Manejando fechas y horas

Pandas ofrece un conjunto de utilidades para manejar datos temporales. El ejemplo más claro es una serie temporal (el índice es temporal) como por ejemplo la cotización de una empresa o la evolución de los casos COVID durante un perido de tiempo.

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`, el paquete `datetime` con el alias `dt` y el paquete `locale` para indicar que usaremos el lenguaje castellano para indicar las magnitudes y fechas. Esto es un dato importante ya que la información temporal se muestra (o se da) en un formato que depende de la localización utilizada.

In [53]:
```
import pandas as pd
import numpy as np
np.set_printoptions(floatmode='fixed', precision=3)
print('Pandas versión: ', pd.__version__)
import datetime as dt
import locale
locale.setlocale(locale.LC_ALL, 'es_ES.UTF-8')
```

Out[53]:
```
Pandas versión:  1.4.3
'es_ES.UTF-8'
```

Cómo describir un dato temporal.

Utilizaremos el término `date` (tiempo) para indicar tanto una fecha de calendario, por ejemplo `1 de junio de 2020`, como un hora, por ejemplo `23:30:00`, o ambas cosas de forma conjunta, por ejemplo `1 de junio de 2020, 23:30:00`.

Se distinguen tres formas de indicar el tiempo:

- Indicar una momento específico o marca de tiempo (`TimeStamp`), por ejemplo las `11:00h del 16 junio de 2020`.
- Indicar un intervalo temporal (`DateTimeIndex`) entre dos marcas de tiempo, por ejemplo el mes de junio de 2020 que sería el intervalo entre las marcas de tiempo: desde `1 de junio de 2020, 00:00:00` hasta `30 de junio de 2020, 23:59:59`.
- Indicar un incremento temporal o duración (`TimeDelta`), por ejemplo, 20 minutos.

Pandas proporciona sus propias funciones para gestionar estas tres formas distintas de indicar un dato temporal. Estas funciones se basan en las correspondientes ofrecidas en el módulo `numpy.datetime64`.

En muchas ocasiones deberemos crear un objeto temporal a partir de una representación textual con `pandas.to_datetime()` o al revés con `Timestamp.strftime()`. Pandas utilizan esta notación para indicar el formato de tiempo deseado.

Especificar una marca de tiempo.

Ejercicio 01: Generar una marca de tiempo para la fecha "12 de junio de 2021" dada como `'12-06-2021'`.

El resultado mostrado debería ser algo parecido a:

```
Fecha:  2021-12-06 00:00:00
```

In [54]:
```
date = '12-06-2021'
dt=None
#Pon tu código aquí.
#Sugerencia: utiliza la función pd.to_datetime(). Ajusta el
#   parámetro format convenientemente.

#

print('Fecha: ', dt)
```

```
Fecha:  None
```

Ejercicio 02: Generar una marca de tiempo para la fecha '12 de junio de 2021, 23:00'.

El resultado mostrado debería ser algo parecido a:

```
Fecha:  2021-06-12 23:30:00
```

In [55]:
```
date = '12 de junio de 2021, 23:30'
dt=None
#Pon tu código aquí.
#Sugerencia: utiliza la función pd.to_datetime(). Ajusta el
#parámetro format convenientemente.

#

print('Fecha: ', dt)
```

```
Fecha:  None
```

Para convertir un objeto TimeStamp en su representación textual podemos utilizar el método `strftime()`.

Ejercicio 03: Data un objeto TimeStamp queremos obtener su representación textual con formato: '12 de junio del 2021, 23:30'

El resultado mostrado debería ser algo parecido a:

```
Fecha: lunes 01 de febrero del 2021, 13:30
```

In [56]:
```
dt = pd.to_datetime('2021-02-01 13:30')

texto=""
#Pon tu código aquí.
#Sugerencia: usa el método strftime() y ajusta el parámetro format
#   convenientemente.

#
print('Fecha: {}'.format(texto))
```

```
Fecha:
```

Especificar un intervalo temporal.

Pandas utiliza el tipo `DateTimeIndex` para representar intervalos temporales. La función `date_range()` permite generar los intervalos temporales de distintas formas, por ejemplo, dadas dos fechas, o dada una fecha y un número de periodos con una frecuencia dada.

Ejercicio 04: Dados dos fechas, obtener el intervalo temporal entre las mismas con un frecuencia de 1 día.

El resultado mostrado debería ser algo parecido a:

```
Intervalo:
 DatetimeIndex(['2021-01-01', '2021-01-02', '2021-01-03', '2021-01-04',
                '2021-01-05', '2021-01-06', '2021-01-07', '2021-01-08',
                '2021-01-09', '2021-01-10', '2021-01-11', '2021-01-12',
                '2021-01-13', '2021-01-14', '2021-01-15', '2021-01-16',
                '2021-01-17', '2021-01-18', '2021-01-19', '2021-01-20',
                '2021-01-21', '2021-01-22', '2021-01-23', '2021-01-24',
                '2021-01-25', '2021-01-26', '2021-01-27', '2021-01-28',
                '2021-01-29', '2021-01-30', '2021-01-31'],
               dtype='datetime64[ns]', freq='D')
```

In [57]:
```
dt1 = pd.to_datetime('2021-01-01')
dt2 = pd.to_datetime('2021-01-31')

intervalo=None
#Pon tu código aquí.
#Sugerencia: usa el función date_range().

#
print('Intervalo:\n', intervalo)
```

```
Intervalo:
 None
```

Ejercicio 05: Dados dos fechas, obtener el intervalo temporal entre las mismas con un frecuencia de 1 hora.

El resultado mostrado debería ser algo parecido a:

```
Intervalo:
 DatetimeIndex(['2021-01-01 00:00:00', '2021-01-01 01:00:00',
                '2021-01-01 02:00:00', '2021-01-01 03:00:00',
                '2021-01-01 04:00:00', '2021-01-01 05:00:00',
                '2021-01-01 06:00:00', '2021-01-01 07:00:00',
                '2021-01-01 08:00:00', '2021-01-01 09:00:00',
                '2021-01-01 10:00:00', '2021-01-01 11:00:00',
                '2021-01-01 12:00:00', '2021-01-01 13:00:00',
```

```
                '2021-01-01 14:00:00', '2021-01-01 15:00:00',
                '2021-01-01 16:00:00', '2021-01-01 17:00:00',
                '2021-01-01 18:00:00', '2021-01-01 19:00:00',
                '2021-01-01 20:00:00', '2021-01-01 21:00:00',
                '2021-01-01 22:00:00', '2021-01-01 23:00:00',
                '2021-01-02 00:00:00'],
               dtype='datetime64[ns]', freq='H')
```

In [58]:
```
dt1 = pd.to_datetime('2021-01-01')
dt2 = pd.to_datetime('2021-01-02')

intervalo=None
#Pon tu código aquí.
#Sugerencia: usa el función date_range(). Utiliza el argumento
#   freq convenientemente.

#
print('Intervalo:\n', intervalo)
```

```
Intervalo:
 None
```

Ejercicio 06: Dada una fecha, obtener el intervalo temporal de 30 segundos con frecuencia de 1 segundo a partir de la misma.

El resultado mostrado debería ser algo parecido a:

```
Intervalo:
 DatetimeIndex(['2021-01-01 00:00:00', '2021-01-01 00:00:01',
                '2021-01-01 00:00:02', '2021-01-01 00:00:03',
                '2021-01-01 00:00:04', '2021-01-01 00:00:05',
                '2021-01-01 00:00:06', '2021-01-01 00:00:07',
                '2021-01-01 00:00:08', '2021-01-01 00:00:09',
                '2021-01-01 00:00:10', '2021-01-01 00:00:11',
                '2021-01-01 00:00:12', '2021-01-01 00:00:13',
                '2021-01-01 00:00:14', '2021-01-01 00:00:15',
                '2021-01-01 00:00:16', '2021-01-01 00:00:17',
                '2021-01-01 00:00:18', '2021-01-01 00:00:19',
                '2021-01-01 00:00:20', '2021-01-01 00:00:21',
                '2021-01-01 00:00:22', '2021-01-01 00:00:23',
                '2021-01-01 00:00:24', '2021-01-01 00:00:25',
                '2021-01-01 00:00:26', '2021-01-01 00:00:27',
                '2021-01-01 00:00:28', '2021-01-01 00:00:29'],
               dtype='datetime64[ns]', freq='S')
```

In [59]:
```
dt1 = pd.to_datetime('2021-01-01')

intervalo=None
#Pon tu código aquí.
#Sugerencia: usa el función date_range(). Utiliza los argumentos periods y freq
#   convenientemente.

#
print('Intervalo:\n', intervalo)
```

```
Intervalo:
 None
```

Especificando un incremento temporal.

Pandas utiliza el tipo `TimeDelta` para representar un incremento temporal. La función `to_timedelta()` permite crear objetos TimeDelta.

Ejercicio 07: Crear un incremento temporal que represente un 1 segundo y obtener la correspondiente fecha incrementada con dicho incremento a partir de la fecha '01-01-2021'.

El resultado mostrado debería ser algo parecido a:

```
2021-01-01 00:00:00 + 0 days 00:00:01 = 2021-01-01 00:00:01
```

In [60]:
```
dt1 = pd.to_datetime('2021-01-01')
delta = 0
dt2 = dt1
#Pon tu código aquí.
#Sugerencia: crea el objeto TimeDelta usando to_timedelta()
#   indicado como unidad "segundos".
#Suma el objeto TimeDelta a la fecha dada para obtener la fecha incrementada.

#
print('{} + {} = {}'.format(dt1, delta, dt2))
```

```
2021-01-01 00:00:00 + 0 = 2021-01-01 00:00:00
```

Ejercicio 08: Crear un intervalo temporal de 12 días a partir de una fecha. Para ello crearemos un array de incrementos temporales de 0 día, 1 días, ... 11 días y sumaremos la fecha origen a dicho array para obtener el intervalo temporal.

El resultado mostrado debería ser algo parecido a:

```
Deltas:
 TimedeltaIndex([ '0 days',  '1 days',  '2 days',  '3 days',  '4 days',
                  '5 days',  '6 days',  '7 days',  '8 days',  '9 days',
                 '10 days', '11 days'],
                dtype='timedelta64[ns]', freq=None)

Intervalo:
 DatetimeIndex(['2021-01-01', '2021-01-02', '2021-01-03', '2021-01-04',
                '2021-01-05', '2021-01-06', '2021-01-07', '2021-01-08',
                '2021-01-09', '2021-01-10', '2021-01-11', '2021-01-12'],
               dtype='datetime64[ns]', freq=None)
```

In [61]:
```
dt1 = pd.to_datetime('2021-01-01')
deltas = []
intervalo = []
#Pon tu código aquí.
#Sugerencia: usa la función to_timedelta() a partir del
#   rango numérico [0,12)
#Suma al array resultante, la fecha origen.

#
print('Deltas:\n', deltas)
print('\nIntervalo:\n', intervalo)
```

```
Deltas:
 []

Intervalo:
 []
```

Ejercicio 09: Dadas dos fechas, queremos calcular el incremento temporal entre ambas.

El resultado mostrado debería ser algo parecido a:

```
2021-01-02 00:00:00 - 2021-01-01 00:00:00 = 1 days 00:00:00
```

In [62]:
```
dt1 = pd.to_datetime('2021-01-01')
dt2 = pd.to_datetime('2021-01-02')
delta = 0

#Pon tu código aquí.
#Sugerencia: Utiliza el operador '-'

#
print('{} - {} = {}'.format(dt2, dt1, delta))
```

```
2021-01-02 00:00:00 - 2021-01-01 00:00:00 = 0
```

Series temporales.

Una serie que utiliza un rango temporal como índice se denomina "serie temporal". Pandas ofrece un grupo de funciones para manipular series temporales.

Ejercicio 10: Dadas un conjunto de 31 muestras tomadas durante un més con frecuencia de 1 día a partir de la fecha 01-01-2021, se requiere crear la correspondiente serie temporal.

El resultado mostrado debería ser algo parecido a:

```
Serie temporal:
 2021-01-01    0.636962
2021-01-02    0.269787
2021-01-03    0.040974
2021-01-04    0.016528
2021-01-05    0.813270
2021-01-06    0.912756
2021-01-07    0.606636
2021-01-08    0.729497
2021-01-09    0.543625
2021-01-10    0.935072
2021-01-11    0.815854
2021-01-12    0.002739
2021-01-13    0.857404
2021-01-14    0.033586
2021-01-15    0.729655
2021-01-16    0.175656
2021-01-17    0.863179
2021-01-18    0.541461
2021-01-19    0.299712
2021-01-20    0.422687
2021-01-21    0.028320
2021-01-22    0.124283
2021-01-23    0.670624
2021-01-24    0.647190
2021-01-25    0.615385
2021-01-26    0.383678
2021-01-27    0.997210
2021-01-28    0.980835
2021-01-29    0.685542
2021-01-30    0.650459
```

```
2021-01-31    0.688447
Freq: D, dtype: float64
```

In [63]:
```
gen=np.random.default_rng(0)
data = gen.random(31)

idx = None #Índice temporal de la serie.

#Pon tu código aquí.
#Sugerencia: usa la función time_range() para crear el intervalo indicado.

#

s = None #Serie temporal.
#Pon tu código aquí.
#Sugerencia: crea la serie temporal, indicando que se use como
#índice el rango temporal creado.

#

print('Serie temporal:\n', s)
```

```
Serie temporal:
 None
```

Indexación de un serie temporal.

Para indexar una serie temporal podemos utilizar como ya sabemos varias alternativas:

- Usar el operador `[]`, por ejemplo `s[0]` para obtener el primer valor o `s[dt]` para obtener el dato asociado objeto Timestamp `dt`. Recuerda que esta forma es menos eficiente que las siguientes.
- Usar índices de posición enteros con el atributo `.iloc[]`, por ejemplo, `s.iloc[0]` nos dará el primer valor de la series, o `s.iloc[0:5]` nos daría las primeras 4 muestras.
- Usar etiquetas de índice con el atributo `.loc[]`, por ejemplo `s.loc[dt1]` nos dará el dato para la fecha `dt1`, o `s.loc[dt1:dt2]` para obtener un rango temporal, siendo `dt1` y `dt2` dos objetos `Timestamp`.
- Alternativamente, podemos acceder con fechas aproximadas, con la función `Series.asof()`, por ejemplo, `s.asof(dt)` nos dará el último valor distinto de 'NaN' cuyo Timestamp es anterior o igual a la fecha `dt`.

Ejercicio 11: Dada una serie temporal con un conjunto de 31 muestras tomadas durante un més con frecuencia de 1 día a partir de la fecha 01-01-2021, se requiere obtener las muestras correspondientes entre las fechas "10-01-2021" a "15-01-2021".

El resultado mostrado debería ser algo parecido a:

```
Muestra:
 2021-01-10    0.935072
2021-01-11    0.815854
2021-01-12    0.002739
2021-01-13    0.857404
2021-01-14    0.033586
2021-01-15    0.729655
Freq: D, dtype: float64
```

In [64]:
```
gen=np.random.default_rng(0)
s = pd.Series(gen.random(31), index=pd.date_range('2021-01-01', periods=31))
```

```
dt1 = 0
#Pon tu código aquí.
#Sugerencia: utiliza el método to_datetime().

#

dt2 = s.shape[0]
#Pon tu código aquí.
#Sugerencia: utiliza el método to_datetime().

#

muestra = None
#Pon tu código aquí.
#Sugerencia: usa el operador .loc[] para obtener la subserie.

#

print('Muestra:\n', muestra)
```

```
Muestra:
 None
```

Ejercicio 11b: Una sonda genera muestras cada segundo, pero debido al mecanismo de transmisión es posible que algunas se pierdan y no sean almacenadas. Se requiere obtener la muestra anterior más próxima a la fecha '15/1/2021'.

El resultado mostrado debería ser algo parecido a:

```
Serie temporal:
 2021-01-01 01:29:43    0.028320
2021-01-01 02:38:43    0.857404
2021-01-01 05:37:59    0.685542
2021-01-01 12:08:38    0.863179
2021-01-02 11:31:05    0.647190
2021-01-03 12:10:39    0.124283
2021-01-05 08:48:02    0.912756
2021-01-05 13:27:29    0.040974
2021-01-06 04:34:23    0.269787
2021-01-06 18:17:21    0.729497
2021-01-07 06:05:48    0.002739
2021-01-07 13:00:30    0.422687
2021-01-09 08:24:42    0.175656
2021-01-09 22:09:57    0.688447
2021-01-10 06:17:37    0.299712
2021-01-11 06:14:54    0.636962
2021-01-12 02:13:31    0.980835
2021-01-13 21:55:12    0.670624
2021-01-15 01:22:39    0.033586
2021-01-15 18:31:58    0.383678
2021-01-16 11:05:16    0.935072
2021-01-16 12:07:07    0.650459
2021-01-17 05:58:30    0.997210
2021-01-21 17:20:43    0.813270
2021-01-22 07:07:27    0.729655
2021-01-23 13:18:31    0.615385
2021-01-24 10:32:06    0.016528
2021-01-25 22:46:18    0.815854
2021-01-27 01:02:27    0.541461
```

```
2021-01-29 02:33:59    0.606636
2021-01-31 23:30:23    0.543625
dtype: float64

Muestra para la fecha 2021-01-15 00:00:00: 0.6706244146936303
```

In [65]:

```python
gen=np.random.default_rng(0)
t = pd.Series(pd.date_range('2021-01-01', '2021-02-1', freq='S')).sample(31)
s = pd.Series(gen.random(31), index=t).sort_index()

print('Serie temporal:\n', s)

dt = 0
#Pon tu código aquí.
#Sugerencia: utiliza el método to_datetime() para indicar la fecha '15/1/2021'

#

muestra = None
#Pon tu código aquí.
#Sugerencia: usa el operador .iloc[] para obtener la subserie.

#

print(f'\nMuestra para la fecha {dt}: {muestra}')
```

```
Serie temporal:
 2021-01-02 18:41:56    0.863179
2021-01-03 15:27:34    0.729655
2021-01-04 15:12:36    0.935072
2021-01-04 15:30:56    0.543625
2021-01-05 09:43:37    0.541461
2021-01-06 17:50:56    0.815854
2021-01-07 12:03:14    0.033586
2021-01-10 22:25:06    0.729497
2021-01-11 15:28:43    0.912756
2021-01-12 05:04:59    0.670624
2021-01-13 20:03:50    0.997210
2021-01-15 00:53:27    0.857404
2021-01-15 12:52:35    0.016528
2021-01-16 20:43:25    0.647190
2021-01-16 23:40:36    0.688447
2021-01-19 01:51:09    0.980835
2021-01-19 16:38:56    0.636962
2021-01-19 18:30:11    0.383678
2021-01-20 03:21:26    0.269787
2021-01-21 02:19:28    0.040974
2021-01-22 10:34:20    0.650459
2021-01-22 15:37:30    0.615385
2021-01-24 05:30:21    0.422687
2021-01-25 22:56:27    0.606636
2021-01-26 23:24:43    0.028320
2021-01-27 05:02:27    0.813270
2021-01-27 14:29:27    0.685542
2021-01-30 01:55:04    0.175656
2021-01-31 07:50:45    0.002739
2021-01-31 12:26:59    0.124283
2021-01-31 23:05:03    0.299712
dtype: float64

Muestra para la fecha 0: None
```

Muestreo de datos temporales

Pandas ofrece la posiblidad de dada un serie temporal obtener una versión como un remuestreo de la original usando la función `resample()`. Esta función permite realizar tanto submuestreos como sobremuestreos.

Sub muestreo

Ejercicio 12: Dada una serie temporal con un conjunto de muestras tomadas durante un año con frecuencia de 1 día a partir de la fecha 01-01-2020, se requiere obtener un submuestreo cada 15 días, utilizando el primer valor de cada intervalo de 15 días como valor submuestreado.

El resultado mostrado debería ser algo parecido a:

```
Sub muestreo:
 2020-01-01    0.636962
2020-01-16    0.175656
2020-01-31    0.688447
2020-02-15    0.890274
2020-03-01    0.404552
2020-03-16    0.425229
2020-03-31    0.927424
2020-04-15    0.538934
2020-04-30    0.344310
2020-05-15    0.412896
2020-05-30    0.009955
2020-06-14    0.308857
2020-06-29    0.860701
2020-07-14    0.571396
2020-07-29    0.177227
2020-08-13    0.634070
2020-08-28    0.182943
2020-09-12    0.422628
2020-09-27    0.744381
2020-10-12    0.701569
2020-10-27    0.896801
2020-11-11    0.127358
2020-11-26    0.943678
2020-12-11    0.073716
2020-12-26    0.821792
Freq: 15D, dtype: float64
```

In [66]:
```
gen=np.random.default_rng(0)
s = pd.Series(gen.random(366), index=pd.date_range('2020-01-01', periods=366))

sub_s = None
#Pon tu código aquí.
#Sugerencia: utiliza el método resample() para indicar el submuestreo y aplica
#   el método first() al resultado.

#

print('Sub muestreo:\n', sub_s)
```

```
Sub muestreo:
 None
```

Ejercicio 13: Dada una serie temporal con un conjunto de muestras tomadas durante un año con frecuencia de 1 día a partir de la fecha 01-01-2020, se requiere obtener un submuestreo cada 15 días,

utilizando el valor promedio de cada intervalo de 15 días como valor submuestreado.

El resultado mostrado debería ser algo parecido a:

```
Sub muestreo:
 2020-01-01    0.529623
2020-01-16    0.539081
2020-01-31    0.510263
2020-02-15    0.442301
2020-03-01    0.487213
2020-03-16    0.643738
2020-03-31    0.683314
2020-04-15    0.482472
2020-04-30    0.475276
2020-05-15    0.583582
2020-05-30    0.475277
2020-06-14    0.644293
2020-06-29    0.504056
2020-07-14    0.516807
2020-07-29    0.461117
2020-08-13    0.527248
2020-08-28    0.618780
2020-09-12    0.457935
2020-09-27    0.731625
2020-10-12    0.491455
2020-10-27    0.577542
2020-11-11    0.452530
2020-11-26    0.503434
2020-12-11    0.485966
2020-12-26    0.371677
Freq: 15D, dtype: float64
```

In [67]:
```
gen=np.random.default_rng(0)
s = pd.Series(gen.random(366), index=pd.date_range('2020-01-01', periods=366))

sub_s = None
#Pon tu código aquí.
#Sugerencia: utiliza el método resample() para indicar el submuestreo y aplica
#el método mean() al resultado.

#

print('Sub muestreo:\n', sub_s)
```

```
Sub muestreo:
 None
```

Sobre muestreo

Ejercicio 14: Dada una serie temporal con un conjunto de muestras tomadas durante un 30 segundos con frecuencia de 1 segundo, a partir de la fecha 01-01-2020, se requiere obtener un sobre muestreo cada 0.5 segundos, utilizando el valor válido anterior para rellenar los valores estimados.

El resultado mostrado debería ser algo parecido a:

```
Sobre muestreo:
 2021-01-01 00:00:00.000    0.636962
2021-01-01 00:00:00.500    0.636962
```

```
2021-01-01 00:00:01.000    0.269787
2021-01-01 00:00:01.500    0.269787
2021-01-01 00:00:02.000    0.040974
2021-01-01 00:00:02.500    0.040974
2021-01-01 00:00:03.000    0.016528
2021-01-01 00:00:03.500    0.016528
2021-01-01 00:00:04.000    0.813270
Freq: 500L, dtype: float64
```

In [68]:

```
gen=np.random.default_rng(0)
s = pd.Series(gen.random(5), index=pd.date_range('2021', periods=5, freq='S'))

sob_s = None
#Pon tu código aquí.
#Sugerencia: utiliza el método resample() para indicar el sobre muestreo y aplica
#el método ffill() al resultado.

#
print('Serie original:\n', s)
print('\nSobre muestreo:\n', sob_s)
```

```
Serie original:
 2021-01-01 00:00:00    0.636962
2021-01-01 00:00:01    0.269787
2021-01-01 00:00:02    0.040974
2021-01-01 00:00:03    0.016528
2021-01-01 00:00:04    0.813270
Freq: S, dtype: float64

Sobre muestreo:
 None
```

Agregar datos usando una ventana deslizante

Pandas proporciona la función `rolling()` para obtener datos agregados usando un mecanismo de ventana deslizante.

Ejercicio 15: Dada una serie temporal con un conjunto de muestras tomadas durante un año con frecuencia de 1 día se requiere obtener una versión con datos obtenidos como la media en la ventana temporal que incluye a los 14 días anteriores.

El resultado mostrado debería ser algo parecido a:

```
Serie original:
 2020-01-01    0.636962
2020-01-02    0.269787
2020-01-03    0.040974
2020-01-04    0.016528
2020-01-05    0.813270
Freq: D, dtype: float64

Medias últimos 15 días:
 2020-01-01    0.636962
2020-01-02    0.453374
2020-01-03    0.315907
2020-01-04    0.241062
2020-01-05    0.355504
Freq: D, dtype: float64
```

In [69]:
```
gen=np.random.default_rng(0)
s = pd.Series(gen.random(366),
              index=pd.date_range('2020-01-01', periods=366))

s2 = s.copy()
#Pon tu código aquí.
#Sugerencia: utiliza el método rolling() para realizar
#   el cálculo por ventanas deslizantes con un periodo
#   de 15 días y aplica el método mean() al resultado.

#

print('Serie original:\n', s.head())
print('\nMedias últimos 15 días:\n', s2.head())
```

```
Serie original:
 2020-01-01    0.636962
2020-01-02    0.269787
2020-01-03    0.040974
2020-01-04    0.016528
2020-01-05    0.813270
Freq: D, dtype: float64

Medias últimos 15 días:
 2020-01-01    0.636962
2020-01-02    0.269787
2020-01-03    0.040974
2020-01-04    0.016528
2020-01-05    0.813270
Freq: D, dtype: float64
```

Ejercicio 16: Una sonda envía información con una frecuencia diaria. Se sabe que las mediciones están afectadas por ruido "blanco" y para atenuarlo se requiere calcular la media en ventanas de tres muestras centradas.

El resultado mostrado debería ser algo parecido a:

```
Serie original:
 2020-01-01    0.636962
2020-01-02    0.269787
2020-01-03    0.040974
2020-01-04    0.016528
2020-01-05    0.813270
Freq: D, dtype: float64

Medias de tres días centradas:
 2020-01-01    0.453374
2020-01-02    0.315907
2020-01-03    0.109096
2020-01-04    0.290257
2020-01-05    0.580851
Freq: D, dtype: float64
```

In [70]:
```
gen=np.random.default_rng(0)
s = pd.Series(gen.random(366), index=pd.date_range('2020-01-01', periods=366))

s2 = s.copy()
#Pon tu código aquí.
#Sugerencia: utiliza el método rolling() para realizar
#   el cálculo por ventanas deslizantes con un periodo
```

```
#   de 3 días.
#Sugerencia: Utiliza el parámetro center para forzar
#   que las ventanas se centren en la muestra a procesar.

#
print('Serie original:\n', s.head())
print('\nMedias de tres días centradas:\n', s2.head())
```

```
Serie original:
 2020-01-01    0.636962
2020-01-02    0.269787
2020-01-03    0.040974
2020-01-04    0.016528
2020-01-05    0.813270
Freq: D, dtype: float64

Medias de tres días centradas:
 2020-01-01    0.636962
2020-01-02    0.269787
2020-01-03    0.040974
2020-01-04    0.016528
2020-01-05    0.813270
Freq: D, dtype: float64
```

In [ ]: