Resumen del capítulo: Análisis de series temporales

Series temporales

Las **series temporales** son las secuencias de números a lo largo del eje del tiempo. El intervalo entre los valores de la serie es constante.

En pandas, el trabajo adecuado con fecha y hora requiere cambiar el tipo de datos en la columna de `object` a `datetime64`. Se puede hacer al leer, usando el argumento `parse_dates`. También tenemos que establecer el índice del DataFrame especificando las columnas deseadas en el argumento `index_col`:

```
# valores de parse_dates = lista de números de columna o nombres de columna
data = pd.read_csv('filename.csv', index_col=[0], parse_dates=[0])
```

Para verificar si las fechas y horas están en orden cronológico, observa el atributo `is_monotonic` del índice de la tabla. Si el orden es cronológico, el atributo devolverá `True`; si no, `False`.

```
print(data.index.is_monotonic)
```

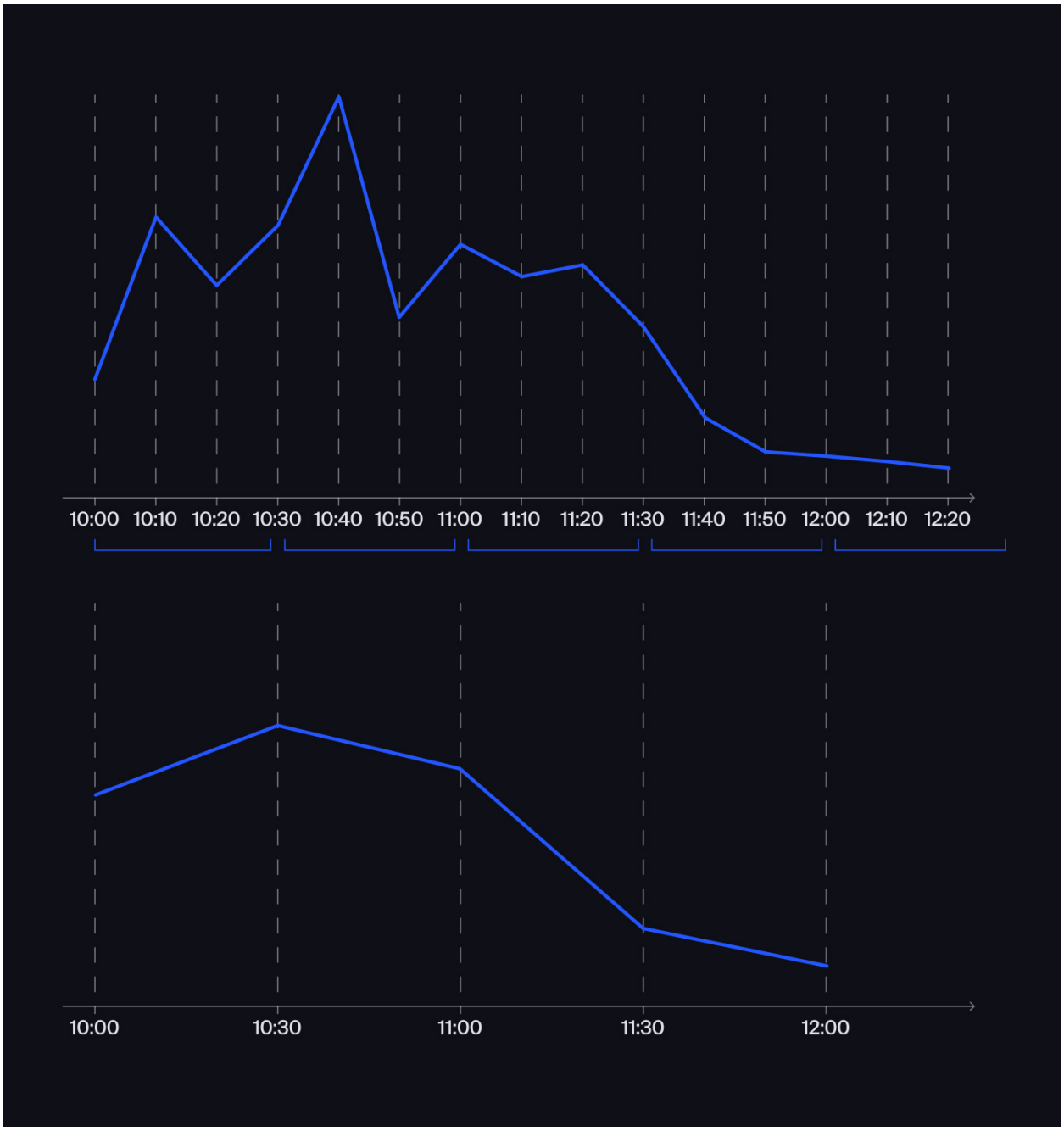
Podemos especificar fecha y hora como índice y seleccionar los datos por fecha:

```
data = data['2016':'2017']
data = data['2016-01':'2016-06']
data = data['2016-01-01':'2016-01-10']
```

Remuestreo

Remuestrear significa cambiar el intervalo con los valores de la serie. Se realiza en dos pasos:

1. Elige la nueva duración del intervalo. Considera que los valores del intervalo existente están agrupados.
2. En cada grupo se calcula el valor acumulado de la serie. Puede ser mediana, media, máximo o mínimo.



Para cambiar el intervalo y agrupar los valores, llama a la función `resample()`. Especifica el nuevo intervalo en el argumento. Aquí tienes un ejemplo:

```
# 1H = una hora
data.resample('1H')
```

```
# 2W = dos semanas
data.resample('2W')
```

La función `resample()` es similar a la función `groupby()`. Después de agrupar, llama a las funciones `mean()` y `max()` para agregar los valores:

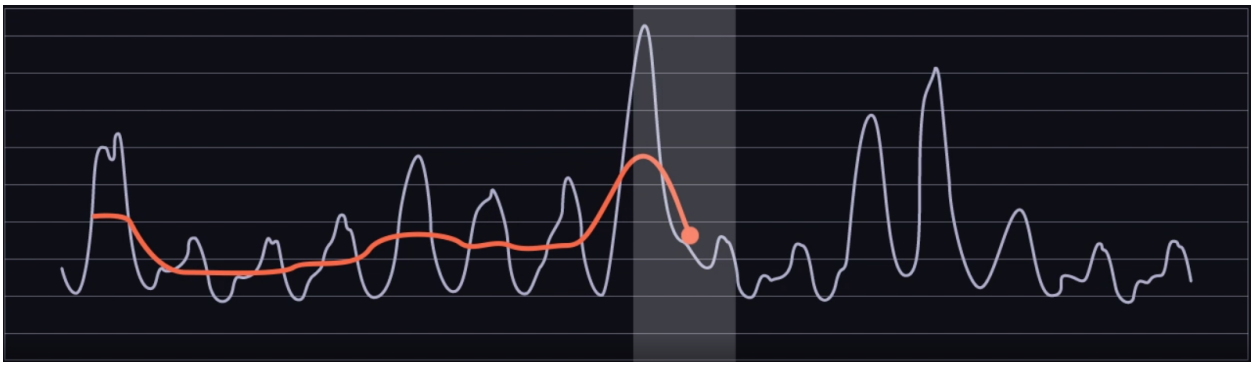
```
# media para cada hora
data.resample('1H').mean()

# máximo por cada dos semanas
data.resample('2W').max()
```

Media móvil

La **media móvil** o **promedio móvil** es un método para suavizar los datos en una serie temporal. El método consiste en encontrar los valores menos susceptibles a fluctuaciones, es decir, la media aritmética.

Así es como funciona el método: el intervalo para promediar (**tamaño de ventana**) se selecciona de forma experimental. Cuanto mayor sea el intervalo, más fuerte será el suavizado. Luego, la ventana comienza a desplazarse casi desde el principio hasta el final de la serie temporal. En cada punto se calcula el valor medio.

En el promedio móvil, las ventanas se superponen y no pueden ir más allá de la serie. Esto significa que el número de medias obtenidas será ligeramente menor que el número de valores iniciales de la serie.



En pandas, la media móvil se calcula en dos pasos:

1. Llama a la función `rolling()` para crear una ventana móvil. Especifica el tamaño de la ventana en el argumento:

```
# tamaño de la ventana 7
data.rolling(7)
```

2. Llama a la función `mean()` para agregar los valores:

```
# media móvil con tamaño de ventana 7
data.rolling(7).mean()
```

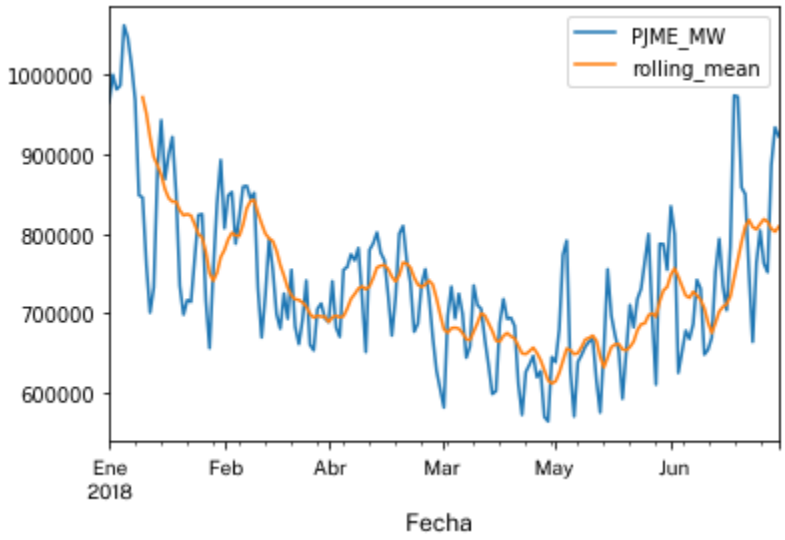
Tendencias y estacionalidad

Una **tendencia** es un cambio ligero del valor medio de la serie sin repetir patrones. Por ejemplo, el incremento anual en la venta de boletos de avión.

Estacionalidad significa patrones que se repiten de forma cíclica en una serie temporal. Por ejemplo, el crecimiento de las ventas de boletos de avión cada verano.

Las tendencias y la estacionalidad dependen de la escala de los datos. No puedes ver patrones que se repiten todos los veranos si solo hay datos de un año.

Observa el gráfico `rolling_mean`. El aumento del consumo eléctrico en invierno y en verano es una tendencia.



Si se analizan estos datos en una escala de varios años, el aumento en el invierno y en el verano serían cambios estacionales.

El módulo **tsa.seasonal** (tsa significa análisis de series temporales en inglés) de la librería **statsmodels** contiene la función `seasonal_decompose()`. Esta función divide la serie en tres componentes: tendencia, estacionalidad y residuos. El componente residual no puede explicarse por la tendencia y la estacionalidad, y es esencialmente solo ruido.

```
from statsmodels.tsa.seasonal import seasonal_decompose

decomposed = seasonal_decompose(data)
```

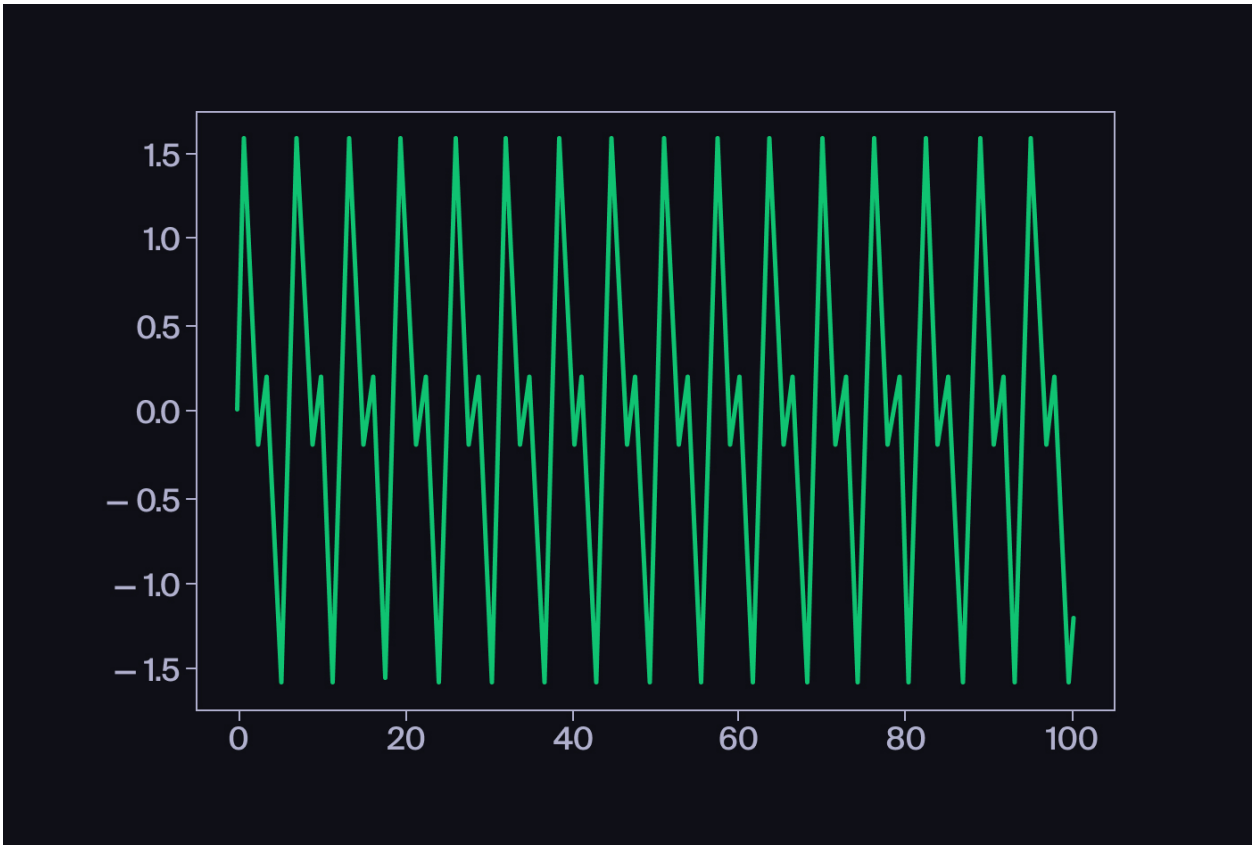
`season_decompose()` toma una serie temporal y devuelve una instancia de la clase `DecomposeResult`. Contiene los atributos requeridos:

- `decomposed.trend`: tendencia
- `decomposed.seasonal`: componente estacional
- `decomposed.resid`: residuos

Series estacionarias

En estadística, la serie temporal se describe como un **proceso estocástico**. Tiene variación aleatoria y su distribución cambia con el tiempo. Tiene una media y una varianza y estos valores también cambian.

Un proceso estocástico es **estacionario** si su distribución no cambia con el tiempo. Un ejemplo de proceso estocástico estacionario son las fluctuaciones periódicas de valores.



Si la distribución cambia, entonces el proceso *estocástico* es **no estacionario.**

No podemos averiguar la distribución de una serie temporal. Entonces, la serie temporal estacionaria es una serie en la que la media y la desviación estándar no cambian. De las dos series, aquella en la que la media y la desviación estándar cambian más lentamente es “más estacionaria”.

Las series de tiempo no estacionarias son más difíciles de pronosticar porque sus propiedades cambian demasiado rápido.

Diferencias de series temporales

Las **diferencias de series temporales** son una secuencia de diferencias entre elementos vecinos de una serie temporal (es decir, el valor anterior se resta del siguiente).

El método `shift()` se usa para encontrar las diferencias de series temporales. Todos los valores se desplazan un paso hacia adelante a lo largo del eje de tiempo:

```
data = pd.Series([0.5, 0.7, 2.4, 3.2])
print(data)
print(data.shift())
```

```
0    0.5
1    0.7
2    2.4
3    3.2
dtype: float64
0    NaN
1    0.5
2    0.7
3    2.4
dtype: float64
```

El último valor de la serie se pierde porque no hay lugar para cambiarlo. El valor cero es `NaN`, ya que no tiene valor. Agrega un argumento para completar los valores ausentes.

```
import pandas as pd

data = pd.Series([0.5, 0.7, 2.4, 3.2])
print(data)
print(data.shift(fill_value=0))
```

```
0    0.5
1    0.7
2    2.4
3    3.2
dtype: float64
0    0.0
1    0.5
2    0.7
```

```
3    2.4
dtype: float64
```

Las diferencias de las series temporales son más estacionarias que la serie en sí. Por ejemplo, una tendencia no lineal se convierte en lineal: