   6742         # raise the original KeyError
-> 6743         raise err
   6745 if isinstance(slc, np.ndarray):
   6746     # get_loc may return a boolean array, which
   6747     # is OK as long as they are representable by a slice.
   6748     assert is_bool_dtype(slc.dtype)

File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\indexes\
 ↪py:6737, in Index.get_slice_bound(self, label, side)
   6735 # we need to look up the label
   6736 try:
-> 6737     slc = self.get_loc(label)
   6738 except KeyError as err:
   6739     try:

File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\indexes\
 ↪py:3797, in Index.get_loc(self, key)
   3792     if isinstance(casted_key, slice) or (
   3793         isinstance(casted_key, abc.Iterable)
   3794         and any(isinstance(x, slice) for x in casted_key)
   3795     ):
   3796         raise InvalidIndexError(key)
-> 3797     raise KeyError(key) from err
   3798 except TypeError:
   3799     # If we have a listlike key, _check_indexing_error will raise
   3800     #  InvalidIndexError. Otherwise we fall through and re-raise
   3801     #  the TypeError.
   3802     self._check_indexing_error(key)
```

```
KeyError: 'pob'
```

[24]:
```
data.loc[:,"pob":"densidad"]# aqui medimos todas las filas ( : ), y despues las columnas que queramos
```

[24]:
```
                pob    densidad
California  38332521   90.413926
Texas       26448193   38.018740
New York    19651127  139.076746
Florida     19552860  114.806121
Illinois    12882135   85.883763
```

Otra forma de acceder solo columnas, no es con data loc, sino a traves de la lista de columnas

[25]:
```
data[["pob","area"]]# convetimos las olumnas en una lista dentro del DF
```

[25]:
```
                pob    area
California  38332521  423967
Texas       26448193  695662
New York    19651127  141297
Florida     19552860  170312
Illinois    12882135  149995
```

se puede combinaer con el loc tb

[ ]:
```

```

[26]:
```
data.loc["Florida":,["pob","area"]]# dara todas las filas desde Florida hasta el final y los dos elementos de la lista ["pob","area"]
```

[26]:
```
               pob    area
Florida   19552860  170312
Illinois  12882135  149995
```

[27]:
```
data.loc["California":,["pob","area"]]#dara todas las filas desde California hasta el final y los dos elementos de la lista ["pob","area"]
```

[27]:
```
                pob    area
California  38332521  423967
Texas       26448193  695662
New York    19651127  141297
Florida     19552860  170312
Illinois    12882135  149995
```

Cualquiera de los patrones de acceso a los datos de estilo NumPy puede ser utilizado dentro de estos indexadores. Por ejemplo, en el indexador `loc` podemos combinar el enmascaramiento y la indexación ”de fantasía” (o selectiva) como en lo siguiente:

POdemos usar loc para establecer condiciones o filtros en DF, POr ejemplo, quiero quedarme con las filas de los estados que superen en poblacion 20 millones

[28]:
```
data.loc[data["pob"] > 20000000]# filas filtradas y todas las columnas
```

[28]:
```
              area       pob   densidad
California  423967  38332521  90.413926
Texas       695662  26448193  38.018740
```

[30]:
```
data.loc[data["pob"] > 20000000, "densidad"]# filas filtrdas con condicion y
 ↪columna densidad
```

[30]:
```
California    90.413926
Texas         38.018740
Name: densidad, dtype: float64
```

[31]:
```
data.loc[data["pob"] > 20000000, ["area", "densidad"]]# filas filtradas y lista
 ↪con dos elementos que son las columnas de interes
```

[31]:
```
              area   densidad
California  423967  90.413926
Texas       695662  38.018740
```

[32]:
```
data.loc[data["pob"]< 20000000, "densidad"]
```

[32]:
```
New York    139.076746
Florida     114.806121
Illinois     85.883763
Name: densidad, dtype: float64
```

[ ]:
```

```

CURIOSIDAD: Si pedimos solo una columna, la informacion nos lo devuelve como una serie panda pero si le ponemos varias columnas nos da un dataframe

Tambien podemos cambiar los valores del dataframe:

[33]:
```
data.iloc[3,2] = 1000
data
```

[33]:
```
              area       pob     densidad
California  423967  38332521    90.413926
Texas       695662  26448193    38.018740
New York    141297  19651127   139.076746
Florida     170312  19552860  1000.000000
Illinois    149995  12882135    85.883763
```

[35]:
```
data.loc[data["pob"] > 20000000, "densidad"]= 12# para todas filas con esa
 ↪poblacion, le vamos a cambair su densidad a 12
```

```
data
```

[35]:

	area	pob	densidad
California	423967	38332521	12.000000
Texas	695662	26448193	12.000000
New York	141297	19651127	139.076746
Florida	170312	19552860	1000.000000
Illinois	149995	12882135	85.883763