Entrada y Salida con Pandas

Pandas ofrece un amplico conjunto de funciones para el intercambio de datos. Existen muchos formatos para el intercambio de información. Estos formatos se puden clasificar en función si son legibles por los humanos o no. Algunos ejemplos son:

- Formato de texto: Permiten editarlos directamente por los humanos. Algunos ejemplos son `CSV` y `JSON`.
- Formato binarios: No son legibles por los humanos. Algunos ejemplos son `XLS` y `HDF5`.

En este cuaderno vamos a estudiar el formato `CSV` como ejemplo de formato texto y el formato `XLS` como ejemplo de formato binario.

Este cuaderno se basa en esta documentación oficial.

Inicialización del entorno.

Lo primero será importar el paquete Pandas con alias pd. Posteriormente visualizamos la versión usada de Pandas ya que es un dato importante para consultar la documentación. En el momento de editar este notebook la versión de pandas es: 1.4.3

Además para facilitar los ejercicios también importamos el paquete `numpy` con el alias `np`. También imporamos el objeto StringIO para simular ficheros tipo texto.

In [97]:
```
import pandas as pd
import numpy as np
np.set_printoptions(floatmode='fixed', precision=3)
print('Pandas versión: ', pd.__version__)
from io import StringIO, BytesIO
```

```
Pandas versión:  1.4.3
```

Formato CSV.

El formato CSV es uno de los formatos más comunes para intercambiar conjuntos de datos. Se recomienda leer la referencia Comma-separated values.

Su principal característica es que toda la información está codificada en forma de texto, separando cada valor de columna por un carácter separdor, normalmente la coma `','`, de ahí su nombre.

Aunque no es requerido, la primera línea del fichero suele tener los nombres de las columnas. Veamos un ejemplo de fichero CSV:

```
Apellidos,Nombre,Teoría,Práctica,Total
```

```
Anguita Pérez,José Luis,4.5,6.1,5.3
Díez Expósito,María,6.4,8.0,7.2
```

Las principales ventajas del formato CSV son: es un formato portable entre sistemas operativos y arquitecturas de máquinas, es flexible, se puede editar directamente por un humano y puede ser manipulado fácilmente con comandos del sistema operativo, en especial, sistemas tipo Unix.

Las principales desventajas son: no ofrece compresión de datos, es lento de procesar ya que hay que convertir los valores no numéricos a texto y viceversa, suele estar "localizado", por ejemplo el formato de fecha, o usar `','` para separar la parte decimal, y puede utilizar distintas codificaciones de texto (por ejemplo para letras acentuadas, la `ñ` en castellano, etc.).

Lectura de un fichero CSV.

La función pandas para leer un conjunto de datos en formato CSV sería `read_csv()`. Se recomienda leer la referencia antes de realizar los ejercicios.

Ejercicio 01: Dado un conjunto de datos almacenado en un fichero en formato CSV, se requiere cargarlo en la variable `data` e imprimir la descripción del mismo.

La salida debería ser algo parecido a lo siguiente:

```
                          A1   A2    A3        A4       A5   A6      A7
\
0                       name  mfr  type  calories  protein  fat  sodium
1                  100% Bran    N     C        70        4    1     130
2          100% Natural Bran  NaN     C       120        3    5      15
3                   All-Bran    K     C        70        4    1    n.e.
4  All-Bran with Extra Fiber    K     C        50        4    0     140

      A8     A9     A10     A11       A12    A13     A14   A15
A16
0  fiber  carbo  sugars  potass  vitamins  shelf  weight  cups
rating
1     10      .       6     280        25      3       1  0.33
68.402973
2      2      8       8     135         0      3       1     1
33.983679
3      9      7       5     320        25      3       1  0.33
59.425505
4     14      8       0     330        25      3       1   0.5
93.704912
```

In [98]:
```
#Extracto del dataset https://www.kaggle.com/crawford/80-cereals/data
data_file = StringIO(
u'''A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13,A14,A15,A16
name,mfr,type,calories,protein,fat,sodium,fiber,carbo,sugars,potass,vitamins,shelf,weigh
100% Bran,N,C,70,4,1,130,10,.,6,280,25,3,1,0.33,68.402973
100% Natural Bran,,C,120,3,5,15,2,8,8,135,0,3,1,1,33.983679
All-Bran,K,C,70,4,1,n.e.,9,7,5,320,25,3,1,0.33,59.425505
All-Bran with Extra Fiber,K,C,50,4,0,140,14,8,0,330,25,3,1,0.5,93.704912
Almond Delight,R,C,110,2,2,200,1,14,8,-1,25,3,1,0.75,34.384843
Apple Cinnamon Cheerios,G,C,110,2,2,180,1.5,10.5,10,70,25,1,1,0.75,29.509541
Apple Jacks,K,C,110,2,0,125,1,11,14,30,25,2,1,1,33.174094
''')
```

```
data = pd.DataFrame()

#Pon tu código aquí.
#Sugerencia: utiliza la función read_csv()

#

print(data.head())
```

```
Empty DataFrame
Columns: []
Index: []
```

En general Pandas va a hacer todo el trabajo por nosotros, por ejemplo, usa la primera fila del fichero como nombre de las columnas, pero podemos ayudarle un poco si es necesario.

En este caso vemos como la fuente de datos tiene añadida una primera fila con nombres de columna que no tienen mucho sentido y que la segunda fila si tienen nombres de columna significativos. Podemos indicar que cuando se importe el fichero se salte la primera fila usando el parámetro `skiprows`.

Ejercicio 02: Dado un conjunto de datos almacenado en un fichero en formato CSV, se requiere cargarlo en la variable `data` e imprimir la descripción del mismo. Se requiere saltar la primera línea del fichero ya que se corresponde con nombres de columna no significativos, siendo la segunda línea del fichero donde sí se encuentran los nombres de columna requeridos.

La salida debería ser algo parecido a lo siguiente:

```
                         name  mfr type  calories  protein  fat sodium
fiber  \
0                  100% Bran    N    C        70        4    1    130
10.0
1          100% Natural Bran  NaN    C       120        3    5     15
2.0
2                   All-Bran    K    C        70        4    1   n.e.
9.0
3  All-Bran with Extra Fiber    K    C        50        4    0    140
14.0
4             Almond Delight    R    C       110        2    2    200
1.0

  carbo  sugars  potass  vitamins  shelf  weight  cups     rating
0     .       6     280        25      3       1  0.33  68.402973
1     8       8     135         0      3       1  1.00  33.983679
2     7       5     320        25      3       1  0.33  59.425505
3     8       0     330        25      3       1  0.50  93.704912
4    14       8      -1        25      3       1  0.75  34.384843
```

In [99]:
```
#Extracto del dataset https://www.kaggle.com/crawford/80-cereals/data
data_file = StringIO(
u'''A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13,A14,A15,A16
name,mfr,type,calories,protein,fat,sodium,fiber,carbo,sugars,potass,vitamins,shelf,weigh
100% Bran,N,C,70,4,1,130,10,.,6,280,25,3,1,0.33,68.402973
100% Natural Bran,,C,120,3,5,15,2,8,8,135,0,3,1,1,33.983679
All-Bran,K,C,70,4,1,n.e.,9,7,5,320,25,3,1,0.33,59.425505
All-Bran with Extra Fiber,K,C,50,4,0,140,14,8,0,330,25,3,1,0.5,93.704912
Almond Delight,R,C,110,2,2,200,1,14,8,-1,25,3,1,0.75,34.384843
Apple Cinnamon Cheerios,G,C,110,2,2,180,1.5,10.5,10,70,25,1,1,0.75,29.509541
Apple Jacks,K,C,110,2,0,125,1,11,14,30,25,2,1,1,33.174094
''')
```

```
data = pd.DataFrame()

#Pon tu código aquí.
#Sugerencia: utilza la función read_csv(). Utiliza el argumento skiprows de forma
#conveniente.

#

print(data.head())
```

```
Empty DataFrame
Columns: []
Index: []
```

Suele ser algo habitual en un conjunto de datos que no se disponga de un valor válido para alguna caracerística de un indivio. Por ejemplo si se observa en la salida del ejemplo anterior, en la línea 1, la columna `mfr` aparece un NaN. Esto es debido a que para esa línea, no se ha dado un valor de columna correspondiente, por lo que Pandas utilizar el valor `NaN` para reflejar este hecho.

Además de dejar vacía una columna, suele ser común utilizar un texto para indicar que no hay un dato válido. Por ejemplo puede observarse en la salida del ejemplo anterior el texto `'n.e.'` (no especifiado) en la fila 2, columna `sodium` o el texto `'.'` en la fila 0, columna `carbo`. En estos casos hay que ayudar a Pandas a detectarlos para que pueda sustituirlos por el valor Pandas NaN utilizando el parámetro `na_values`.

Ejercicio 03: Dado un conjunto de datos almacenado en un fichero en formato CSV, se requiere cargarlo en la variable `data` e imprimir la descripción del mismo. Se requiere saltar la primera línea del fichero ya que se corresponde con nombres de columna no significativos, siendo la segunda línea del fichero donde sí se encuentran los nombres de columna requeridos. Además se sabe que los textos `'n.e.'` y `'.'` identifican valores no especificados y se requiere que sean traducidos al valor Pandas para gestionar valores incompletos (NaN).

La salida debería ser algo parecido a lo siguiente:

```
                         name  mfr type  calories  protein  fat  sodium
fiber  \
0                  100% Bran    N    C        70        4    1   130.0
10.0
1          100% Natural Bran  NaN    C       120        3    5    15.0
2.0
2                   All-Bran    K    C        70        4    1     NaN
9.0
3  All-Bran with Extra Fiber    K    C        50        4    0   140.0
14.0
4             Almond Delight    R    C       110        2    2   200.0
1.0

   carbo  sugars  potass  vitamins  shelf  weight  cups     rating
0    NaN       6     280        25      3       1  0.33  68.402973
1    8.0       8     135         0      3       1  1.00  33.983679
2    7.0       5     320        25      3       1  0.33  59.425505
3    8.0       0     330        25      3       1  0.50  93.704912
4   14.0       8      -1        25      3       1  0.75  34.384843
```

In [100…

```
#Extracto del datset https://www.kaggle.com/crawford/80-cereals/data
data_file = StringIO(
```

```
u'''A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13,A14,A15,A16
name,mfr,type,calories,protein,fat,sodium,fiber,carbo,sugars,potass,vitamins,shelf,weigh
100% Bran,N,C,70,4,1,130,10,.,6,280,25,3,1,0.33,68.402973
100% Natural Bran,,C,120,3,5,15,2,8,8,135,0,3,1,1,33.983679
All-Bran,K,C,70,4,1,n.e.,9,7,5,320,25,3,1,0.33,59.425505
All-Bran with Extra Fiber,K,C,50,4,0,140,14,8,0,330,25,3,1,0.5,93.704912
Almond Delight,R,C,110,2,2,200,1,14,8,-1,25,3,1,0.75,34.384843
Apple Cinnamon Cheerios,G,C,110,2,2,180,1.5,10.5,10,70,25,1,1,0.75,29.509541
Apple Jacks,K,C,110,2,0,125,1,11,14,30,25,2,1,1,33.174094
''')

data = pd.DataFrame()

#Pon tu código aquí.
#Sugerencia: utilza la función read_csv().
#Utiliza el argumento skiprows de forma conveniente.
#Utiliza el argumento na_values para indicar que los textos '.' y 'n.e.'
#identifican valores incompletos.

#

print(data.head())
```

```
Empty DataFrame
Columns: []
Index: []
```

Otro problema común es que los datos de entrada utilicen un formato de codificación de texto distinto al que se usa el sistema operativo donde vamos cargar los datos. Por ejemplo si los datos se generan en sistemas "Windows" normalmente se usará el formato de condificación unicode `iso8859_15` (en europa occidental), mientras que en sistemas Linux, es común utilizar el sistema `utf-8`.

Pandas genera un excepción si no puede importar un fichero porque el sistema de codificación de la entrada usado no coincide con el que se está usando (por defecto se utiliza `ascii`).

Opcionalmente, podremos indicarle a Pandas el sistema de codificación utilizado en la entrada para que traduzca al sistema de codificación activo. Para ello utilizamos el parámetro `encoding`.

Ejercicio 04: Dado un conjunto de datos almacenado en un fichero en formato CSV, se requiere cargarlo en la variable `data` e imprimir la descripción del mismo. Se requiere saltar la primera línea del fichero ya que se corresponde con nombres de columna no significativos, siendo la segunda línea del fichero donde sí se encuentran los nombres de columna requeridos. Además se sabe que los textos `'n.e.'` y `'.'` identifican valores no especificados y se requiere que sean traducidos al valor Pandas para gestionar valores incompletos (NaN). Se sabe que el sistema de codificación usado es utf-8 por lo que se debería indicar en el proceso de carga.

La salida debería ser algo parecido a lo siguiente:

```
                        name  mfr type  calories  protein  fat  sodium
fiber  \
0                  100% Bran    N    C        70        4    1   130.0
10.0
1          100% Natural Bran  NaN    C       120        3    5    15.0
2.0
2                   All-Bran    K    C        70        4    1     NaN
9.0
3  All-Bran with Extra Fiber    K    C        50        4    0   140.0
14.0
4             Almond Delight    R    C       110        2    2   200.0
```

```
1.0
```

	carbo	sugars	potass	vitamins	shelf	weight	cups	rating
0	NaN	6	280	25	3	1	0.33	68.402973
1	8.0	8	135	0	3	1	1.00	33.983679
2	7.0	5	320	25	3	1	0.33	59.425505
3	8.0	0	330	25	3	1	0.50	93.704912
4	14.0	8	-1	25	3	1	0.75	34.384843

In [101…

```
#Extracto del datset https://www.kaggle.com/crawford/80-cereals/data
data_file = StringIO(
u'''A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13,A14,A15,A16
name,mfr,type,calories,protein,fat,sodium,fiber,carbo,sugars,potass,vitamins,shelf,weigh
100% Bran,N,C,70,4,1,130,10,.,6,280,25,3,1,0.33,68.402973
100% Natural Bran,,C,120,3,5,15,2,8,8,135,0,3,1,1,33.983679
All-Bran,K,C,70,4,1,n.e.,9,7,5,320,25,3,1,0.33,59.425505
All-Bran with Extra Fiber,K,C,50,4,0,140,14,8,0,330,25,3,1,0.5,93.704912
Almond Delight,R,C,110,2,2,200,1,14,8,-1,25,3,1,0.75,34.384843
Apple Cinnamon Cheerios,G,C,110,2,2,180,1.5,10.5,10,70,25,1,1,0.75,29.509541
Apple Jacks,K,C,110,2,0,125,1,11,14,30,25,2,1,33.174094
''')

data = pd.DataFrame()

#Pon tu código aquí.
#Sugerencia: utilza la función read_csv().
#Utiliza el argumento skiprows de forma conveniente.
#Utiliza el argumento na_values para indicar que los textos '.' y 'n.e.'
#identifican valores incompletos.
#Utiliza el parámetro encoding para indicar que la entrada está codificaca
#con formato utf-8

#

print(data.head())
```

```
Empty DataFrame
Columns: []
Index: []
```

Otro problema común es que las magnitudes y fechas estén localizadas. Por ejemplo en España, se utiliza el carácter `','` para separar la parte fraccionaria mientras que en el mundo anglosajón se utiliza el caracter `'.'`. Además en este caso posible que se utilice otro caracter distinto a `','` para separar las columnas. Para indicar qué caracter se utiliza como separador de campo se utiliza el parámetro `sep`. Para indicar cómo se indica la parte decimal se utiliza el parámetro `decimal`.

Ejercicio 05: Dado un conjunto de datos almacenado en un fichero en formato CSV, se requiere cargarlo en la variable `data`. Se sabe que el sistema de codificación de texto usado es `utf-8` por lo que se debería indicar en el proceso de carga. Además los datos están localizados en "Español" usando `','` para separar la parte entera de la decimal de los números y que por este motivo se utiliza el caracter `':'` para separar columnas.

La salida debería ser algo parecido a lo siguiente:

	Apellidos	Nombre	Teoría	Práctica	Total
0	Anguita Pérez	José Luis	4.5	6.1	5.3
1	Díez Expósito	María	6.4	8.0	7.2
2	Fernández Muñoz	Antonia	7.0	5.4	6.2
3	García Pérez	José	6.3	8.5	7.4

In [102…

```
data_file = StringIO(
u'''Apellidos:Nombre:Teoría:Práctica:Total:Fecha
Anguita Pérez: José Luis: 4,5: 6,1: 5,3:05-10-2021
Díez Expósito: María: 6,4: 8,0: 7,2:15-01-2021
Fernández Muñoz: Antonia: 7,0: 5,4: 6,2:04-06-2021
García Pérez: José: 6,3: 8,5: 7,4:25-09-2021
''')

data = pd.DataFrame()

#Pon tu código aquí.
#Sugerencia: utiliza la función read_csv().
#Utiliza el parámetro encoding para indicar que la entrada está codificaca
#   con formato utf-8
#Utiliza el parámetro 'sep' para indicar el caracter ':' de separación de columna.
#Utiliza el parámetro 'decimal' para indicar el caracter ',' usado para separar decimale

#

print(data.head())
```

```
Empty DataFrame
Columns: []
Index: []
```

Cuando una columna tiene un texto que representa una fecha, se puede ayudar a Pandas a decodificarlas para generar un valor `datatime`. Para ello se utiliza el parámetro `parse_dates`.

Ejercicio 06: Dado un conjunto de datos almacenado en un fichero en formato CSV, se requiere cargarlo en la variable `data`. Se sabe que el sistema de codificación de texto usado es `utf-8` por lo que se debería indicar en el proceso de carga. Además los datos están localizados en "Español" usando `','` para separar la parte entera de la decimal de los números y que por este motivo se utiliza el caracter `':'` para separar columnas. Hay una columna 'Fecha' que tiene valores temporales usado el formato dd-mm-año.

La salida debería ser algo parecido a lo siguiente:

```
          Apellidos      Nombre  Teoría  Práctica  Total      Fecha
   0    Anguita Pérez   José Luis     4.5       6.1    5.3 2021-10-05
   1    Díez Expósito       María     6.4       8.0    7.2 2021-01-15
   2  Fernández Muñoz     Antonia     7.0       5.4    6.2 2021-06-04
   3     García Pérez        José     6.3       8.5    7.4 2021-09-25

   dtype de la columna 'Fecha':  datetime64[ns]
```

In [103…

```
data_file = StringIO(
u'''Apellidos:Nombre:Teoría:Práctica:Total:Fecha
Anguita Pérez: José Luis: 4,5: 6,1: 5,3:05-10-2021
Díez Expósito: María: 6,4: 8,0: 7,2:15-01-2021
Fernández Muñoz: Antonia: 7,0: 5,4: 6,2:04-06-2021
García Pérez: José: 6,3: 8,5: 7,4:25-09-2021
''')

data = pd.DataFrame()

#Pon tu código aquí.
#Sugerencia: utilza la función read_csv().
#   Utiliza el parámetro 'parse_dates' para indicar la columna
#   con fechas.
#   Utiliza el parámetro 'dayfirst' para indicar que el formato
#   de fecha ponen primero el día y después el mes.
```

```
#

print(data.head())
if data.get('Fecha') is not None:
    print("\ndtype de la columna 'Fecha': ", data['Fecha'].dtype)
```

```
Empty DataFrame
Columns: []
Index: []
```

Escribiendo en formato CSV.

Para almacenar en un fichero con formato CVS un objeto Series o DataFrame usamos el método `to_csv()`.

Ejercicio 07: Dado un dataframe `df` se desea almacenarlo usando el format CSV. Se requiere usar codificación 'utf-8'.

La salida debería ser algo parecido a lo siguiente:

```
El fichero generado es:

,Apellidos,Nombre,Total,Fecha
0,Anguita Pérez,José Luis,5.3,2021-10-05
1,Anguita Pérez,María,7.2,2021-01-15
2,Fernández Muñoz,Antonia,6.2,2021-06-04
3,García Pérez,José,7.4,2021-09-25
```

In [104…

```
file = StringIO()
df = pd.DataFrame({'Apellidos':['Anguita Pérez', 'Anguita Pérez',
                                'Fernández Muñoz', 'García Pérez'],
                   'Nombre':['José Luis', 'María', 'Antonia', 'José'],
                   'Total':[5.3,7.2,6.2,7.4],
                   'Fecha':pd.to_datetime(['2021-10-05','2021-01-15',
                                           '2021-06-04','2021-09-25'], dayfirst=True)})
#Pon tu código aquí.
#Sugerencia: usa el método to_csv().
#Utiliza el argumento 'encoding' para indicar utf-8.

#

file.seek(0) #simular reapertura de archivo.
print("El fichero generado es:\n")
for line in file:
    print(line, end='')
```

```
El fichero generado es:
```

Observa que el índice de filas se guarda también. Si no es necesario guardarlo, podemos usar el parámetro `index=False`.

Ejercicio 08: Dado un dataframe `df` se desea almacenarlo usando el format CSV. Se requiere usar codificación 'utf-8'. El índice de las filas no es requerido.

La salida debería ser algo parecido a lo siguiente:

```
El fichero generado es:
```

```
Apellidos,Nombre,Total,Fecha
Anguita Pérez,José Luis,5.3,2021-10-05
Anguita Pérez,María,7.2,2021-01-15
Fernández Muñoz,Antonia,6.2,2021-06-04
García Pérez,José,7.4,2021-09-25
```

In [105…

```
file = StringIO()
df = pd.DataFrame({'Apellidos':['Anguita Pérez', 'Anguita Pérez',
                                'Fernández Muñoz', 'García Pérez'],
                   'Nombre':['José Luis', 'María', 'Antonia', 'José'],
                   'Total':[5.3,7.2,6.2,7.4],
                   'Fecha':pd.to_datetime(['2021-10-05','2021-01-15',
                                          '2021-06-04','2021-09-25'], dayfirst=True)})
#Pon tu código aquí.
#Sugerencia: usa el método to_csv().
#Utiliza el argumento 'index' para indicar que no se guarde el índice de filas.

#

file.seek(0) #simular reapertura de archivo.
print("El fichero generado es:\n")
for line in file:
    print(line, end='')
```

```
El fichero generado es:
```

También se puede indicar el caracter usado para separar los campos con el parámetro `del` y el caracter usado para separar la parte decimal con el parámetro `decimal`.

Ejercicio 09: Dado un dataframe `df` se desea almacenarlo usando el format CSV. Se requiere usar codificación 'utf-8'. El índice de las filas no es requerido. Se quiere usar el caracter `','` para separar la parte decimal del los número flotantes. Se caracter separador de columna será `':'`.

La salida debería ser algo parecido a lo siguiente:

```
El fichero generado es:

Apellidos:Nombre:Total:Fecha
Anguita Pérez:José Luis:5,3:2021-10-05
Anguita Pérez:María:7,2:2021-01-15
Fernández Muñoz:Antonia:6,2:2021-06-04
García Pérez:José:7,4:2021-09-25
```

In [106…

```
file = StringIO()
df = pd.DataFrame({'Apellidos':['Anguita Pérez', 'Anguita Pérez',
                                'Fernández Muñoz', 'García Pérez'],
                   'Nombre':['José Luis', 'María', 'Antonia', 'José'],
                   'Total':[5.3,7.2,6.2,7.4],
                   'Fecha':pd.to_datetime(['2021-10-05','2021-01-15',
                                          '2021-06-04','2021-09-25'], dayfirst=True)})
#Pon tu código aquí.
#Sugerencia: usa el método to_csv().
#Utiliza el argumento 'sep' para indicar el caracter separador de columnas.
#Utiliza el argumento 'decimal' para indicar cómo mostrar la parte decimal.

#

file.seek(0) #simular reapertura de archivo.
print("El fichero generado es:\n")
```

```
for line in file:
    print(line, end='')
```

```
El fichero generado es:
```

Si tenemos datos temporales, podemos indicar con el parámetro `date_format` el formato usado para escrbirlos (ver notación).

Ejercicio 10: Dado un dataframe `df` se desea almacenarlo usando el format CSV. Se requiere usar codificación 'utf-8'. El índice de las filas no es requerido. Se quiere usar el caracter `';'` para separar la parte decimal del los número flotantes. Se caracter separador de columna será ':'. Las fechas se requieren que se escriban con el formato "miércoles 12 de mayo de 2021, 23:30".

La salida debería ser algo parecido a lo siguiente:

```
El fichero generado es:

Apellidos:Nombre:Total:Fecha
Anguita Pérez:José Luis:5,3:"martes 05 de octubre del 2021, 00:00"
Anguita Pérez:María:7,2:"viernes 15 de enero del 2021, 00:00"
Fernández Muñoz:Antonia:6,2:"viernes 04 de junio del 2021, 00:00"
García Pérez:José:7,4:"sábado 25 de septiembre del 2021, 00:00"
```

In [107…

```
import locale
locale.setlocale(locale.LC_ALL,'es_ES.UTF-8')
file = StringIO()
df = pd.DataFrame({'Apellidos':['Anguita Pérez', 'Anguita Pérez',
                                'Fernández Muñoz', 'García Pérez'],
                   'Nombre':['José Luis', 'María', 'Antonia', 'José'],
                   'Total':[5.3,7.2,6.2,7.4],
                   'Fecha':pd.to_datetime(['2021-10-05','2021-01-15',
                                           '2021-06-04','2021-09-25'], dayfirst=True)})
#Pon tu código aquí.
#Sugerencia: usa el método to_csv().
#Utiliza el argumento 'date_format' para indicar el formato para poner la fecha.

#

file.seek(0) #simular reapertura de archivo.
print("El fichero generado es:\n")
for line in file:
    print(line, end='')
```

```
El fichero generado es:
```

Observa que las fechas se han puesto entre comillas `""` ya que se utiliza el caracter `';'` en su formato.

Formato XLS

El formato XLS hace referencia al formato utilizado originalmente por la hoja de cálculo Excel de Microsoft (tm). Pandas puede almacenar y escribir en este formato utilizando los paquetes Python:

- Formato '.xls' de Excel 2003 con el paquete `xlrd`.
- Formato '.xlsx' de Excel 2007+ con el paquete `openpyxl`.
- Formato '.xlsb' (formato binario) con el paquete `pyxlsb`.

Para realizar los ejercicios siguientes, se recomienda instalar los paquetes anteriores y descargar el fichero `'datos_covid_andalucia.xls'` accesible desde este enlace. Asegurate descargarlo en formato 'xls' y guardarlo en la misma carpeta donde tengas este cuaderno.

Leyendo datos desde formato XLS.

Para leer una fuente de datos en formato XLS usaremos la función `read_excel()` Esta función permite leer desde un fichero, o incluso desde una URL. Si la entrada consite un un libro de hojas, se puede indicar el nombre de la hoja que queremos cargar. Además podemos utilizar parámetros similares a `read_csv()` para gestionar la entrada.

Ejercicio 11: Se desea cargar la hoja de cálculo almacenada en el archivo `'datos_covid_andalucia.xls'`. Tras una inspección preliminar, se observa que las primeras 12 lineas no aportan información de interés.

La salida esperada debe ser algo parecido a lo siguiente:

```
Primeras líneas:
   Fecha diagnóstico Territorio  Confirmados PDIA  Total confirmados  \
0        06/05/2021  Andalucía               268                270
1               NaN    Almería                12                 12
2               NaN      Cádiz                 9                  9
3               NaN    Córdoba                17                 17
4               NaN    Granada                 7                  8

   Hospitalizados  UCI  Fallecidos
0               3    0           0
1               0    0           0
2               1    0           0
3               2    0           0
4               0    0           0

Tipos de las columnas:
 Fecha diagnóstico    object
Territorio           object
Confirmados PDIA      int64
Total confirmados     int64
Hospitalizados        int64
UCI                   int64
Fallecidos            int64
dtype: object
```

In [108…

```
df = pd.DataFrame()

#Pon tu código aquí.
#Sugerencia: usa la función read_excel() usando path de acceso al fichero.
#Sugerencia: usa el parámetro skiprows para ignorar las primeras 12 lineas.

#

print('Primeras líneas:\n', df.head())
print('\nTipos de las columnas:\n',df.dtypes)
```

```
Primeras líneas:
 Empty DataFrame
Columns: []
```

```
Index: []

Tipos de las columnas:
 Series([], dtype: object)
```

Para leer un libro con varias hojas de cálculo y poder acceder a cada una de ellas podemos utilizar el parámetro `sheet_name` indicando una o varias hojas (en este caso se devuelve un diccionario de objetos DataFrame, uno para cada hoja cargada). Cuando no se indica este parámetro se cargan todas las hojas contenidas en el archivo.

Ejercicio 12: Se desea cargar la hoja de cálculo `'Datos'` almacenada en el archivo `'datos_covid_andalucia.xls'`. Tras una inspección preliminar, se observa que las primeras 12 lineas no aportan información de interés.

La salida esperada debe ser algo parecido a lo siguiente:

```
   Primeras líneas de la hoja 'Datos':
      Fecha diagnóstico Territorio  Confirmados PDIA  Total confirmados  \
   0        06/05/2021  Andalucía               268                270
   1               NaN    Almería                12                 12
   2               NaN      Cádiz                 9                  9
   3               NaN    Córdoba                17                 17
   4               NaN    Granada                 7                  8

      Hospitalizados  UCI  Fallecidos
   0               3    0           0
   1               0    0           0
   2               1    0           0
   3               2    0           0
   4               0    0           0
```

In [109…

```
df = pd.DataFrame()

#Pon tu código aquí.
#Sugerencia: usa la función read_excel() usando path de acceso al fichero.
#Sugerencia: usa el parámetro skiprows para ignorar las primeras 12 lineas.

#

print('Primeras líneas de la hoja \'Datos\':\n', df.head())
```

```
Primeras líneas de la hoja 'Datos':
 Empty DataFrame
Columns: []
Index: []
```

Usar directamente `read_excel()` para leer más de una hoja de un libro tiene el incoveniente de que los parámetro usados para la carga se aplican igual a todas y puede ocurrir que esto no es lo requerido, por lo que habría que cargar cada hoja por separado usando distintas llamadas a `read_excel()` seleccionando cada hoja con el parámetro `sheet_name`. Esta forma de trabajar puede ser poco eficiente.

Alternativamente, en este caso, sería más eficiente cargar un libro usando un objeto `pandas.ExcelFile` y usando este objeto, cargar cada hoja (con el método `parse()`). La secuencia de pasos serían:

1. Crear un objeto `obj = pandas.ExcelFile('fichero.xls')`
2. El atributo `obj.sheet_names` es una lista con los nombres de las hojas almacenadas en el libro.

3. Para cada hoja, cargar con `obj.parse()` indicando el nombre de la hoja y las opciones específicas para esa hoja.

Ejercicio 13: Se desea cargar la hoja de cálculo `'Datos'` almacenada en el archivo `'datos_covid_andalucia.xls'`. Tras una inspección preliminar, se observa que las primeras 12 lineas no aportan información de interés. Utiliza un objeto `ExcelFile` para cargar el libro.

La salida esperada debe ser algo parecido a lo siguiente:

```
Hojas en el libro:  ['Datos']

Primeras líneas de la hoja 'Datos':
   Fecha diagnóstico Territorio  Confirmados PDIA  Total confirmados  \
0        06/05/2021  Andalucía               268                270
1               NaN    Almería                12                 12
2               NaN      Cádiz                 9                  9
3               NaN    Córdoba                17                 17
4               NaN    Granada                 7                  8

   Hospitalizados  UCI  Fallecidos
0               3    0           0
1               0    0           0
2               1    0           0
3               2    0           0
4               0    0           0
```

In [110…

```
filename = 'data/datos_covid_andalucia.xls' #Adapta el camino al fichero.

book = None
#Pon tu código aquí.
#Sugerencia: crea un objeto ExcelFile con el camino al fichero a cargar.

#
hojas = []

#Pon tu código aquí.
#Sugerencia: utiliza el atributo sheet_names para obtener una lista
#   con las hojas almacenadas.

#

print('Hojas en el libro: ', hojas)

df = None
#Pon tu código aquí.
#Sugerencia: utiliza el método parse del objeto ExcelFile para cargar
#   la hoja 'Datos'. Recuerda que las primeras 12 lineas no son necesarias.

#
if df is not None:
    print('\nPrimeras líneas de la hoja \'Datos\':\n', df.head())
```

```
Hojas en el libro:  []
```

Como se puede observar, al cargar una hoja, por defecto, el objeto DataFrame devuelto usa un índice entero de filas. Si sabemos que el índice de la hoja está en una columna (o columnas si se trata de índice jerárquico) podremos indicarlo con el argumento `index_col`.

Ejercicio 14: Se desea cargar la hoja de cálculo `'Datos'` almacenada en el archivo

`'datos_covid_andalucia.xls'`. Tras una inspección preliminar, se observa que las primeras 12 lineas no aportan información de interés. Además los datos utilizan un índice jerárquico almacenado en las columnas `'Fecha diagnóstico'` como primer nivel y `'Territorio'` como segundo nivel.

La salida esperada debe ser algo parecido a lo siguiente:

```
Primeras líneas de la hoja 'Datos':
                                Confirmados PDIA  Total confirmados  \
Fecha diagnóstico Territorio
06/05/2021        Andalucía               268                270
                  Almería                  12                 12
                  Cádiz                     9                  9
                  Córdoba                  17                 17
                  Granada                   7                  8

                                Hospitalizados  UCI  Fallecidos
Fecha diagnóstico Territorio
06/05/2021        Andalucía                3    0           0
                  Almería                  0    0           0
                  Cádiz                    1    0           0
                  Córdoba                  2    0           0
                  Granada                  0    0           0

Index dtypes:
 Fecha diagnóstico    object
Territorio           object
dtype: object
```

In [111…

```python
df = pd.DataFrame()

#Pon tu código aquí.
#Sugerencia: usa la función read_excel() usando path de acceso al fichero.
#Sugerencia: usa el parámetro skiprows para ignorar las primeras 12 lineas.
#Sugerencia: usa el parámetro index para indicar las columnas que forman el
#   multiindex.

#
if not df.empty:
    print('Primeras líneas de la hoja \'Datos\':\n', df.head())
    print('\nIndex dtypes:\n', df.index.dtypes)
```

Cuando alguna columna almacena información temporal (fechas) podremos indicarlo usando el parámetro `parse_dates=True`.

Ejercicio 14: Se desea cargar la hoja de cálculo `'Datos'` almacenada en el archivo `'datos_covid_andalucia.xls'`. Tras una inspección preliminar, se observa que las primeras 12 lineas no aportan información de interés. Además los datos utilizan un índice jerárquico almacenado en las columnas `'Fecha diagnóstico'` como primer nivel y `'Territorio'` como segundo nivel. Asegurar que el primer nivel del índice tiene tipo `datetime`.

La salida esperada debe ser algo parecido a lo siguiente:

```
Primeras líneas de la hoja 'Datos':
                                Confirmados PDIA  Total confirmados  \
Fecha diagnóstico Territorio
```

```
2021-06-05        Andalucía                 268                 270
                  Almería                    12                  12
                  Cádiz                       9                   9
                  Córdoba                    17                  17
                  Granada                     7                   8

                                    Hospitalizados  UCI  Fallecidos
Fecha diagnóstico Territorio
2021-06-05        Andalucía                      3    0           0
                  Almería                        0    0           0
                  Cádiz                          1    0           0
                  Córdoba                        2    0           0
                  Granada                        0    0           0

Index dtypes:
 Fecha diagnóstico    datetime64[ns]
Territorio                    object
dtype: object
```

In [112…

```
df = pd.DataFrame()

#Pon tu código aquí.
#Sugerencia: usa la función read_excel() usando path de acceso al fichero.
#Sugerencia: usa el parámetro skiprows para ignorar las primeras 12 lineas.
#Sugerencia: usa el parámetro index para indicar las columnas que forman el
#   multiindex.

#
if not df.empty:
    print('Primeras líneas de la hoja \'Datos\':\n', df.head())
    print('\nIndex dtypes:\n', df.index.dtypes)
```

Escribiendo datos Pandas con formato XLS.

Para escribir un objeto DataFrame usando formato XLS usaremos le método `to_excel()`. Podemos utilizar parámetro similares a to_csv() para gestionar la salida.

Ejercicio 15: Dado un dataframe 'df' se requiere almacenarlo en formato XLS usando el nombre 'Calificaciones' para la hoja creada. No almacenar el índice de filas.

La salida generada debería ser algo parecido a lo siguiente:

```
Hojas almacenadas:  ['Calificaciones']

Hoja 'Calificaciones':
          Apellidos     Nombre  Total      Fecha
0    Anguita Pérez  José Luis    5.3 2021-10-05
1    Anguita Pérez      María    7.2 2021-01-15
2  Fernández Muñoz    Antonia    6.2 2021-06-04
3     García Pérez       José    7.4 2021-09-25
```

In [113…

```
file = BytesIO()
df = pd.DataFrame({'Apellidos':['Anguita Pérez', 'Anguita Pérez',
                                'Fernández Muñoz', 'García Pérez'],
                   'Nombre':['José Luis', 'María', 'Antonia', 'José'],
                   'Total':[5.3,7.2,6.2,7.4],
                   'Fecha':pd.to_datetime(['2021-10-05','2021-01-15',
```

```
                                        '2021-06-04','2021-09-25'], dayfirst=True)})

#Guardar en file el dataframe.
#Pon tu código aquí
#Sugerencia: usa el método to_excel() para almacenar el dataframe.
#Utiliza el argumento 'sheet_name' para indicar el nombre de la hoja.
#Utiliza el argumento index para indicar que no se guarde el índice de filas.

#

if file.tell()>0:
    file.seek(0)#Simular reapertura del archivo.
    book = pd.ExcelFile(file)

hojas=[]
#Pon tu código aquí.
#Sugerencia: utiliza el atributo sheet_names para asignar a la variable
#hojas.

#
print('Hojas almacenadas: ', hojas)

df2 = pd.DataFrame() #donde cargar la hoja.

#Pon tu código aquí.
#Sugerencia: utiliza el objeto book como argumento para
#la función read_excel(). Indica el nombre de la hoja a cargar.

#
if not df2.empty:
    print("\nHoja 'Calificaciones':\n", df2.head())
```

```
Hojas almacenadas:  []
```