09_Seleccion_en_DataFrame_I

November 28, 2023

0.1 Selección en DataFrame (I)

Recuerda que un `DataFrame` actúa en muchos aspectos como una matriz bidimensional o estructurada, y en otros aspectos como un diccionario de estructuras `Series` que comparten el mismo índice. Estas analogías pueden ser útiles para tener en cuenta mientras exploramos la selección de datos dentro de esta estructura. NOTA: las series pandas recordamos, podrian ser vistas como un array unidimensional o como un diccionario con claves y valores

0.1.1 DataFrame como un diccionario: un conjunto de series que estan como si fueran los valores de un diccionario, cuyo nombre el de las columnas y delas series (filas) es de las claves de nuestro megadiccionario dataframe

La primera analogía que consideraremos es el `DataFrame` como diccionario de objetos `Series` relacionados. Volvamos a nuestro ejemplo de áreas y poblaciones de los estados:

[1]:
```
import pandas as pd

superficie = pd.Series({'California': 423967, 'Texas': 695662,# el index seran␣
 ↪las claves del diccionario
                 '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})# nos  creamos otro␣
 ↪diccionario donde cada serie tentra los normbres de cad auna de las columnas␣
 ↪con nuestro dataframe
data
```

[1]:
```
              area       pob
California  423967  38332521
Texas       695662  26448193
New York    141297  19651127
Florida     170312  19552860
Illinois    149995  12882135
```

Se puede acceder a las `Series` individuales que componen las columnas del `DataFrame` a través de la indexación de estilo diccionario del nombre de la columna:

[9]:
```
data["area"]# nos ha devuelto la primera de las columnas
```

[9]:
```
California    423967
Texas         695662
New York      141297
Florida       170312
Illinois      149995
Name: area, dtype: int64
```

[5]:
```
area_var = data["area"]
print(area_var, type(area_var))# nos diece que es una sarie de panda
```

```
California    423967
Texas         695662
New York      141297
Florida       170312
Illinois      149995
Name: area, dtype: int64 <class 'pandas.core.series.Series'>
```

De forma equivalente, podemos utilizar un acceso de tipo atributo con nombres de columna que sean cadenas:

Otra forma de acceder a las columnas, esta vez NUEVA, que el dataframe una vez creado tiene las columnas como atributos, devolviendo el mismo objeto

[10]:
```
data.pob
```

[10]:
```
California    38332521
Texas         26448193
New York      19651127
Florida       19552860
Illinois      12882135
Name: pob, dtype: int64
```

[11]:
```
data["pob"]
```

[11]:
```
California    38332521
Texas         26448193
New York      19651127
Florida       19552860
Illinois      12882135
Name: pob, dtype: int64
```

[12]:
```
data.area
```

[12]:
```
California    423967
Texas         695662
New York      141297
Florida       170312
Illinois      149995
Name: area, dtype: int64
```

[13]:
```
data ["area"]
```

[13]:
```
California    423967
Texas         695662
New York      141297
Florida       170312
Illinois      149995
Name: area, dtype: int64
```

Este acceso a la columna de estilo atributo accede en realidad al mismo objeto que el acceso de estilo diccionario:

[14]:
```
data.pob is data["pob"] # ==, is comprueba si apuntanen memoria al mismo lugar
```

[14]:
```
True
```

Al igual que con los objetos `Series` discutidos anteriormente, esta sintaxis de estilo diccionario también se puede utilizar para modificar el objeto, en este caso añadiendo una nueva columna:

[18]:
```
var = data.pob
var2 = data["pob"]
var["Kansas"] =123000# vemos que apuntan al mismo objeto por lo que cambian el␣
 ↪valor auqnue sean distintas variables
```

[19]:
```
var2
```

[19]:
```
California    38332521
Texas         26448193
New York      19651127
Florida       19552860
Illinois      12882135
```

```
Kansas          123000
Name: pob, dtype: int64
```

Las operaciones con columnas y series las veremos en sesiones posteriores con más detalle. ¿como crear una columna nueva? desidad = poblacion / superfice

[21]:
```
data["densidad"] =data["pob"]/data["area"]
```

simplemente creando una nueva columna al dataframe, y a la vez haciendo la division entre las dos columnas existentes para hallar la densidad de pobacion, nos devolvera el dataframe orginal(data) con esta tercera columna nueva

[23]:
```
data
```

[23]:

	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

[Para terminar y revisando los dos últimos puntos]. En general, es mejor evitar la tentación de intentar la asignación de columnas a través de atributos (es decir, usa `datos['pob'] = z` en lugar de `datos.pob = z`). Lo vemos con un ejemplo y de camino veremos como generar columnas a partir de listas panda

[33]:
```
nombres_cortos = ["CA","TX","NY","FL","IL"]
data["nombres cortos"] = nombres_cortos
data
```

[33]:

	area	pob	densidad	nombre cortos	nombres cortos
California	423967	38332521	90.413926	CA	CA
Texas	695662	26448193	38.018740	TX	TX
New York	141297	19651127	139.076746	NY	NY
Florida	170312	19552860	114.806121	FL	FL
Illinois	149995	12882135	85.883763	IL	IL

recordamos: para dataframe acceder tipo diccionario, y en las series panda con loc si es explicito u iloc si es implicito(slicing)

[36]:
```
data["nombres cortos"]
```

[36]:
```
California    CA
Texas         TX
New York      NY
Florida       FL
Illinois      IL
Name: nombres cortos, dtype: object
```

[37]:
```
data.nombres cortos
```

```
  Cell In[37], line 1
    data.nombres cortos
                 ^
SyntaxError: invalid syntax
```

Por esete motivo, que al usar de atributo la fila separada en dos palabras da error, asi que es reconmendable al asignar variables acceder como diccionario en estos casos

Esto muestra una vista previa de la sintaxis directa de la aritmética elemento por elemento entre los objetos `Series` que veremos más adelante.

[ ]: