Clases - Semana 2

September 27, 2020

1 The Series Data Structure

[4]:
```
import pandas as pd
```

[5]:
```
students = ['Alice', 'Jack', 'Molly']   #creo una serie pasandole una lista
pd.Series(students)
```

[5]:
```
0    Alice
1     Jack
2    Molly
dtype: object
```

[6]:
```
numbers = [1, 2, 3]
pd.Series(numbers)
```

[6]:
```
0    1
1    2
2    3
dtype: int64
```

[7]:
```
students = ['Alice', 'Jack', None]  #puedo asignar valores vacios
pd.Series(students)
```

[7]:
```
0    Alice
1     Jack
2     None
dtype: object
```

[8]:
```
numbers = [1, 2, None]
pd.Series(numbers)  #pandas convierte el tipo a float, porque falta un valor
```

[8]:
```
0    1.0
1    2.0
2    NaN
dtype: float64
```

[9]:
```
#Nan no es lo mismo que None (son muy similares), se tratan distinto por␣
 ↪razones de eficiencia.
```

[10]:
```
#NaN: not a number
import numpy as np
np.nan == None #no se puede igualar un Nan por si mismo, siempre sera False
```

[10]:
```
False
```

[11]:
```
np.nan == np.nan #esto tambien da False
```

[11]:
```
False
```

[12]:
```
#hay que usar funciones para detectar un NaN:
np.isnan(np.nan)
```

[12]:
```
True
```

[13]:
```
students_score = {'Alice': 'Physics',  #creo una serie con un diccionario.
                  'Jack': 'Chemistry',
                  'Molly': 'English'}
s = pd.Series(students_score)
s
```

[13]:
```
Alice      Physics
Jack     Chemistry
Molly      English
dtype: object
```

[14]:
```
s.index #los index tambien pueden ser string, con esta funcion los veo
```

[14]:
```
Index(['Alice', 'Jack', 'Molly'], dtype='object')
```

[15]:
```
students = [('Alice', 'Brown'), ('Jack', 'White'), ('Molly', 'Green')] #hago␣
 ↪una serie con una lista de tuplas
pd.Series(students)
```

[15]:
```
0    (Alice, Brown)
1     (Jack, White)
2    (Molly, Green)
dtype: object
```

[16]:
```
#se puede pasar el index cuando se crea la serie:
s = pd.Series(['Physics', 'Chemistry', 'English'], index = ['Alice', 'Jack',␣
 ↪'Molly'])
s
```

[16]:
```
Alice      Physics
Jack     Chemistry
Molly      English
dtype: object
```

[17]:
```
#si creo una serie con un diccionario y el index que le pongo no coincide con␣
 ↪las keys del dict, los valores del index
#que no sean keys del dict se llenaran con NaN. Por otro lado las keys que no␣
 ↪esten en el index no apareceran en la serie
students_score = {'Alice': 'Physics',
                  'Jack': 'Chemistry',
                  'Molly': 'English'}
s = pd.Series(students_score, index = ['Alice', 'Jack', 'Sam'])
```

```
s #aparece Sam y no aparece Molly
```

[17]:
```
Alice      Physics
Jack     Chemistry
Sam            NaN
dtype: object
```

2 Queryng a Series

[18]:
```
import pandas as pd
students_classes = {'Alice': 'Physics',
                    'Jack': 'Chemistry',
                    'Molly': 'English',
                    'Sam': 'History'}
s = pd.Series(students_classes)
s
```

[18]:
```
Alice      Physics
Jack     Chemistry
Molly      English
Sam        History
dtype: object
```

[19]:
```
#el iloc se utiliza para ver una posicion numerica (index position)
s.iloc[3]
```

[19]:
```
'History'
```

[20]:
```
#el loc se utiliza para ver una posicion por la etiqueta del index (index␣
 ↪label)
s.loc['Molly']
```

[20]:
```
'English'
```

[21]:
```
#iloc y loc son atributos (y no metodos), por eso se usan corchetes.
#lo mismo se puede hacer de la siguiente manera:
s[3]
```

[21]:
```
'History'
```

[22]:
```
s['Molly']
```

[22]:
```
'English'
```

Si tengo numeros enteros como index, como sabe pandas si me refiero a index position o index label? Por eso siempre es recomendable usar loc y iloc.

[23]:
```
class_code = {99: 'Physics',
              100: 'Chemistry',
              101: 'English',
              102: 'History'}
s = pd.Series(class_code)
```

[24]:
```
#en este caso poner: s[0] tiraria error porque no hay ningun item con index 0.␣
↪Para eso tenemos que usar iloc:
s.iloc[0]
```

[24]:
```
'Physics'
```

[25]:
```
#ejemplo de una serie con notas de alumnos y el calculo del promedio:
grades = pd.Series([90, 80, 70, 60])
total = 0

for grade in grades:
    total += grade
print(total/len(grades))
```

```
75.0
```

[26]:
```
#esto se puede hacer de manera mas sencilla usando numpy
import numpy as np

total = np.sum(grades)
print(total/len(grades))
```

```
75.0
```

[27]:
```
#Ahora veamos cual de los dos metodos es mas rapido, usando una serie de mayor␣
↪tamaño:

numbers = pd.Series(np.random.randint(0, 1000, 10000)) #estoy pidiendo 10.000␣
↪numeros enteros random entre 0 y 1000
numbers.head() #veo los primeros 5 valores
```

[27]:
```
0    663
1    845
2    517
3    698
4    939
dtype: int64
```

[28]:
```
%%timeit -n 100
total = 0
for number in numbers:
    total += number
total/len(numbers)
#mido el tiempo con un magic para la primera opcion. El n es la cantidad de␣
↪veces que mide para luego devolver un promedio.
```

```
1.13 ms ś 7.55 ţs per loop (mean ś std. dev. of 7 runs, 100 loops each)
```

[29]:
```
%%timeit -n 100
total = np.sum(numbers)
total/len(numbers)
#mido el tiempo para la segunda opcion, que es mucho mas rapida. La funcion␣
 ↪magic siempre debe ser la primer linea de la celda.
```

```
66.7 µs ś 8.3 µs per loop (mean ś std. dev. of 7 runs, 100 loops each)
```

[30]:
```
numbers.head()
```

[30]:
```
0    663
1    845
2    517
3    698
4    939
dtype: int64
```

[31]:
```
numbers += 2 #le sumo 2 a todos los valores de la serie, no es necesario iterar
numbers.head()
```

[31]:
```
0    665
1    847
2    519
3    700
4    941
dtype: int64
```

[32]:
```
#tambien se puede hacer iterando, de la siguiente manera:
for label, value in numbers.iteritems():
    numbers.set_value(label, value+2)
numbers.head()
```

[32]:
```
0    667
1    849
2    521
3    702
4    943
dtype: int64
```

[33]:
```
#vamos a medir los tiempos a ver cual de las dos opciones es mas rapida:
```

[34]:
```
%%timeit -n 10
s = pd.Series(np.random.randint(0, 1000, 1000))
s += 2
```

```
240 µs ś 19.2 µs per loop (mean ś std. dev. of 7 runs, 10 loops each)
```

[35]:
```
%%timeit -n 10
s = pd.Series(np.random.randint(0, 1000, 1000))
for label, value in s.iteritems():
```

```
    s.loc[label] = value + 2
```

```
124 ms ś 2.85 ms per loop (mean ś std. dev. of 7 runs, 10 loops each)
```

[36]:
```
#puedo tener distintos tipos de index y agregarlos con .loc:
s = pd.Series([1, 2, 3])
s.loc['History'] = 102
s
```

[36]:
```
0            1
1            2
2            3
History    102
dtype: int64
```

[37]:
```
#que pasa si los index no son unicos? aca va un ejemplo:
students_classes = pd.Series({'Alice': 'Physics',
                  'Jack': 'Chemistry',
                  'Molly': 'English',
                  'Sam': 'History'})
students_classes
```

[37]:
```
Alice      Physics
Jack     Chemistry
Molly      English
Sam        History
dtype: object
```

[38]:
```
kelly_classes = pd.Series(['Philosophy', 'Arts', 'Math'], index = ['Kelly',␣
 ↪'Kelly', 'Kelly'])
kelly_classes
```

[38]:
```
Kelly    Philosophy
Kelly          Arts
Kelly          Math
dtype: object
```

[39]:
```
#ahora agrego las clases de kelly con el resto de los estudiantes
all_students_classes = students_classes.append(kelly_classes)
all_students_classes
```

[39]:
```
Alice       Physics
Jack      Chemistry
Molly       English
Sam         History
Kelly    Philosophy
Kelly          Arts
Kelly          Math
dtype: object
```

[40]:
```
all_students_classes.loc['Kelly']
```

[40]:
```
Kelly    Philosophy
Kelly          Arts
Kelly          Math
dtype: object
```

[41]:
```
students_classes #esta sigue siendo igual, ya que con el append no se modifica
 ↪el valor original, sino que se devuelve otro con
                 #el agregado
```

[41]:
```
Alice      Physics
Jack     Chemistry
Molly      English
Sam        History
dtype: object
```

3 DataFrame Data Structure

[42]:
```
import pandas as pd
```

[43]:
```
#creo tres series
record1 = pd.Series({'Name': 'Alice',
                     'Class': 'Physics',
                     'Score': 85})
record2 = pd.Series({'Name': 'Jack',
                     'Class': 'Chemistry',
                     'Score': 82})
record3 = pd.Series({'Name': 'Helen',
                     'Class': 'Biology',
                     'Score': 90})
```

[44]:
```
#creo data frame con las series
df = pd.DataFrame([record1, record2, record3], index = ['School 1', 'School 2',
 ↪'School 1'])
df.head()
```

[44]:

	Name	Class	Score
School 1	Alice	Physics	85
School 2	Jack	Chemistry	82
School 1	Helen	Biology	90

[45]:
```
#puedo hacer lo mismo con una lista de diccionarios:
students = [{'Name': 'Alice',
             'Class': 'Physics',
             'Score': 85},
            {'Name': 'Jack',
             'Class': 'Chemistry',
             'Score': 82},
            {'Name': 'Helen',
             'Class': 'Biology',
```

```
            'Score': 90}]
df = pd.DataFrame(students, index = ['School 1', 'School 2', 'School 1'])
df
```

[45]:
```
           Name      Class  Score
School 1  Alice    Physics     85
School 2   Jack  Chemistry     82
School 1  Helen    Biology     90
```

[46]:
```
#para seleccionar datos puedo usar loc:
df.loc['School 2']
```

[46]:
```
Name          Jack
Class    Chemistry
Score           82
Name: School 2, dtype: object
```

[47]:
```
type(df.loc['School 2']) #veo el tipo de objeto
```

[47]:
```
pandas.core.series.Series
```

[48]:
```
df.loc['School 1'] #obviamente los index pueden no ser unicos
```

[48]:
```
           Name    Class  Score
School 1  Alice  Physics     85
School 1  Helen  Biology     90
```

[49]:
```
#si quiero los nombres de los estudiantes de la escuela 1:
df.loc['School 1', 'Name']
```

[49]:
```
School 1    Alice
School 1    Helen
Name: Name, dtype: object
```

[50]:
```
#si quiero saber los nombres de todos:
df['Name']
#df.loc['Name'] tira error, no se puede aplicar loc en los nombres de columnas
```

[50]:
```
School 1    Alice
School 2     Jack
School 1    Helen
Name: Name, dtype: object
```

[51]:
```
df.loc['School 1']['Name'] #tambien se puede hacer de esta manera, aplico .loc␣
 ↪y despues indico las columnas que deseo
```

[51]:
```
School 1    Alice
School 1    Helen
Name: Name, dtype: object
```

[52]:
```
#si quiero todos los nombres y puntajes puedo hacer slicing:
df.loc[:, ['Name', 'Score']]
```

[52]:
```
           Name  Score
School 1  Alice     85
School 2   Jack     82
School 1  Helen     90
```

[53]:
```
#para borrar puedo usar .drop. Esta funcion no cambia el df original, sino que devuelve una copia
df.drop('School 1')
```

[53]:
```
          Name      Class  Score
School 2  Jack  Chemistry     82
```

[54]:
```
df #puedo ver que este df permanece igual, no se borro nada
```

[54]:
```
           Name      Class  Score
School 1  Alice    Physics     85
School 2   Jack  Chemistry     82
School 1  Helen    Biology     90
```

[55]:
```
#hay dos parametros importantes: inplace e axis
#si pongo el parametro inplace en True los datos se actualizaran en el df original (en vez de hacer una copia)
#el axis viene seteado en 0, si lo pongo en 1 voy a borrar columnas en vez de filas
```

[56]:
```
copy_df = df.copy() #creo una copia del df
copy_df.drop('Name', inplace = True, axis = 1) #borro los nombres inplace
copy_df
```

[56]:
```
             Class  Score
School 1    Physics     85
School 2  Chemistry     82
School 1    Biology     90
```

[57]:
```
#usando del puedo borrar datos tambien. Esto no devuelve una vista, hace efecto sobre el df original
del copy_df['Class'] #borro la columna de clases, si corro nuevamente esta celda tira error (porque ya no hay que borrar)
copy_df
```

[57]:
```
          Score
School 1     85
School 2     82
School 1     90
```

[58]:
```
df['Class Ranking'] = None #de esta manera puedo añadir columnas, todas con valor vacio
df
```

[58]:
```
           Name      Class  Score Class Ranking
School 1  Alice    Physics     85          None
School 2   Jack  Chemistry     82          None
```

```
School 1  Helen    Biology     90          None
```

4 Data Frame Indexing and Loading

[59]:
```
!cat Admission_Predict.csv #con !cat muestro los calores de un archivo
```

[60]:
```
import pandas as pd
```

[61]:
```
df = pd.read_csv('Admission_Predict.csv')
df.head()
```

[61]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65

[62]:
```
#veo que los index empiezan con 0 y el numero de serie con 1. Por eso se␣
 ↪cargara de nuevo y se pondra como index el numero de
#serie:
df = pd.read_csv('Admission_Predict.csv', index_col = 0)
df.head()
```

[62]:
```
             GRE Score  TOEFL Score  University Rating  SOP  LOR   CGPA  \
Serial No.
1                  337          118                  4  4.5   4.5  9.65
2                  324          107                  4  4.0   4.5  8.87
3                  316          104                  3  3.0   3.5  8.00
4                  322          110                  3  3.5   2.5  8.67
5                  314          103                  2  2.0   3.0  8.21

             Research  Chance of Admit
Serial No.
1                   1             0.92
2                   1             0.76
3                   1             0.72
4                   1             0.80
5                   0             0.65
```

[63]:
```
#ahora le vamos a cambiar el nombre a las columnas SOP y LOR, por su nombre␣
 ↪completo (son siglas). Para eso es necesario crear
#un diccionario para pasarle los nuevos nombres de columnas:
new_df = df.rename(columns = { 'GRE Score': 'GRE Score',
                               'TOEFL Score': 'TOEFL Score',
                               'University Rating': 'University Rating',
                               'SOP': 'Statement of Purpose',
                               'LOR': 'Letter of Recomendation',
                               'CGPA': 'CGPA',
                               'Research': 'Research',
                               'Chance of Admit': 'Chance of Admit'})
new_df.head()
```

[63]:
```
             GRE Score  TOEFL Score  University Rating  Statement of Purpose  \
Serial No.
1                  337          118                  4                   4.5
2                  324          107                  4                   4.0
3                  316          104                  3                   3.0
4                  322          110                  3                   3.5
5                  314          103                  2                   2.0

             LOR   CGPA  Research  Chance of Admit
Serial No.
1            4.5  9.65         1             0.92
2            4.5  8.87         1             0.76
3            3.5  8.00         1             0.72
4            2.5  8.67         1             0.80
5            3.0  8.21         0             0.65
```

[64]:
```
#ahora vemos que SOP cambio pero LOR no. Vamos a ver los nombres de las␣
 ↪columnas:
new_df.columns
```

[64]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'Statement of Purpose',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

[65]:
```
#vemos que despues de LOR hay un espacio, por eso el rename no funcionaba para␣
 ↪LOR. Renombro devuelta con el espacio:
new_df = new_df.rename(columns = {'LOR ': 'Letter of Recomendation'})
new_df.head()
```

[65]:

Serial No.	GRE Score	TOEFL Score	University Rating	Statement of Purpose	Letter of Recomendation	CGPA	Research	Chance of Admit
1	337	118	4	4.5	4.5	9.65	1	0.92
2	324	107	4	4.0	4.5	8.87	1	0.76
3	316	104	3	3.0	3.5	8.00	1	0.72
4	322	110	3	3.5	2.5	8.67	1	0.80
5	314	103	2	2.0	3.0	8.21	0	0.65

[66]:
```
#esto es medio fragil porque podrian haber sido dos espacios, un un tab. Para␣
 ↪esp usamos la funcion strip, que saca los
#espacios en blanco (del final del texto). Le indico que lo haga a traves de␣
 ↪las columnas
new_df = new_df.rename(mapper = str.strip, axis = 'columns')
new_df.head()
```

[66]:

Serial No.	GRE Score	TOEFL Score	University Rating	Statement of Purpose	Letter of Recomendation	CGPA	Research	Chance of Admit
1	337	118	4	4.5	4.5	9.65	1	0.92
2	324	107	4	4.0	4.5	8.87	1	0.76
3	316	104	3	3.0	3.5	8.00	1	0.72
4	322	110	3	3.5	2.5	8.67	1	0.80
5	314	103	2	2.0	3.0	8.21	0	0.65

[67]:
```
#el df sigue siendo el mismo, solo hay una copia en new_df con los nombres de␣
 ↪las columnas cambiados:
df.columns
```

[67]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research', 'Chance of Admit '],
      dtype='object')
```

[68]:
```
#tambien puedo asignar una lista con los nombres de las columnas con df.columns.
 ↪ En este caso estoy modificando el df original
cols = list(df.columns) #creo una lista a partir de los nombres de columnas␣
 ↪originales
cols = [x.lower().strip() for x in cols] #paso todo a minuscula y elimino␣
 ↪espacios en blanco al final
df.columns = cols #realizo los cambios de nombres con la lista cols
df.head()
```

[68]:

Serial No.	gre score	toefl score	university rating	sop	lor	cgpa	research	chance of admit
1	337	118	4	4.5	4.5	9.65	1	0.92
2	324	107	4	4.0	4.5	8.87	1	0.76
3	316	104	3	3.0	3.5	8.00	1	0.72
4	322	110	3	3.5	2.5	8.67	1	0.80
5	314	103	2	2.0	3.0	8.21	0	0.65

5 Querying a DataFrame

Una boolean mask es una serie o data frame (con la misma cantidad de filas y columnas que el archivo de interes) con valores de True o False. Cuando paso la mascara por el archivo, los valores que coincidan con un True se mantendran mientras que los que coincidan con False se perderan.

[69]:
```
import pandas as pd
df = pd.read_csv('Admission_Predict.csv', index_col = 0)
df.columns = [x.lower().strip() for x in cols]
df.head()
```

[69]:

Serial No.	gre score	toefl score	university rating	sop	lor	cgpa	research	chance of admit
1	337	118	4	4.5	4.5	9.65	1	0.92
2	324	107	4	4.0	4.5	8.87	1	0.76
3	316	104	3	3.0	3.5	8.00	1	0.72

```
4                322          110                  3  3.5  2.5  8.67
5                314          103                  2  2.0  3.0  8.21

            research  chance of admit
Serial No.
1                  1              0.92
2                  1              0.76
3                  1              0.72
4                  1              0.80
5                  0              0.65
```

[70]:
```
#creo una mascara paralos estudiantes con chance of admit mayor a 0.7:
admit_mask = df['chance of admit'] > 0.7
admit_mask
```

[70]:
```
Serial No.
1       True
2       True
3       True
4       True
5      False
       ...
396     True
397     True
398     True
399    False
400     True
Name: chance of admit, Length: 400, dtype: bool
```

[71]:
```
#ahora podemos pasar  la mascara con la funcion .where():
df.where(admit_mask).head()
#los valores con chance of admit menor a 0.7 no se borran, sino que aparecen␣
 ↪como Nan
```

[71]:
```
            gre score  toefl score  university rating  sop  lor  cgpa  \
Serial No.
1               337.0        118.0                4.0  4.5  4.5  9.65
2               324.0        107.0                4.0  4.0  4.5  8.87
3               316.0        104.0                3.0  3.0  3.5  8.00
4               322.0        110.0                3.0  3.5  2.5  8.67
5                 NaN          NaN                NaN  NaN  NaN   NaN

            research  chance of admit
Serial No.
1                1.0              0.92
2                1.0              0.76
3                1.0              0.72
4                1.0              0.80
5                NaN               NaN
```

[72]:
```
#con el dropna() elimino todos los valores nulos
df.where(admit_mask).dropna().head()
```

[72]:
```
             gre score  toefl score  university rating  sop  lor  cgpa  \
Serial No.
1                337.0        118.0                4.0  4.5  4.5  9.65
2                324.0        107.0                4.0  4.0  4.5  8.87
3                316.0        104.0                3.0  3.0  3.5  8.00
4                322.0        110.0                3.0  3.5  2.5  8.67
6                330.0        115.0                5.0  4.5  3.0  9.34

             research  chance of admit
Serial No.
1                 1.0             0.92
2                 1.0             0.76
3                 1.0             0.72
4                 1.0             0.80
6                 1.0             0.90
```

[73]:
```
#con pandas se puede hacer where y dropna al mismo tiempo:
df[admit_mask].head()
```

[73]:
```
             gre score  toefl score  university rating  sop  lor  cgpa  \
Serial No.
1                  337          118                  4  4.5  4.5  9.65
2                  324          107                  4  4.0  4.5  8.87
3                  316          104                  3  3.0  3.5  8.00
4                  322          110                  3  3.5  2.5  8.67
6                  330          115                  5  4.5  3.0  9.34

             research  chance of admit
Serial No.
1                   1             0.92
2                   1             0.76
3                   1             0.72
4                   1             0.80
6                   1             0.90
```

[74]:
```
#esto tambien se puede usar para vero una o mas columnas:
df['gre score'].head()
```

[74]:
```
Serial No.
1    337
2    324
3    316
4    322
5    314
Name: gre score, dtype: int64
```

[75]:
```
df[['gre score', 'toefl score']].head()
```

[75]:
```
             gre score  toefl score
Serial No.
1                  337          118
2                  324          107
3                  316          104
4                  322          110
5                  314          103
```

[77]:
```
#le pongo directamente una mascara
df[df['gre score'] > 320].head()
```

[77]:
```
             gre score  toefl score  university rating  sop  lor  cgpa  \
Serial No.
1                  337          118                  4  4.5  4.5  9.65
2                  324          107                  4  4.0  4.5  8.87
4                  322          110                  3  3.5  2.5  8.67
6                  330          115                  5  4.5  3.0  9.34
7                  321          109                  3  3.0  4.0  8.20

             research  chance of admit
Serial No.
1                   1             0.92
2                   1             0.76
4                   1             0.80
6                   1             0.90
7                   1             0.75
```

[78]:
```
#para combinar muchas mascaras puedo usar and (&) y or (|).
(df['chance of admit'] > 0.7) & (df['chance of admit'] < 0.9)
```

[78]:
```
Serial No.
1      False
2       True
3       True
4       True
5      False
       ...
396     True
397     True
398    False
399    False
400    False
Name: chance of admit, Length: 400, dtype: bool
```

[79]:
```
#otra forma de hacerlo
df['chance of admit'].gt(0.7) & df['chance of admit'].lt(0.9) #gt: greater than.
 ↪ lt: lower than
```

[79]:
```
Serial No.
1       True
```

```
2      False
3      False
4      False
5      False
       ...
396    False
397    False
398     True
399    False
400     True
Name: chance of admit, Length: 400, dtype: bool
```

[81]:
```
df['chance of admit'].gt(0.7).lt(0.9) #tambien puedo encadenar gt y lt
```

[81]:
```
Serial No.
1      False
2      False
3      False
4      False
5       True
       ...
396    False
397    False
398    False
399     True
400    False
Name: chance of admit, Length: 400, dtype: bool
```

6 Indexing DataFrames

La funcion set_index() es destructiva y no mantiene el indice anterior, por lo que si se lo quiere conservar se debera copiar a una nueva columna antes de hacer el set_index.

[85]:
```
import pandas as pd
df = pd.read_csv('Admission_Predict.csv', index_col = 0)
df.head()
```

[85]:

Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
1	337	118	4	4.5	4.5	9.65	1	0.92
2	324	107	4	4.0	4.5	8.87		
3	316	104	3	3.0	3.5	8.00		
4	322	110	3	3.5	2.5	8.67		
5	314	103	2	2.0	3.0	8.21		

```
2                  1              0.76
3                  1              0.72
4                  1              0.80
5                  0              0.65
```

[86]:
```
#vamos a usar dei ndice el chance of admit, pero queremos conservar el numero␣
 ↪de serie por lo que debemos hacer una copia.
df['Serial number'] = df.index #copio el index a una nueva columna
df = df.set_index('Chance of Admit ') #seteo chance of admit como indice
df.head()
```

[86]:
```
                   GRE Score  TOEFL Score  University Rating  SOP  LOR   CGPA  \
Chance of Admit
0.92                     337          118                  4  4.5   4.5  9.65
0.76                     324          107                  4  4.0   4.5  8.87
0.72                     316          104                  3  3.0   3.5  8.00
0.80                     322          110                  3  3.5   2.5  8.67
0.65                     314          103                  2  2.0   3.0  8.21

                   Research  Serial number
Chance of Admit
0.92                      1              1
0.76                      1              2
0.72                      1              3
0.80                      1              4
0.65                      0              5
```

[87]:
```
#tambien puedo usar reset_index() para volver a los indices por defecto y␣
 ↪mantener el indice actual como columna:
df = df.reset_index()
df.head()
```

[87]:
```
    Chance of Admit   GRE Score  TOEFL Score  University Rating  SOP  LOR   \
0              0.92        337          118                  4  4.5   4.5
1              0.76        324          107                  4  4.0   4.5
2              0.72        316          104                  3  3.0   3.5
3              0.80        322          110                  3  3.5   2.5
4              0.65        314          103                  2  2.0   3.0

    CGPA  Research  Serial number
0  9.65         1              1
1  8.87         1              2
2  8.00         1              3
3  8.67         1              4
4  8.21         0              5
```

[88]:
```
#importo otro csv para mostrar index multi nivel
df = pd.read_csv('census.csv')
df.head()
```

[88]:
```
   SUMLEV  REGION  DIVISION  STATE  COUNTY   STNAME         CTYNAME  \
0      40       3         6      1       0  Alabama         Alabama
1      50       3         6      1       1  Alabama  Autauga County
2      50       3         6      1       3  Alabama  Baldwin County
3      50       3         6      1       5  Alabama  Barbour County
4      50       3         6      1       7  Alabama     Bibb County

   CENSUS2010POP  ESTIMATESBASE2010  POPESTIMATE2010  ...  RDOMESTICMIG2011  \
0        4779736            4780127          4785161  ...          0.002295
1          54571              54571            54660  ...          7.242091
2         182265             182265           183193  ...         14.832960
3          27457              27457            27341  ...         -4.728132
4          22915              22919            22861  ...         -5.527043

   RDOMESTICMIG2012  RDOMESTICMIG2013  RDOMESTICMIG2014  RDOMESTICMIG2015  \
0         -0.193196          0.381066          0.582002         -0.467369
1         -2.915927         -3.012349          2.265971         -2.530799
2         17.647293         21.845705         19.243287         17.197872
3         -2.500690         -7.056824         -3.904217        -10.543299
4         -5.068871         -6.201001         -0.177537          0.177258

   RNETMIG2011  RNETMIG2012  RNETMIG2013  RNETMIG2014  RNETMIG2015
0     1.030015     0.826644     1.383282     1.724718     0.712594
1     7.606016    -2.626146    -2.722002     2.592270    -2.187333
2    15.844176    18.559627    22.727626    20.317142    18.293499
3    -4.874741    -2.758113    -7.167664    -3.978583   -10.543299
4    -5.088389    -4.363636    -5.403729     0.754533     1.107861

[5 rows x 100 columns]
```

[90]:
```
df['SUMLEV'].unique() #veo los valores unicos de la columna sumlev
```

[90]:
```
array([40, 50])
```

[92]:
```
#me quedo solo con los valores de sumlev igual a 50
df = df[df['SUMLEV'] == 50]
df.head()
```

[92]:
```
   SUMLEV  REGION  DIVISION  STATE  COUNTY   STNAME         CTYNAME  \
1      50       3         6      1       1  Alabama  Autauga County
2      50       3         6      1       3  Alabama  Baldwin County
3      50       3         6      1       5  Alabama  Barbour County
4      50       3         6      1       7  Alabama     Bibb County
5      50       3         6      1       9  Alabama   Blount County

   CENSUS2010POP  ESTIMATESBASE2010  POPESTIMATE2010  ...  RDOMESTICMIG2011  \
1          54571              54571            54660  ...          7.242091
2         182265             182265           183193  ...         14.832960
3          27457              27457            27341  ...         -4.728132
```

```
4          22915             22919             22861  ...         -5.527043
5          57322             57322             57373  ...          1.807375

   RDOMESTICMIG2012  RDOMESTICMIG2013  RDOMESTICMIG2014  RDOMESTICMIG2015  \
1         -2.915927         -3.012349          2.265971         -2.530799
2         17.647293         21.845705         19.243287         17.197872
3         -2.500690         -7.056824         -3.904217        -10.543299
4         -5.068871         -6.201001         -0.177537          0.177258
5         -1.177622         -1.748766         -2.062535         -1.369970

   RNETMIG2011  RNETMIG2012  RNETMIG2013  RNETMIG2014  RNETMIG2015
1     7.606016    -2.626146    -2.722002     2.592270    -2.187333
2    15.844176    18.559627    22.727626    20.317142    18.293499
3    -4.874741    -2.758113    -7.167664    -3.978583   -10.543299
4    -5.088389    -4.363636    -5.403729     0.754533     1.107861
5     1.859511    -0.848580    -1.402476    -1.577232    -0.884411

[5 rows x 100 columns]
```

[93]:
```
#vamos a limpiar un poco y quedarnos solo con las columnas que nos interesan:
columns_to_keep = ['STNAME', 'CTYNAME', 'BIRTHS2010', 'BIRTHS2011',␣
 ↪'BIRTHS2012',
                   'BIRTHS2013', 'BIRTHS2014', 'BIRTHS2015', 'POPESTIMATE2013',␣
 ↪'POPESTIMATE2014', 'POPESTIMATE2015']
df = df[columns_to_keep]
df.head()
```

[93]:
```
    STNAME         CTYNAME  BIRTHS2010  BIRTHS2011  BIRTHS2012  BIRTHS2013  \
1  Alabama  Autauga County         151         636         615         574
2  Alabama  Baldwin County         517        2187        2092        2160
3  Alabama  Barbour County          70         335         300         283
4  Alabama     Bibb County          44         266         245         259
5  Alabama   Blount County         183         744         710         646

   BIRTHS2014  BIRTHS2015  POPESTIMATE2013  POPESTIMATE2014  POPESTIMATE2015
1         623         600            55038            55290            55347
2        2186        2240           195126           199713           203709
3         260         269            26973            26815            26489
4         247         253            22512            22549            22583
5         618         603            57734            57658            57673
```

[94]:
```
#ahora vamos a asignar dos index
df = df.set_index(['STNAME', 'CTYNAME'])
df.head()
```

[94]:
```
                      BIRTHS2010  BIRTHS2011  BIRTHS2012  BIRTHS2013  \
STNAME  CTYNAME
Alabama Autauga County         151         636         615         574
        Baldwin County         517        2187        2092        2160
```

```
                Barbour County          70         335         300         283
                Bibb County             44         266         245         259
                Blount County          183         744         710         646

                                BIRTHS2014  BIRTHS2015  POPESTIMATE2013  \
STNAME   CTYNAME
Alabama  Autauga County            623         600            55038
         Baldwin County           2186        2240           195126
         Barbour County            260         269            26973
         Bibb County               247         253            22512
         Blount County             618         603            57734

                                POPESTIMATE2014  POPESTIMATE2015
STNAME   CTYNAME
Alabama  Autauga County              55290            55347
         Baldwin County             199713           203709
         Barbour County              26815            26489
         Bibb County                 22549            22583
         Blount County               57658            57673
```

[96]:
```
#puedo consultar poniendo ambos indices, deben estar en orden
df.loc['Michigan', 'Washtenaw County'] #pido los datos del condado de washtenaw␣
 ↪(estado de michigan)
```

[96]:
```
BIRTHS2010            977
BIRTHS2011           3826
BIRTHS2012           3780
BIRTHS2013           3662
BIRTHS2014           3683
BIRTHS2015           3709
POPESTIMATE2013    354289
POPESTIMATE2014    357029
POPESTIMATE2015    358880
Name: (Michigan, Washtenaw County), dtype: int64
```

[99]:
```
#tambien puedo seleccionar dos condados para comparar los datos:
df.loc[[('Michigan', 'Washtenaw County'), ('Michigan', 'Wayne County')]]
```

[99]:
```
                            BIRTHS2010  BIRTHS2011  BIRTHS2012  BIRTHS2013  \
STNAME   CTYNAME
Michigan Washtenaw County          977        3826        3780        3662
         Wayne County             5918       23819       23270       23377

                            BIRTHS2014  BIRTHS2015  POPESTIMATE2013  \
STNAME   CTYNAME
Michigan Washtenaw County          3683        3709           354289
         Wayne County             23607       23586          1775713

                            POPESTIMATE2014  POPESTIMATE2015
```

```
STNAME    CTYNAME
Michigan Washtenaw County           357029           358880
         Wayne County              1766008          1759335
```

7 Missing Values

[100]:
```
import pandas as pd
df = pd.read_csv('class_grades.csv')
df.head(10)
```

[100]:

	Prefix	Assignment	Tutorial	Midterm	TakeHome	Final
0	5	57.14	34.09	64.38	51.48	52.50
1	8	95.05	105.49	67.50	99.07	68.33
2	8	83.70	83.17	NaN	63.15	48.89
3	7	NaN	NaN	49.38	105.93	80.56
4	8	91.32	93.64	95.00	107.41	73.89
5	7	95.00	92.58	93.12	97.78	68.06
6	8	95.05	102.99	56.25	99.07	50.00
7	7	72.85	86.85	60.00	NaN	56.11
8	8	84.26	93.10	47.50	18.52	50.83
9	7	90.10	97.55	51.25	88.89	63.61

[101]:
```
#utilizo la funcion isnull() para crear una mascara:
mask = df.isnull()
mask.head(10)
```

[101]:

	Prefix	Assignment	Tutorial	Midterm	TakeHome	Final
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	True	False	False
3	False	True	True	False	False	False
4	False	False	False	False	False	False
5	False	False	False	False	False	False
6	False	False	False	False	False	False
7	False	False	False	False	True	False
8	False	False	False	False	False	False
9	False	False	False	False	False	False

[102]:
```
#con el dropna() elimino todas las filas que tienen algn valor vacio
df.dropna().head(10)
```

[102]:

	Prefix	Assignment	Tutorial	Midterm	TakeHome	Final
0	5	57.14	34.09	64.38	51.48	52.50
1	8	95.05	105.49	67.50	99.07	68.33
4	8	91.32	93.64	95.00	107.41	73.89
5	7	95.00	92.58	93.12	97.78	68.06
6	8	95.05	102.99	56.25	99.07	50.00
8	8	84.26	93.10	47.50	18.52	50.83

```
9        7       90.10     97.55    51.25     88.89  63.61
10       7       80.44     90.20    75.00     91.48  39.72
12       8       97.16    103.71    72.50     93.52  63.33
13       7       91.28     83.53    81.25     99.81  92.22
```

[108]:
```
#con fillna() puede rellenar los valores nulos con algun valor:
df.fillna(0, inplace = True) #el inplace hace que se modifiquen los datos␣
 ↪originales, no devuelve una copia
df.head(10)
```

[108]:
```
   Prefix  Assignment  Tutorial  Midterm  TakeHome  Final
0       5       57.14     34.09    64.38     51.48  52.50
1       8       95.05    105.49    67.50     99.07  68.33
2       8       83.70     83.17     0.00     63.15  48.89
3       7        0.00      0.00    49.38    105.93  80.56
4       8       91.32     93.64    95.00    107.41  73.89
5       7       95.00     92.58    93.12     97.78  68.06
6       8       95.05    102.99    56.25     99.07  50.00
7       7       72.85     86.85    60.00      0.00  56.11
8       8       84.26     93.10    47.50     18.52  50.83
9       7       90.10     97.55    51.25     88.89  63.61
```

[109]:
```
df = pd.read_csv('log.csv')
df.head(10)
```

[109]:
```
         time    user       video  playback position paused  volume
0  1469974424  cheryl  intro.html                  5  False    10.0
1  1469974454  cheryl  intro.html                  6    NaN     NaN
2  1469974544  cheryl  intro.html                  9    NaN     NaN
3  1469974574  cheryl  intro.html                 10    NaN     NaN
4  1469977514     bob  intro.html                  1    NaN     NaN
5  1469977544     bob  intro.html                  1    NaN     NaN
6  1469977574     bob  intro.html                  1    NaN     NaN
7  1469977604     bob  intro.html                  1    NaN     NaN
8  1469974604  cheryl  intro.html                 11    NaN     NaN
9  1469974694  cheryl  intro.html                 14    NaN     NaN
```

[110]:
```
#en el sistema que tomo estos datos, los valores de paused y volume no se␣
 ↪actualizan si no sufren cambios. Para ello se puede
#utilizar ffill (foward filling) que actualiza un na de una celda en particular␣
 ↪por el valor de la fila anterior. bfill
#(backward filling) es lo mismo pero al reves.
```

[111]:
```
df = df.set_index('time') #seteo al tiempo de indice
df = df.sort_index() #ordeno por el indice
df.head()
```

[111]:
```
             user          video  playback position paused  volume
time
1469974424  cheryl     intro.html                  5  False    10.0
```

```
1469974424     sue  advanced.html                23  False    10.0
1469974454  cheryl     intro.html                 6    NaN     NaN
1469974454     sue  advanced.html                24    NaN     NaN
1469974484  cheryl     intro.html                 7    NaN     NaN
```

[112]:
```
#aca podemos ver que el index no tiene que ser necesariamente unico. Ahora␣
 ↪vamos a hacer un index multi nivel con time y user:
df = df.reset_index()
df = df.set_index(['time', 'user'])
df.head()
```

[112]:
```
                          video  playback position paused  volume
time       user
1469974424 cheryl     intro.html                  5  False    10.0
           sue     advanced.html                 23  False    10.0
1469974454 cheryl     intro.html                  6    NaN     NaN
           sue     advanced.html                 24    NaN     NaN
1469974484 cheryl     intro.html                  7    NaN     NaN
```

[114]:
```
#ahora que la lista esta ordenada puedo usar ffill:
df = df.fillna(method = 'ffill')
df.head()
```

[114]:
```
                          video  playback position  paused  volume
time       user
1469974424 cheryl     intro.html                  5   False    10.0
           sue     advanced.html                 23   False    10.0
1469974454 cheryl     intro.html                  6   False    10.0
           sue     advanced.html                 24   False    10.0
1469974484 cheryl     intro.html                  7   False    10.0
```

[115]:
```
#otros ejemplos con el siguiente df:
df = pd.DataFrame({'A': [1, 1, 2, 3, 4],
                   'B': [3, 6, 3, 8, 9],
                   'C': ['a', 'b', 'c', 'd', 'e']})
df
```

[115]:
```
   A  B  C
0  1  3  a
1  1  6  b
2  2  3  c
3  3  8  d
4  4  9  e
```

[117]:
```
df.replace(1, 100) #con replace indico que quiero reemplazar los 1 por 100
```

[117]:
```
     A  B  C
0  100  3  a
1  100  6  b
2    2  3  c
3    3  8  d
```

```
4    4  9  e
```

[119]:
```
df.replace([1, 3], [100, 300]) #reemplazo los 1 por 100 y los 3 por 300
```

[119]:
```
     A    B  C
0  100  300  a
1  100    6  b
2    2  300  c
3  300    8  d
4    4    9  e
```

[3]:
```
import pandas as pd
```

[17]:
```
#volvems al archivo log
df = pd.read_csv('log.csv')
df
```

[17]:

	time	user	video	playback position	paused	volume
0	1469974424	cheryl	intro.html	5	False	10.0
1	1469974454	cheryl	intro.html	6	NaN	NaN
2	1469974544	cheryl	intro.html	9	NaN	NaN
3	1469974574	cheryl	intro.html	10	NaN	NaN
4	1469977514	bob	intro.html	1	NaN	NaN
5	1469977544	bob	intro.html	1	NaN	NaN
6	1469977574	bob	intro.html	1	NaN	NaN
7	1469977604	bob	intro.html	1	NaN	NaN
8	1469974604	cheryl	intro.html	11	NaN	NaN
9	1469974694	cheryl	intro.html	14	NaN	NaN
10	1469974724	cheryl	intro.html	15	NaN	NaN
11	1469974454	sue	advanced.html	24	NaN	NaN
12	1469974524	sue	advanced.html	25	NaN	NaN
13	1469974424	sue	advanced.html	23	False	10.0
14	1469974554	sue	advanced.html	26	NaN	NaN
15	1469974624	sue	advanced.html	27	NaN	NaN
16	1469974654	sue	advanced.html	28	NaN	5.0
17	1469974724	sue	advanced.html	29	NaN	NaN
18	1469974484	cheryl	intro.html	7	NaN	NaN
19	1469974514	cheryl	intro.html	8	NaN	NaN
20	1469974754	sue	advanced.html	30	NaN	NaN
21	1469974824	sue	advanced.html	31	NaN	NaN
22	1469974854	sue	advanced.html	32	NaN	NaN
23	1469974924	sue	advanced.html	33	NaN	NaN
24	1469977424	bob	intro.html	1	True	10.0
25	1469977454	bob	intro.html	1	NaN	NaN
26	1469977484	bob	intro.html	1	NaN	NaN
27	1469977634	bob	intro.html	1	NaN	NaN
28	1469977664	bob	intro.html	1	NaN	NaN
29	1469974634	cheryl	intro.html	12	NaN	NaN
30	1469974664	cheryl	intro.html	13	NaN	NaN

```
31  1469977694     bob     intro.html                  1    NaN     NaN
32  1469977724     bob     intro.html                  1    NaN     NaN
```

[18]:
```
#tambien se pueden hacer reemplazos usando regex. Supongamos que queremos␣
 ↪cambiar todos los valores de video terminados en
#'.html' por 'webpage'.
df.replace(to_replace = '.*.html$', value = 'webpage', regex = True)
#el $ es para coincidir con el final de la cadena mientras que el .* es para␣
 ↪coincidir con todo el resto, sea lo que sea
```

[18]:

	time	user	video	playback position	paused	volume
0	1469974424	cheryl	webpage	5	False	10.0
1	1469974454	cheryl	webpage	6	NaN	NaN
2	1469974544	cheryl	webpage	9	NaN	NaN
3	1469974574	cheryl	webpage	10	NaN	NaN
4	1469977514	bob	webpage	1	NaN	NaN
5	1469977544	bob	webpage	1	NaN	NaN
6	1469977574	bob	webpage	1	NaN	NaN
7	1469977604	bob	webpage	1	NaN	NaN
8	1469974604	cheryl	webpage	11	NaN	NaN
9	1469974694	cheryl	webpage	14	NaN	NaN
10	1469974724	cheryl	webpage	15	NaN	NaN
11	1469974454	sue	webpage	24	NaN	NaN
12	1469974524	sue	webpage	25	NaN	NaN
13	1469974424	sue	webpage	23	False	10.0
14	1469974554	sue	webpage	26	NaN	NaN
15	1469974624	sue	webpage	27	NaN	NaN
16	1469974654	sue	webpage	28	NaN	5.0
17	1469974724	sue	webpage	29	NaN	NaN
18	1469974484	cheryl	webpage	7	NaN	NaN
19	1469974514	cheryl	webpage	8	NaN	NaN
20	1469974754	sue	webpage	30	NaN	NaN
21	1469974824	sue	webpage	31	NaN	NaN
22	1469974854	sue	webpage	32	NaN	NaN
23	1469974924	sue	webpage	33	NaN	NaN
24	1469977424	bob	webpage	1	True	10.0
25	1469977454	bob	webpage	1	NaN	NaN
26	1469977484	bob	webpage	1	NaN	NaN
27	1469977634	bob	webpage	1	NaN	NaN
28	1469977664	bob	webpage	1	NaN	NaN
29	1469974634	cheryl	webpage	12	NaN	NaN
30	1469974664	cheryl	webpage	13	NaN	NaN
31	1469977694	bob	webpage	1	NaN	NaN
32	1469977724	bob	webpage	1	NaN	NaN

8 Example: Manipulating DataFrame

[9]:
```
import pandas as pd
df = pd.read_csv('presidents.csv')
df.head()
```

[9]:
```
   #          President             Born      Age atstart of presidency  \
0  1  George Washington  Feb 22, 1732[a]  57ãyears, 67ãdaysApr 30, 1789
1  2         John Adams  Oct 30, 1735[a]  61ãyears, 125ãdaysMar 4, 1797
2  3   Thomas Jefferson  Apr 13, 1743[a]  57ãyears, 325ãdaysMar 4, 1801
3  4      James Madison  Mar 16, 1751[a]  57ãyears, 353ãdaysMar 4, 1809
4  5       James Monroe     Apr 28, 1758  58ãyears, 310ãdaysMar 4, 1817

         Age atend of presidency Post-presidencytimespan          Died  \
0   65ãyears, 10ãdaysMar 4, 1797       2ãyears, 285ãdays  Dec 14, 1799
1  65ãyears, 125ãdaysMar 4, 1801      25ãyears, 122ãdays   Jul 4, 1826
2  65ãyears, 325ãdaysMar 4, 1809      17ãyears, 122ãdays   Jul 4, 1826
3  65ãyears, 353ãdaysMar 4, 1817      19ãyears, 116ãdays  Jun 28, 1836
4  66ãyears, 310ãdaysMar 4, 1825       6ãyears, 122ãdays   Jul 4, 1831

                  Age
0  67ãyears, 295ãdays
1  90ãyears, 247ãdays
2   83ãyears, 82ãdays
3  85ãyears, 104ãdays
4   73ãyears, 67ãdays
```

[10]:
```
#vamos a separar el nombre y apelldio a los presidentes
df['First'] = df['President'] #copio la columna presidente y le pongo first
df['First'] = df['First'].replace('[ ].*', '', regex = True )
#con '[ ].*' le digo que matchee con todo lo que esta despues del espacio y lo␣
 ↪cambie por el segundo parametro ''.
df.head()
```

[10]:
```
   #          President             Born      Age atstart of presidency  \
0  1  George Washington  Feb 22, 1732[a]  57ãyears, 67ãdaysApr 30, 1789
1  2         John Adams  Oct 30, 1735[a]  61ãyears, 125ãdaysMar 4, 1797
2  3   Thomas Jefferson  Apr 13, 1743[a]  57ãyears, 325ãdaysMar 4, 1801
3  4      James Madison  Mar 16, 1751[a]  57ãyears, 353ãdaysMar 4, 1809
4  5       James Monroe     Apr 28, 1758  58ãyears, 310ãdaysMar 4, 1817

         Age atend of presidency Post-presidencytimespan          Died  \
0   65ãyears, 10ãdaysMar 4, 1797       2ãyears, 285ãdays  Dec 14, 1799
1  65ãyears, 125ãdaysMar 4, 1801      25ãyears, 122ãdays   Jul 4, 1826
2  65ãyears, 325ãdaysMar 4, 1809      17ãyears, 122ãdays   Jul 4, 1826
3  65ãyears, 353ãdaysMar 4, 1817      19ãyears, 116ãdays  Jun 28, 1836
4  66ãyears, 310ãdaysMar 4, 1825       6ãyears, 122ãdays   Jul 4, 1831
```

```
                    Age   First
0  67ãyears, 295ãdays  George
1  90ãyears, 247ãdays    John
2   83ãyears, 82ãdays  Thomas
3  85ãyears, 104ãdays   James
4   73ãyears, 67ãdays   James
```

[11]:
```
#hay mejores maneras de hacer esto, como la funcion apply(). Vamos a borrar la␣
 ↪columna first:
del(df['First'])

def splitname(row): #creo funcion
    row['First'] = row['President'].split(' ')[0] #extraigo el primer nombre y␣
 ↪genero la nueva columna
    row['Last'] = row['President'].split(' ')[-1] #extraigo el apellido y␣
 ↪genero la nueva columna
    return(row)
df = df.apply(splitname, axis = 'columns')
df.head()
```

[11]:
```
   #          President              Born      Age atstart of presidency  \
0  1  George Washington  Feb 22, 1732[a]  57ãyears, 67ãdaysApr 30, 1789
1  2         John Adams  Oct 30, 1735[a]  61ãyears, 125ãdaysMar 4, 1797
2  3   Thomas Jefferson  Apr 13, 1743[a]  57ãyears, 325ãdaysMar 4, 1801
3  4      James Madison  Mar 16, 1751[a]  57ãyears, 353ãdaysMar 4, 1809
4  5       James Monroe     Apr 28, 1758  58ãyears, 310ãdaysMar 4, 1817

         Age atend of presidency Post-presidencytimespan          Died  \
0   65ãyears, 10ãdaysMar 4, 1797       2ãyears, 285ãdays  Dec 14, 1799
1  65ãyears, 125ãdaysMar 4, 1801      25ãyears, 122ãdays   Jul 4, 1826
2  65ãyears, 325ãdaysMar 4, 1809      17ãyears, 122ãdays   Jul 4, 1826
3  65ãyears, 353ãdaysMar 4, 1817      19ãyears, 116ãdays  Jun 28, 1836
4  66ãyears, 310ãdaysMar 4, 1825       6ãyears, 122ãdays   Jul 4, 1831

                    Age   First        Last
0  67ãyears, 295ãdays  George  Washington
1  90ãyears, 247ãdays    John       Adams
2   83ãyears, 82ãdays  Thomas   Jefferson
3  85ãyears, 104ãdays   James     Madison
4   73ãyears, 67ãdays   James      Monroe
```

[12]:
```
#elimino las columnas first y last:
del(df['First'])
del(df['Last'])
```

[13]:
```
#ahora vamos a utilizar la funcion extract():

pattern = '(^[\w]*)(?:.* )([\w]*$)' #le digo que quiero 3 grupos. El primero␣
 ↪anclado al inicio, cualquier tipo de caracteres.
```

```
#Al segundo le pongo ?: porque este grupo no quiero retornarlo, no me interesa␣
 ↪(cualquier numero de caracteres seguido de un
#espacio). En el tercero quiero cualquier tipo de caracteres que esten anclados␣
 ↪al final.

df['President'].str.extract(pattern).head() #utilizo extract pasandole el␣
 ↪patron
```

[13]:
```
         0           1
0   George  Washington
1     John       Adams
2   Thomas   Jefferson
3    James     Madison
4    James      Monroe
```

[15]:
```
#si le pongo nombre a los grupos del patron, se pasaran a los nombres de las␣
 ↪columnas.
pattern = '(?P<First>^[\w]*)(?:.* )(?P<Last>[\w]*$)'
names = df['President'].str.extract(pattern).head()
names
```

[15]:
```
     First        Last
0   George  Washington
1     John       Adams
2   Thomas   Jefferson
3    James     Madison
4    James      Monroe
```

[16]:
```
#ahora copio esto al df original:
df['First'] = names['First']
df['Last'] = names['Last']
df.head()
```

[16]:
```
   #          President              Born      Age atstart of presidency  \
0  1  George Washington  Feb 22, 1732[a]  57ãyears, 67ãdaysApr 30, 1789
1  2         John Adams  Oct 30, 1735[a]  61ãyears, 125ãdaysMar 4, 1797
2  3   Thomas Jefferson  Apr 13, 1743[a]  57ãyears, 325ãdaysMar 4, 1801
3  4      James Madison  Mar 16, 1751[a]  57ãyears, 353ãdaysMar 4, 1809
4  5       James Monroe     Apr 28, 1758  58ãyears, 310ãdaysMar 4, 1817

         Age atend of presidency Post-presidencytimespan          Died  \
0   65ãyears, 10ãdaysMar 4, 1797       2ãyears, 285ãdays  Dec 14, 1799
1  65ãyears, 125ãdaysMar 4, 1801      25ãyears, 122ãdays   Jul 4, 1826
2  65ãyears, 325ãdaysMar 4, 1809      17ãyears, 122ãdays   Jul 4, 1826
3  65ãyears, 353ãdaysMar 4, 1817      19ãyears, 116ãdays  Jun 28, 1836
4  66ãyears, 310ãdaysMar 4, 1825       6ãyears, 122ãdays   Jul 4, 1831

                  Age   First        Last
0  67ãyears, 295ãdays  George  Washington
```

```
1  90ãyears, 247ãdays    John       Adams
2   83ãyears, 82ãdays  Thomas   Jefferson
3  85ãyears, 104ãdays   James     Madison
4   73ãyears, 67ãdays   James      Monroe
```

[17]:
```
#Ahora vamos a limpiar la columna Born:
df['Born'] = df['Born'].str.extract('([\w]{3} [\w]{1,2}, [\w]{4})')
df['Born'].head()
```

[17]:
```
0    Feb 22, 1732
1    Oct 30, 1735
2    Apr 13, 1743
3    Mar 16, 1751
4    Apr 28, 1758
Name: Born, dtype: object
```

[18]:
```
#se puede mejorar el formato de la siguiente manera:
df['Born'] = pd.to_datetime(df['Born']) #le cambio el formato usando una␣
 ↪funcion de pandas
df['Born'].head()
```

[18]:
```
0   1732-02-22
1   1735-10-30
2   1743-04-13
3   1751-03-16
4   1758-04-28
Name: Born, dtype: datetime64[ns]
```

[20]:
```
#para mas informacion de este tema ver los modulos str de pandas.
```

[ ]:
```

```