05_DataFrame_II

November 28, 2023

0.1 Pandas: DataFrame (II)

En esta sesión hablaremos de DataFrames como diccionarios, o algo así, y luego al igual que con las series practicaremos la construcción de DataFrames.

Pero lo primero es importar pandas y recuperar las Series y el DataFrame que creamos en la sesión anterior.

[5]:
```
import pandas as pd

population_dict = {'California': 38332521,
                   'Texas': 26448193,
                   'New York': 19651127,
                   'Florida': 19552860,
                   'Illinois': 12882135}

population = pd.Series(population_dict)


area_dict = {'California': 423967, 'Texas': 695662, 'New York': 141297,
             'Florida': 170312, 'Illinois': 149995}


area = pd.Series(area_dict)

estados = {"poblacion": population,
           "superficie": area}
```

```
states = pd.DataFrame(estados)
```

[2]:
```
states
```

[2]:
```
            poblacion  superficie
California   38332521      423967
Texas        26448193      695662
New York     19651127      141297
Florida      19552860      170312
Illinois     12882135      149995
```

0.1.1 DataFrame como diccionario especializado

Del mismo modo,como tb haciemos con las series, también podemos pensar en un `DataFrame` como una especialización de un diccionario. Mientras que un diccionario asigna una clave a un valor, un `DataFrame` asigna un nombre de columna a una `Serie` de datos de columna. Por ejemplo, pedir el atributo `superficie` devuelve el objeto `Series` que contiene las áreas que vimos anteriormente:

[6]:
```
states["superficie"]
```

[6]:
```
California    423967
Texas         695662
New York      141297
Florida       170312
Illinois      149995
Name: superficie, dtype: int64
```

[7]:
```
states["poblacion"]
```

[7]:
```
California    38332521
Texas         26448193
New York      19651127
Florida       19552860
Illinois      12882135
Name: poblacion, dtype: int64
```

[8]:
```
states["Florida"]# al intentar buscar por filas nos da un error ya que Florida␣
 ↪no esta en columnas que es lo busca
```

```
---------------------------------------------------------------------------
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: 'Florida'

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\05_DataFrame_II.ipynb Celda 9 line 1
----> <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%201/05_DataFrame_II.
 ↪ipynb#X11sZmlsZQ%3D%3D?line=0'>1</a> states["Florida"]

File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\frame.
 ↪py:3893, in DataFrame.__getitem__(self, key)
   3891 if self.columns.nlevels > 1:
   3892     return self._getitem_multilevel(key)
-> 3893 indexer = self.columns.get_loc(key)
   3894 if is_integer(indexer):
   3895     indexer = [indexer]

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: 'Florida'
```

[9]:
```
states.columns# el atrubuto para saber las columnas de  un diccionario
```

[9]:
```
Index(['poblacion', 'superficie'], dtype='object')
```

[10]:
```
states.loc["Florida"]# pero si existe una forma de acceder por el indice a una␣
↪fila que es esta
```

[10]:
```
poblacion     19552860
superficie      170312
Name: Florida, dtype: int64
```

Exploraremos medios más flexibles para indexar `DataFrame` sesiones posteriores

0.1.2 Construyendo DataFrames

Un `DataFrame` de Pandas se puede construir de varias maneras. Veremos varios ejemplos.

A partir de un único objeto Serie Un `DataFrame` es una colección de objetos `Series`, y se puede construir un `DataFrame` de una sola columna a partir de una única `Series`: No le damos variable(solo lo muestra) en data ponemos la serie population, y en columns, el nombre que le queremos dar a cada serie

[11]:
```
pd.DataFrame (data = population, columns =["poblacion"])
```

[11]:

	poblacion
California	38332521
Texas	26448193
New York	19651127
Florida	19552860
Illinois	12882135

[ ]:
```

```

A partir de una lista de diccionarios Cualquier lista de dictos puede convertirse en un `DataFrame`. Utilizaremos una simple comprensión de la lista para crear algunos datos:

[14]:
```
data = [{"a":i, "b": 2*i} for i in range(3)] # nos va a crear el diccionario 3␣
↪veces y en cada iteracion nos va a ir cambiando el valor de i(0,1,2)
data
```

[14]:
```
[{'a': 0, 'b': 0}, {'a': 1, 'b': 2}, {'a': 2, 'b': 4}]
```

Incluso si faltan algunas claves en el diccionario, Pandas las rellenará con valores `NaN` (es decir, "no un número"): [De NaN y nulos como también se les conoce hablaremos en sesiones posteriores NaN es el acronimo de Not a Number]

[15]:
```
pd.DataFrame(data)# nos crea un dataframe con las 4 filas, cogiendo como nombre␣
↪de las columnas a y b( tiene que tener las mismas claves, y sino los␣
↪rellenara con NaN)
```

[15]:
```
   a  b
0  0  0
1  1  2
2  2  4
```

[17]:
```
pd.DataFrame( [{"a":1, "b":2},{"pepito":2, "juanita":4}]) # el dataframe ha␣
↪cogido los nombre de las 4 columnas y ha rellenado con Nan los valores␣
↪faltantes
```

[17]:
```
     a    b  pepito  juanita
0  1.0  2.0     NaN      NaN
1  NaN  NaN     2.0      4.0
```

A partir de un diccionario de objetos Serie Como vimos antes, un `DataFrame` puede construirse también a partir de un diccionario de objetos `Series` [que es como hemos hecho en la sesion anterior y al principio de esta]:

[18]:
```
pd.DataFrame({"population_bis": population, "area": area})# esto nos dara un␣
↪dataframe cuyas columans seran population bis y ares
```

[18]:
```
            population_bis    area
California        38332521  423967
Texas             26448193  695662
New York          19651127  141297
Florida           19552860  170312
Illinois          12882135  149995
```

[19]:
```
# otra forma de hacerlos era cogiendo en vez de columnas: un diccionario de␣
↪listas,lo que tiens que tener en cuenta es que yiene que haber el mismo␣
↪numero de leementos en las distauintas listas del diccionario
pd.DataFrame({"columna_1":[12,3,4], "columna_2": [4,5,6]})
```

[19]:
```
   columna_1  columna_2
0         12          4
1          3          5
2          4          6
```

A partir de un array bidimensional de NumPy Dado un array bidimensional de datos, podemos crear un `DataFrame` con los nombres de columna e índice que se especifiquen. Si se omite, se utilizará un índice entero para cada una:

[21]:
```
import numpy as np

np.random.seed(10)# estableciendo la semilla, nos permite que siempre nos de el␣
↪mismo array
```

```
array_base = np.random.rand(3,2)

array_base
```

[21]:
```
array([[0.77132064, 0.02075195],
       [0.63364823, 0.74880388],
       [0.49850701, 0.22479665]])
```

[26]:
```
# vamos contruir el DataFrame
pd.DataFrame(array_base) # si no le ponemos nombre a las columnas ni a las␣
↪filas(index) nos dara 0 e 1
```

[26]:

	0	1
0	0.771321	0.020752
1	0.633648	0.748804
2	0.498507	0.224797

[24]:
```
# vamos contruir el DataFrame
pd.DataFrame(array_base, columns= ["col_2", "col_2"], index= ["fil_1", "fil_2",␣
↪"fil_3"])
```

[24]:

	col_2	col_2
fil_1	0.771321	0.020752
fil_2	0.633648	0.748804
fil_3	0.498507	0.224797

[ ]:
```

```