Trabajando con datos incompletos

Una de las características de trabajar con fuentes de datos en el mundo real es que proporcionan datos incompletos, es decir, no proporcionan un valor para algunos item de datos. Hay muchas situaciones que pueden dar lugar a este problema, por ejemplo:

- Si la fuente de datos es un sensor, este puede fallar durante un cierto tiempo. Los valores proporcionados durante ese periodo de tiempo no serán válidos.
- Otro ejemplo es la situación de un encuestador que por algún motivo no puede completar algún dato de una encuesta, por ejemplo porque no obtuvo respuesta por parte del entrevistado.
- También los valores perdidos o incompletos pueden surgir al procesar los datos, por ejemplo, al alinear ejes con distintas etiquetas de índice, o porque no se pueda calcular el resultado de una operación.

En este cuaderno vamos a estudiar distintas utilidades que ofrece el paquete Pandas para tratar con conjuntos de datos con datos incompletos.

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.

In [1]:
```
import pandas as pd
import numpy as np
np.set_printoptions(floatmode='fixed', precision=3)
print('Pandas versión: ', pd.__version__)
```

```
Pandas versión:  1.4.3
```

Mecanismo de valor centinela.

Pandas proporciona un mecanismo de valor centinela para indicar que un determinado item de dato tiene un valor no válido. Se utilizan dos valores centinelas:

- El valor `numpy.nan` para indicar un valor flotante no válido.
- El valor python `None` para indicar un valor no válido para el resto de tipos.

Como Pandas representa internamente sus datos con `numpy.ndarray`, el valor centinela `None` debe ser convertido al valor `numpy.nan` si es posible para permitir que las operaciones con los datos usen la eficiencia proporionada por numpy.

A continuación se muestra una tabla de conversiones usada por Pandas:

Tipo	Convertir al almacenar NAN	Valor centinela
float	No es necesario	np.nan
integer	Convertir a np.float64	np.nan
object	No es necesario	np.nan o None
boolean	Convertir a object	np.nan o None

Ejercicio 01: Se crea una serie con valores enteros y un valor perdido. ¿Qué dtype tendrá la serie? ¿Por qué?.

El resultado debería ser algo parecedio a lo siguiente:

```
El tipo de serie es: float64
```

In [2]:
```
s1 = pd.Series([-1, 2, None, 0])
print('El tipo de serie es: ', end='')
#Pon tu código aquí.
#Sugerencia: Imprime el atributo dtype del objeto s1.

#
```

```
El tipo de serie es:
```

Aunque los datos de entrada son enteros, al tener un valor `None`, según la regla de conversión vistas anteriormente, estos se convierten al tipo `np.float64` para representar el valor `None` como `np.nan`.

Sin embargo, al extender el ejemplo a un DataFrame la cosa cambia un poco.

Ejercicio 02: Se crea un Dataframe con valores enteros y varios valores perdidos. ¿cuáles serán los atributoa dtype que tendrán las columnas? ¿y el DataFrame? ¿Por qué?.

El resultado debería ser algo parecedio a lo siguiente:

```
A    float64
B    float64
C     object
dtype: object
```

In [3]:
```
df1 = pd.DataFrame([[1,  2, None],
                    [None, None, None],
                    [1,  3, None]], columns=list('ABC'))
#Pon tu código aquí.
#Sugerencia: imprime el atributo dtypes del objeto df1.

#
```

Como puedes observar, al crear el objeto DataFrame cada columna será un un objeto Series y se le aplica la regla de conversión por separado.

Como resumen, dado que `np.nan` es un valor Numpy y `None` un valor python (object), se recomienda usar `np.nan` siempre que sea posible para asegurarnos el mejor desempeño en las operaciones con los datos.

Manipulando los valores nulos.

Pandas ofrece varias funciones para manipular los valores nulos.

Generando máscaras para filtar los valores.

Ejercicio 03: Dada una serie se quiere obtener una máscara que indique qué valores son valores "nulos".

La salida obtenida debería parecerse a lo siguiente:

```
Los valores nulos son:
 0    False
1    False
2     True
3    False
dtype: bool
```

In [4]:
```
s1 = pd.Series([-1, 2, None, 0])
masc = None
#Pon tu código aquí.
#Sugerencia: Utiliza la función Series.isnull

#
print("Los valores nulos son:\n", masc)
```

```
Los valores nulos son:
 None
```

Ejercicio 04: Dado un dataframe se quiere obtener una máscara que indique qué valores son valores "nulos".

La salida obtenida debería parecerse a lo siguiente:

```
Los valores nulos son:
        A      B     C
0  False  False  True
1   True   True  True
2  False  False  True
```

In [5]:
```
df1 = pd.DataFrame([[1,  2, None],
                    [None, None, None],
                    [1,  3, None]], columns=list('ABC'))
masc = None
#Pon tu código aquí.
#Sugerencia: Utiliza la función DataFrame.isnull

#
print("Los valores nulos son:\n", masc)
```

```
Los valores nulos son:
 None
```

Ejercicio 05: Dado una serie se quiere filtrar los valores no nulos.

La salida obtenida debería parecerse a lo siguiente:

```
Los valores nulos son:
 0   -1.0
1    2.0
```

```
3    0.0
dtype: float64
```

In [6]:
```
s1 = pd.Series([-1, 2, None, 0])
s2 = np.zeros_like(s1.shape)
#Pon tu código aquí.
#Sugerencia: Utiliza la función Series.notnull para obtener una
#máscara y usa la máscara para indexar la serie.

#
print("Los valores no nulos son:\n", s2)
```

```
Los valores no nulos son:
 [0]
```

Para filtrar valores nulos en un objeto DataFrame se recomienda usar la función `DataFrame.dropna()`. Esta función tiene varios argumentos para ajustar su comportamiento de forma flexible. También podemos usar esta función en un objeto `Series.dropna()`.

Ejercicio 06: Dado un dataframe se filtrar los valores nulos. Se desea eliminar una fila del DataFrame, sólo cuando todos los valores de la fila sean nulos.

La salida obtenida debería parecerse a lo siguiente:

```
Los valores no nulos son:
     A    B     C
0  1.0  2.0  None
2  1.0  3.0  None
```

In [7]:
```
df1 = pd.DataFrame([[1,  2, None],
                    [None, None, None],
                    [1,  3, None]], columns=list('ABC'))
df2 = None
#Pon tu código aquí.
#Sugerencia: Utiliza la función DataFrame.dropna(), utiliza los
#   argumentos axis y how convenientemente.

#
print("Los valores no nulos son:\n", df2)
```

```
Los valores no nulos son:
 None
```

Ejercicio 07: Dado un dataframe se quiere filtrar los valores nulos. Se desea eliminar una columna del DataFrame, sólo cuando todos los valores de la columna sean nulos.

La salida obtenida debería parecerse a lo siguiente:

```
Los valores no nulos son:
     A    B
0  1.0  2.0
1  NaN  NaN
2  1.0  3.0
```

In [8]:
```
df1 = pd.DataFrame([[1,  2, None],
                    [None, None, None],
                    [1,  3, None]], columns=list('ABC'))
df2 = None
#Pon tu código aquí.
#Sugerencia: Utiliza la función DataFrame.dropna(), utiliza los
```

```
#    argumentos axis y how convenientemente.

#
print("Los valores no nulos son:\n", df2)
```

```
Los valores no nulos son:
 None
```

Rellenando los valores nulos con un valor dado.

Pandas ofrece la funciones `Series.fillna()` y `DataFrame.fillna()` para rellenar los valores nulos con un valor dado. Además existen variantes de esta función que permiten que se usen datos del propio objeto para rellenar.

Ejercicio 08: Dado un dataframe se quiere rellenar los valores nulos con el valor 0.0

La salida obtenida debería parecerse a lo siguiente:

```
   DateFrame original:
         A    B     C
   0  1.0  2.0  None
   1  NaN  NaN  None
   2  1.0  3.0  None

   DateFrame relleno:
         A    B    C
   0  1.0  2.0  0.0
   1  0.0  0.0  0.0
   2  1.0  3.0  0.0
```

In [9]:
```
df1 = pd.DataFrame([[1,  2, None],
                    [None, None, None],
                    [1,  3, None]], columns=list('ABC'))
df2 = np.zeros_like(df1.shape)
#Pon tu código aquí.
#Sugerencia: Utiliza la función fillna(), utiliza los
#argumentos axis y how convenientemente.

#
print("DateFrame original:\n", df1)
print("\nDateFrame relleno:\n", df2)
```

```
DateFrame original:
      A    B     C
0  1.0  2.0  None
1  NaN  NaN  None
2  1.0  3.0  None

DateFrame relleno:
 [0 0]
```

Ejercicio 09: Dado un dataframe se quiere rellenar los valores nulos con el valor 0.0. En este caso se desea modificar el dataframe original, es decir, sin devolver una copia modificada. Esta forma de procesar se denomina "*inplace*".

La salida obtenida debería parecerse a lo siguiente:

```
   DataFrame original:
         A    B     C
   0  1.0  2.0  None
```

```
   1  NaN  NaN  None
   2  1.0  3.0  None

   DateFrame modificado:
         A    B    C
   0  1.0  2.0  0.0
   1  0.0  0.0  0.0
   2  1.0  3.0  0.0
```

In [10]:
```
df1 = pd.DataFrame([[1,  2, None],
                    [None, None, None],
                    [1,  3, None]], columns=list('ABC'))
print('DataFrame original:\n', df1)
#Pon tu código aquí.
#Sugerencia: Utiliza la función fillna(), utiliza el
#   argumento inplace.

#
print("\nDateFrame modificado:\n", df1)
```

```
DataFrame original:
      A    B     C
0  1.0  2.0  None
1  NaN  NaN  None
2  1.0  3.0  None

DateFrame modificado:
      A    B     C
0  1.0  2.0  None
1  NaN  NaN  None
2  1.0  3.0  None
```

Ejercicio 10: Dado un dataframe se quiere rellenar los valores nulos con el último valor válido en su misma columna. La operación debe realizar *inplace*.

La salida obtenida debería parecerse a lo siguiente:

```
   DataFrame original:
         A    B    C
   0  1.0  2.0 -1.0
   1  NaN  NaN  NaN
   2  1.0  3.0  NaN

   DateFrame relleno:
         A    B    C
   0  1.0  2.0 -1.0
   1  1.0  2.0 -1.0
   2  1.0  3.0 -1.0
```

In [11]:
```
df1 = pd.DataFrame([[1,  2, -1],
                    [None, None, None],
                    [1,  3, None]], columns=list('ABC'))
print('DataFrame original:\n', df1)
#Pon tu código aquí.
#Sugerencia: Utiliza la función fillna() con el método 'ffill',
#   y el argumento inplace. También puedes utilizar la función
#   equivaliente ffill().

#
print("\nDateFrame relleno:\n", df1)
```

```
DataFrame original:
```

```
     A    B    C
0  1.0  2.0 -1.0
1  NaN  NaN  NaN
2  1.0  3.0  NaN

DateFrame relleno:
     A    B    C
0  1.0  2.0 -1.0
1  NaN  NaN  NaN
2  1.0  3.0  NaN
```

Ejercicio 11: Dada un dataframe se quiere rellenar los valores nulos con el siguiente valor válido en su misma columna. La operación debe modificar el dataframe.

La salida obtenida debería parecerse a lo siguiente:

```
DataFrame original:
     A    B    C
0  1.0  2.0  NaN
1  NaN  NaN  NaN
2  1.0  3.0 -1.0

DateFrame relleno:
     A    B    C
0  1.0  2.0 -1.0
1  1.0  3.0 -1.0
2  1.0  3.0 -1.0
```

In [12]:
```
df1 = pd.DataFrame([[1,  2, None],
                    [None, None, None],
                    [1,  3, -1]], columns=list('ABC'))
print('DataFrame original:\n', df1)
#Pon tu código aquí.
#Sugerencia: Utiliza la función fillna() con el método 'bfill',
#   y el argumento inplace. También puedes utilizar la función
#equivaliente bfill().

#
print("\nDateFrame relleno:\n", df1)
```

```
DataFrame original:
     A    B    C
0  1.0  2.0  NaN
1  NaN  NaN  NaN
2  1.0  3.0 -1.0

DateFrame relleno:
     A    B    C
0  1.0  2.0  NaN
1  NaN  NaN  NaN
2  1.0  3.0 -1.0
```

Otra alternativa para encontrar el valor de relleno es interpolarlo usando los valores cercanos. Para ello Pandas ofrece las funciones `Series.interpolate()` y `DataFrame.interpolate()`.

Ejercicio 12: Dada un dataframe se quiere rellenar los valores nulos interpolando por columnas de forma lineal. La operación debe modificar el dataframe.

La salida obtenida debería parecerse a lo siguiente:

```
Dataframe original:
          a         b         c
A  1.764052  0.400157  0.978738
B  2.240893       NaN -0.977278
C       NaN -0.151357       NaN
D  0.410599  0.144044  1.454274

DateFrame relleno:
          a         b         c
A  1.764052  0.400157  0.978738
B  2.240893  0.124400 -0.977278
C  1.325746 -0.151357  0.238498
D  0.410599  0.144044  1.454274
```

In [13]:

```python
np.random.seed(0)
df1 = pd.DataFrame(np.random.randn(12).reshape(4,3),
                   index=list('ABCD'),
                   columns=list('abc'))
#Forzamos algunos valores "perdidos"
df1.iloc[2,0]=np.nan
df1.iloc[1,1]=np.nan
df1.iloc[2,2]=np.nan
print("Dataframe original:\n", df1)
#Pon tu código aquí.
#Sugerencia: Utiliza la función interpolate().

#
print("\nDateFrame relleno:\n", df1)
```

```
Dataframe original:
          a         b         c
A  1.764052  0.400157  0.978738
B  2.240893       NaN -0.977278
C       NaN -0.151357       NaN
D  0.410599  0.144044  1.454274

DateFrame relleno:
          a         b         c
A  1.764052  0.400157  0.978738
B  2.240893       NaN -0.977278
C       NaN -0.151357       NaN
D  0.410599  0.144044  1.454274
```

In [ ]: