Máster Universitario en *Data Analytics for Business*

Análisis exploratorio de datos

Alexandra Abós

upf. BARCELONA SCHOOL OF MANAGEMENT

Diciembre-2025

Unidad 9: Análisis exploratorio de series temporales

upf. BARCELONA SCHOOL OF MANAGEMENT

9.

Análisis exploratorio de series temporales

1. ¿Qué es una serie temporal?
2. Visualización de series temporales
3. Gestión de datos tipo fecha
4. Detección y gestión de datos nulos y atípicos en series temporales
5. Suavizado de series temporales

upf. BARCELONA SCHOOL OF MANAGEMENT

1

¿Qué es una serie temporal?

Una **serie temporal** se define como una colección de observaciones de una variable recogidas secuencialmente en el tiempo. Su análisis permite identificar y modelar patrones temporales y tendencias.

Las observaciones consecutivas no son independientes, se debe considerar la secuencia temporal de estas observaciones.

Ejemplos: previsión del tiempo, análisis financiero, predicción de ventas, control de procesos, vigilancia epidemiológica.

1

¿Qué es una serie temporal?

Cuando se trabaja con series temporales, se distinguen dos objetivos principales:

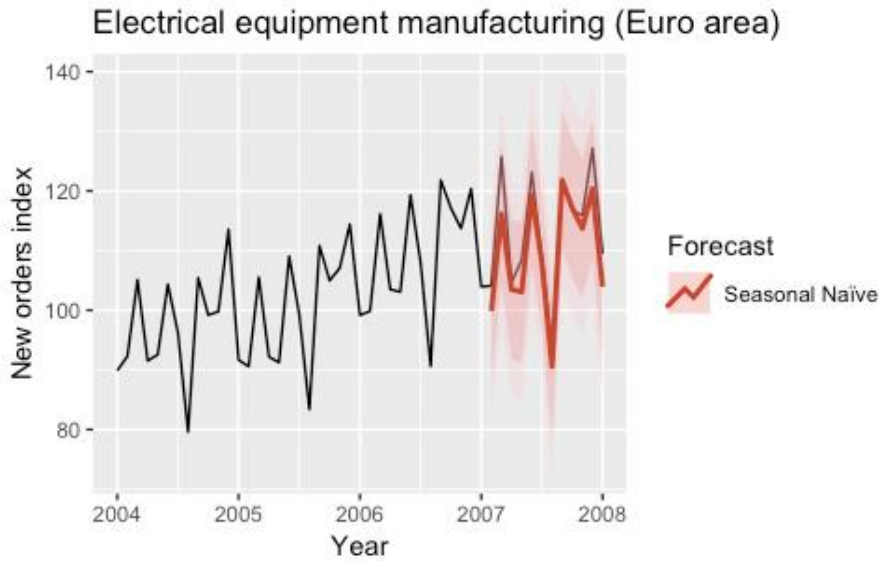
Descripción: visualizar y calcular las medidas descriptivas básicas de una serie temporal.

Es importante tener en cuenta:

- La presencia de tendencias en los datos (ascendente, descendente).
- Si hay estacionalidad (seasonality) en los datos.
- La identificación de posibles valores atípicos o observaciones inusuales.

¿Qué es una serie temporal?

Predicción: usar la información del pasado para poder predecir los valores futuros.



1

Componentes de una serie temporal

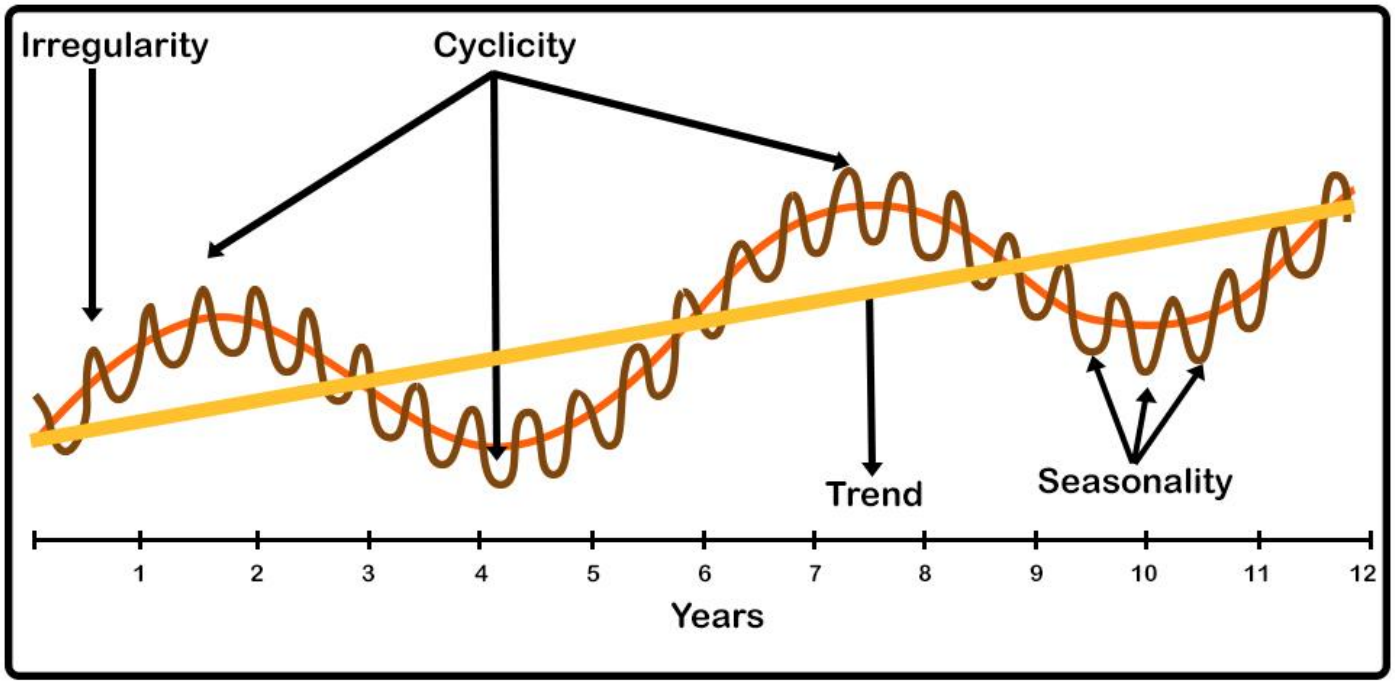
Una serie temporal se puede descomponer en distintas componentes básicas:

- **Tendencia**: representa la dirección general de la serie temporal a lo largo del tiempo (ascendente, descendente, sin patrón).

- **Estacionalidad**: refleja patrones que se repiten a intervalos regulares dentro de un período específico (semanal, mensual, anual, etc).

- **Componente Aleatoria o ruido**: Una vez identificados los componentes anteriores y después de haberlos eliminado, quedan unos valores que son aleatorios (ruido), que contribuyen a la variabilidad de la serie temporal.

Componentes de una serie temporal



2

Visualización de series temporales

La visualización de series temporales es fundamental para entender patrones, identificar tendencias, y detectar eventos significativos a lo largo del tiempo.

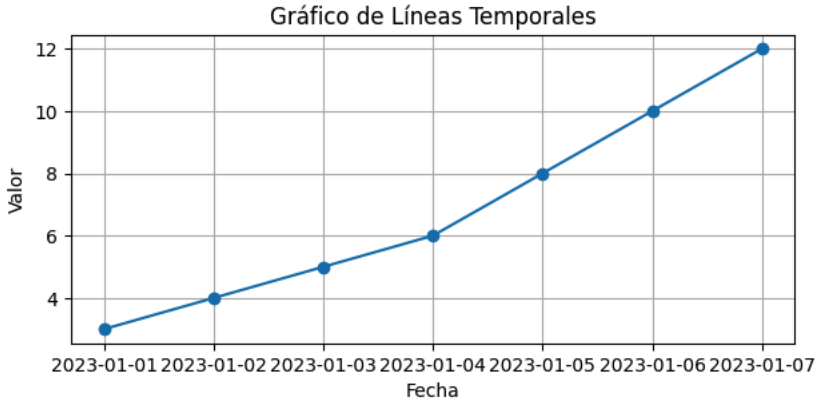
Los gráficos más usados son:

- Gráfico de **Líneas Temporales**: permite representar la evolución temporal de la variable.
- Gráfico de **Barras Temporales**: permite identificar y resaltar patrones estacionales o variaciones/outliers.
- Gráfico de **Áreas Temporales**: útil para mostrar cómo contribuyen diferentes partes a la serie en un momento temporal concreto.

2

Visualización de series temporales

```python
# Gráfico de Líneas Temporales

import matplotlib.pyplot as plt
import pandas as pd

# Creamos una serie temporal
datos = pd.Series([3, 4, 5, 6, 8, 10, 12],
                  index=pd.date_range(start='2023-01-01', periods=7))

# Especificamos las características del gráfico
plt.figure(figsize=(7, 3))
plt.plot(datos.index, datos.values, marker='o', linestyle='-')
plt.title('Gráfico de Líneas Temporales')
plt.xlabel('Fecha')
plt.ylabel('Valor')
plt.grid(True)
plt.show()
```

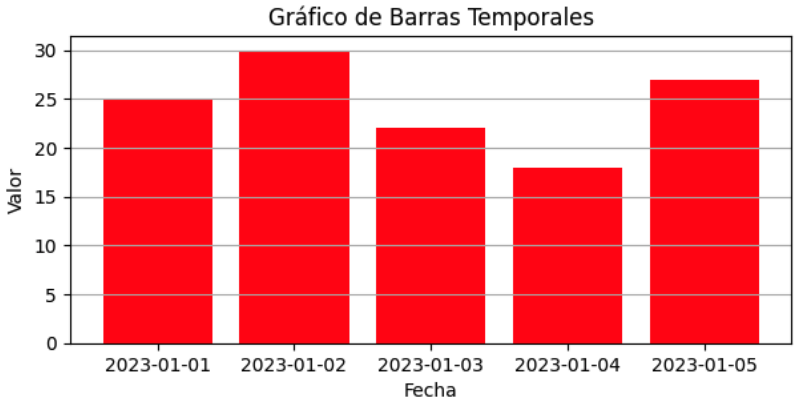


upf. BARCELONA SCHOOL OF MANAGEMENT

2

Visualización de series temporales

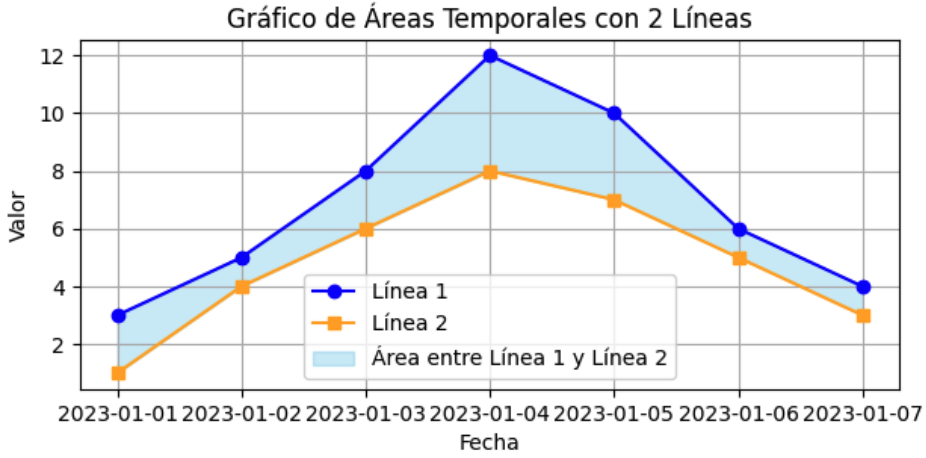
```
# Gráfico de Barras Temporales

# Creamos unos datos mensuales
datos_mensuales = pd.Series(
    [25, 30, 22, 18, 27],
    index=pd.date_range(start='2023-01-01', periods=5))

# Especificamos las características del gráfico
plt.figure(figsize=(7, 3))
plt.bar(datos_mensuales.index, datos_mensuales.values, color='red')
plt.title('Gráfico de Barras Temporales')
plt.xlabel('Fecha')
plt.ylabel('Valor')
plt.grid(axis='y')
plt.show()
```



2

Visualización de series temporales

```
# Gráfico de Áreas Temporales

# Creamos 2 series temporales
fechas = pd.date_range(start='2023-01-01', periods=7)
linea1 = pd.Series([3, 5, 8, 12, 10, 6, 4], index=fechas)
linea2 = pd.Series([1, 4, 6, 8, 7, 5, 3], index=fechas)

# Hacemos el gráfico de las dos líneas
plt.figure(figsize=(7, 3))
plt.plot(linea1.index, linea1.values,
         label='Línea 1', marker='o',
         linestyle='-', color='blue')
plt.plot(linea2.index, linea2.values,
         label='Línea 2', marker='s',
         linestyle='-', color='orange')

# Coloreamos el area entre las líneas
plt.fill_between(linea1.index, linea1.values, linea2.values,
                 color='skyblue', alpha=0.4,
                 label='Área entre Línea 1 y Línea 2')

# Especificamos las características del gráfico
plt.title('Gráfico de Áreas Temporales con 2 Líneas')
plt.xlabel('Fecha')
plt.ylabel('Valor')
plt.legend()
plt.grid(True)
plt.show()
```



2

Visualización de series temporales

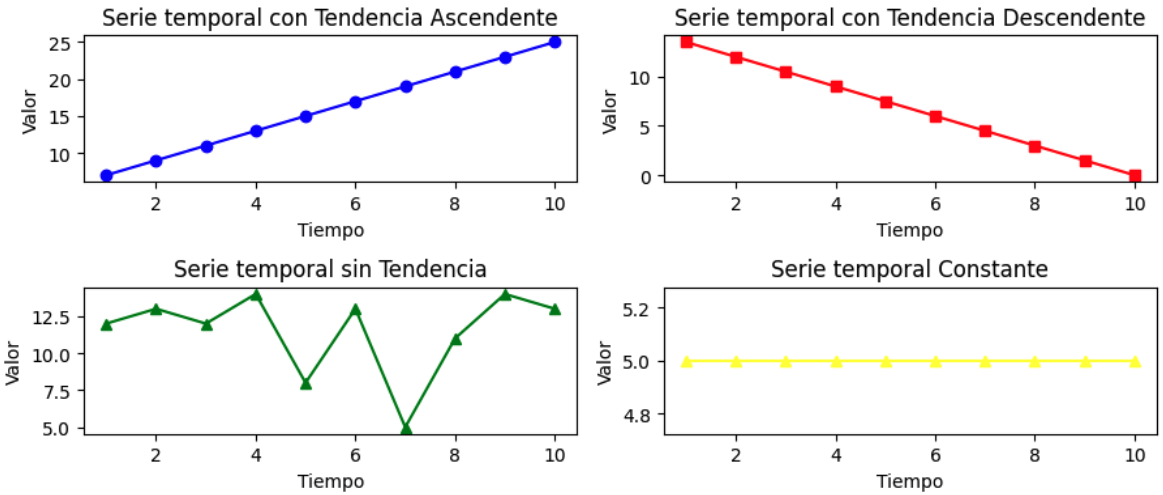
Ahora vamos a ver series temporales con distintas tendencias.

```
# Ejemplos tendencia

import matplotlib.pyplot as plt
import numpy as np

# Generamos 4 series temporales con distinta tendencia
tiempo = np.arange(1, 11)
linea_ascendente = 2 * tiempo + 5
linea_descendente = -1.5 * tiempo + 15
linea_sin_tendencia = np.random.randint(5, 15, size=10)
linea_constante = np.ones(10)*5
```

2

Visualización de series temporales

```python
# Creamos el subplot con cuatro series temporales
fig, axs = plt.subplots(2, 2, figsize=(9, 4))

# Serie temporal con tendencia ascendente
axs[0][0].plot(tiempo, linea_ascendente,
               marker='o', linestyle='-', color='blue')
axs[0][0].set_title('Serie temporal con Tendencia Ascendente')
axs[0][0].set_xlabel('Tiempo')
axs[0][0].set_ylabel('Valor')

# Serie temporal con tendencia descendente
axs[0][1].plot(tiempo, linea_descendente,
               marker='s', linestyle='-', color='red')
axs[0][1].set_title('Serie temporal con Tendencia Descendente')
axs[0][1].set_xlabel('Tiempo')
axs[0][1].set_ylabel('Valor')

# Serie temporal sin tendencia
axs[1][0].plot(tiempo, linea_sin_tendencia,
               marker='^', linestyle='-', color='green')
axs[1][0].set_title('Serie temporal sin Tendencia')
axs[1][0].set_xlabel('Tiempo')
axs[1][0].set_ylabel('Valor')

# Serie temporal constante
axs[1][1].plot(tiempo, linea_constante,
               marker='^', linestyle='-', color='yellow')
axs[1][1].set_title('Serie temporal Constante')
axs[1][1].set_xlabel('Tiempo')
axs[1][1].set_ylabel('Valor')

# Ajustamos el diseño y mostramos gráfico
plt.tight_layout()
plt.show()
```



2

Cálculo de la tendencia

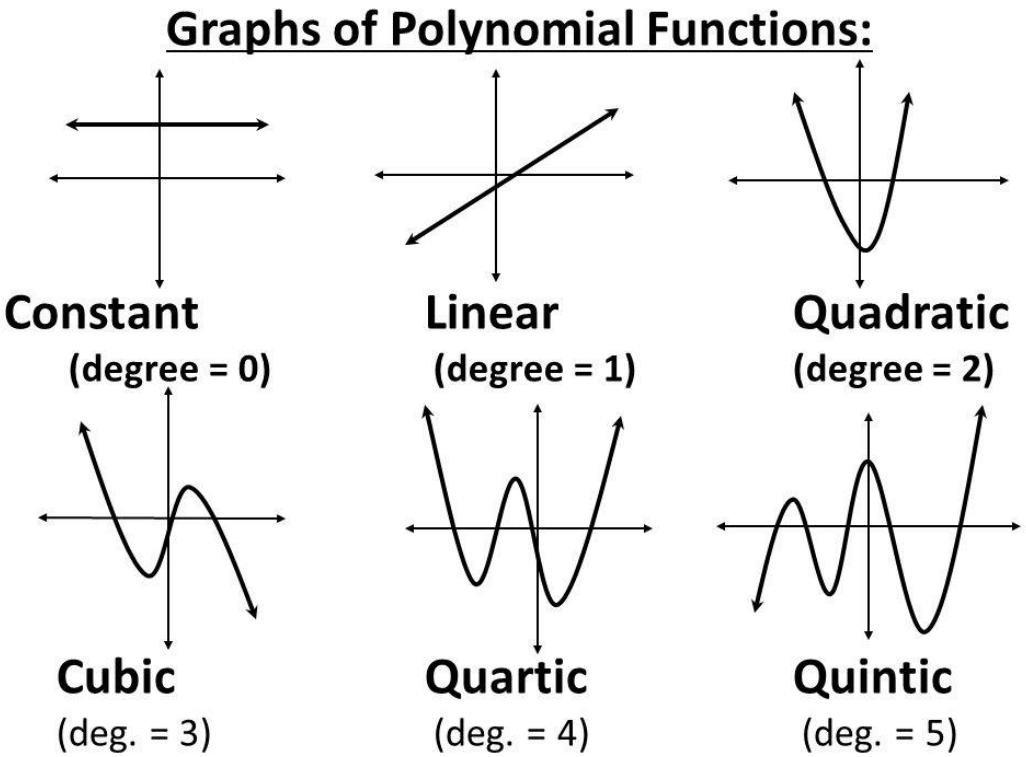
Para calcular de manera analítica la **tendencia**, se puede ajustar un polinomio de primer grado.

Un polinomio de primer grado está caracterizado por dos parámetros: la **pendiente** y la **ordenada** en el origen.

- Pendiente > 0 : tendencia ascendente.
- Pendiente = 0: sin tendencia.
- Pendiente < 0: tendencia descendente.

2

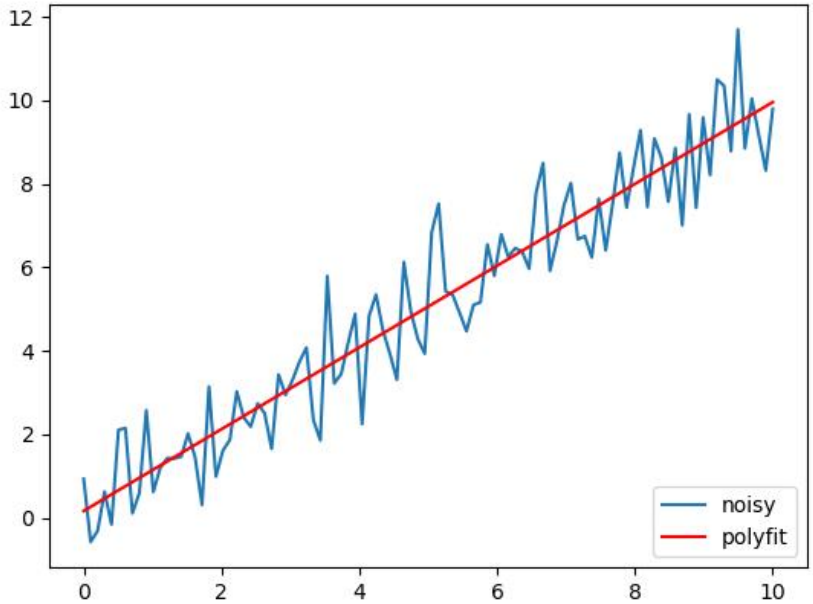
Cálculo de la tendencia



2

Cálculo de la tendencia

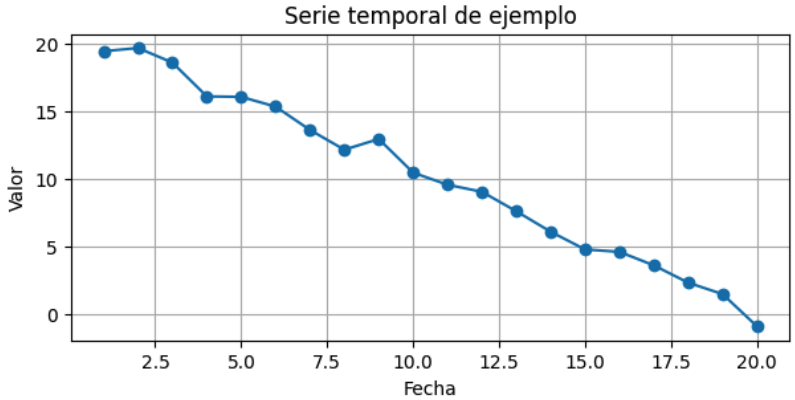
¿Qué valor de pendiente tendría esta serie temporal?



2

Cálculo de la tendencia

```
# Creamos fechas para la serie temporal
# con 20 valores
tiempo = np.arange(1, 21)

# Generamos valores con tendencia ascendente
# Valores linealmente descendente de 20 a 0 con ruido
valores = np.linspace(20, 0, num=20) + np.random.normal(0, 0.8, size=20)

# Creamos el DataFrame con fechas y valores
ts_df = pd.DataFrame({'Fecha': tiempo, 'Valor': valores})
```



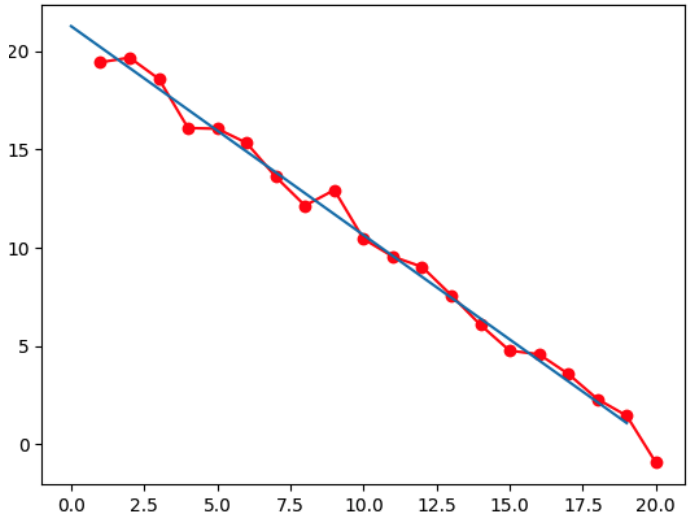
2

Cálculo de la tendencia

```
# Usamos la función polyfit, que ajusta un polinomio a los datos
# En este caso, usamos una función de grado 1
coefficients, _, _, _, _ = np.polyfit(tiempo,valores,1,full=True)
print('Slope ' + str(coefficients[0]))
```

```
Slope -1.0799227016094741
```

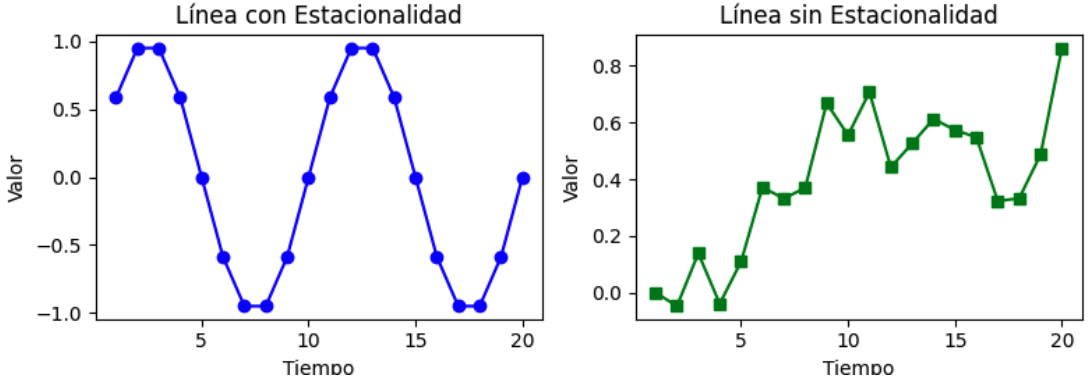
```
# Mostramos la serie temporal descendente con la línea del fit
plt.plot(tiempo, valores, marker='o', linestyle='-', color='red')
plt.plot([coefficients[0]*x + coefficients[1] for x in range(len(valores))])
plt.show()
```



upf. BARCELONA SCHOOL OF MANAGEMENT

Estacionalidad - Ejemplos

```python
# Ejemplos de estacionalidad

import matplotlib.pyplot as plt
import numpy as np

# Generaramos dos líneas temporales, con y sin estacionalidad
tiempo = np.arange(1, 21)
linea_con_estacionalidad = np.sin(2 * np.pi * tiempo / 10)
linea_sin_estacionalidad = np.cumsum(np.random.normal(0, 0.2, size=20))
```



2

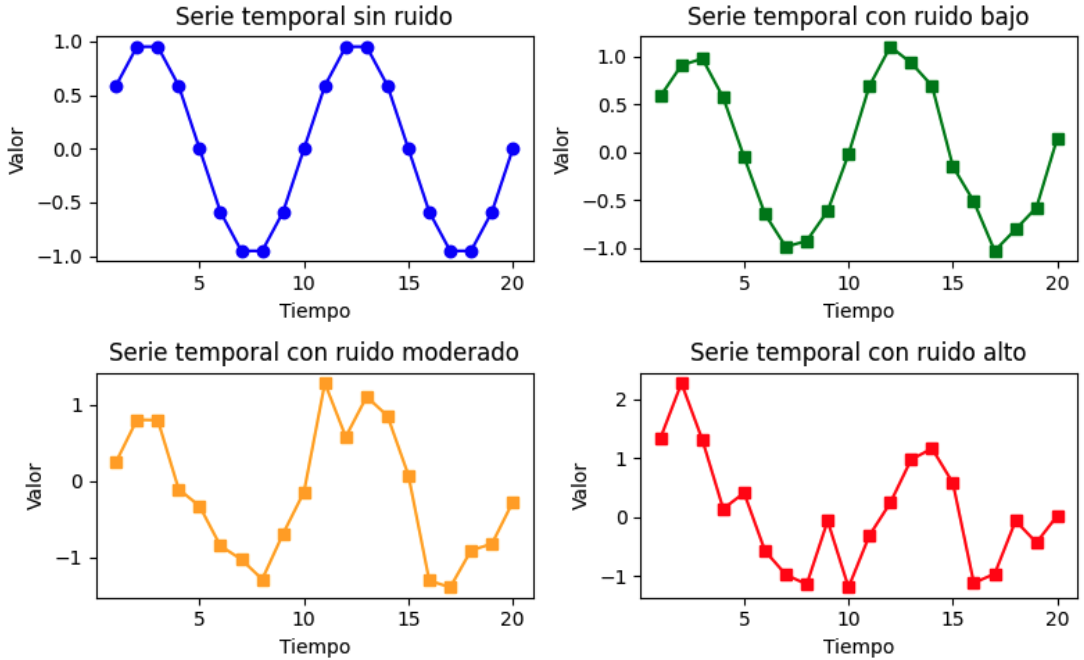
Efecto de ruido - Ejemplos

La cantidad de ruido afecta a los valores de la serie temporal.

Ahora añadimos ruido de distinta magnitud:

```
# Ejemplo ruido

# Generaramos dos líneas temporales
# con y sin ruido
tiempo = np.arange(1, 21)
linea_sin_ruido = np.sin(2 * np.pi * tiempo / 10)
linea_con_ruido_bajo = np.sin(2 * np.pi * tiempo / 10) + np.random.normal(0, 0.1, size=20)
linea_con_ruido_moderado = np.sin(2 * np.pi * tiempo / 10) + np.random.normal(0, 0.4, size=20)
linea_con_ruido_alto = np.sin(2 * np.pi * tiempo / 10) + np.random.normal(0, 0.8, size=20)
```

Efecto de ruido - Ejemplos



2

Descomposición serie temporal

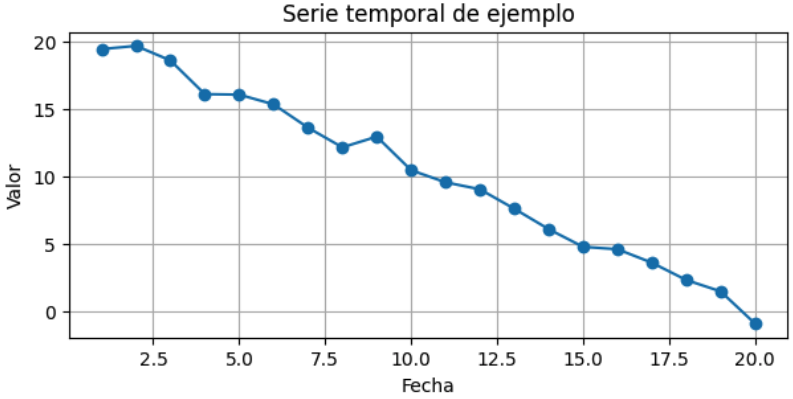
Si queremos hacer un análisis global de la serie temporal podemos descomponerla en las 3 componentes básicas:

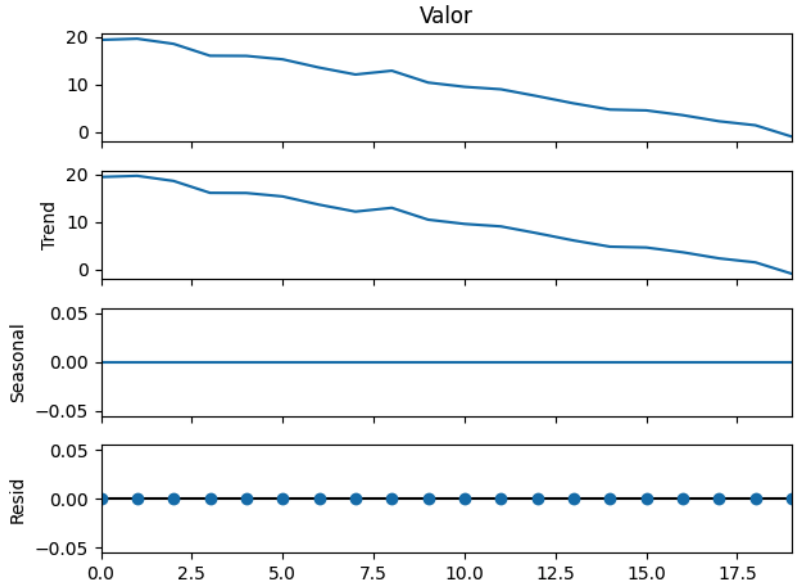
```
# Si queremos hacer un análisis global de la serie temporal
# podemos descomponerla en las 3 componentes básicas

# Cargamos la función seasonal_decompose
from statsmodels.tsa.seasonal import seasonal_decompose

# Aplicamos la descomposición estacional
resultados = seasonal_decompose(
    ts_df.Valor, period=1)

# Mostramos los resultados
resultados.plot()
plt.show()
```

2

Descomposición serie temporal

Si queremos hacer un análisis global de la serie temporal podemos descomponerla en las 3 componentes básicas:





3

Gestión de datos tipo fecha

La gestión de datos de tipo fecha y hora es esencial en el análisis de series temporales y datos que involucran información temporal.

En Python, el módulo **datetime** proporciona las funcionalidades para trabajar con fechas y horas.

3

Gestión de datos tipo fecha

```
# Creación de Objetos datetime

# Se pueden crear objetos datetime para representar fechas y horas
# usando la clase datetime del módulo datetime.
from datetime import datetime

# Especificamos fecha y hora actual
fecha_actual = datetime.now()
print("Fecha y hora actual:", fecha_actual)

# Creamos un objeto datetime específico
fecha_especifica = datetime(2025, 12, 1, 15, 30, 0)
print("Fecha y hora específica:", fecha_especifica)
```

```
Fecha y hora actual: 2025-11-29 16:11:51.190235
Fecha y hora específica: 2025-12-01 15:30:00
```

upf. BARCELONA SCHOOL OF MANAGEMENT

3

Gestión de datos tipo fecha

Se puede formatear objetos datetime como cadenas de texto y viceversa.

- **Parsear** consiste en convertir una cadena de texto que representa una fecha en un objeto de fecha en python.
- **Formatear** una fecha significa representar un objeto de fecha en formato de cadena de texto.

```
# Formateamos un datetime como cadena
fecha_formateada = fecha_actual.strftime("%Y-%m-%d %H:%M:%S")
print("Fecha formateada:", fecha_formateada)
print("Tipo de la variable:", type(fecha_formateada))

# Parseamos una cadena de fecha
cadena_fecha = "12/01/2025"
fecha_parseada = datetime.strptime(cadena_fecha, "%d/%m/%Y")
print("Fecha parseada:", fecha_parseada)
print("Tipo de la variable:", type(fecha_parseada))
```

```
Fecha formateada: 2025-11-29 16:11:51
Tipo de la variable: <class 'str'>
Fecha parseada: 2025-01-12 00:00:00
Tipo de la variable: <class 'datetime.datetime'>
```

Gestión de datos tipo fecha

```
# Operaciones con datetime

# Se pueden realizar operaciones aritméticas
# y comparaciones con objetos datetime

# Calculamos la diferencia entre dos fechas
diferencia = fecha_especifica - fecha_actual
print("Diferencia entre fechas:", diferencia)

# Verificamos si una fecha es anterior o posterior a otra
es_anterior = fecha_actual < fecha_especifica
print("¿Fecha actual es anterior a fecha específica?", es_anterior)
```

```
Diferencia entre fechas: 1 day, 23:18:08.809765
¿Fecha actual es anterior a fecha específica? True
```

4

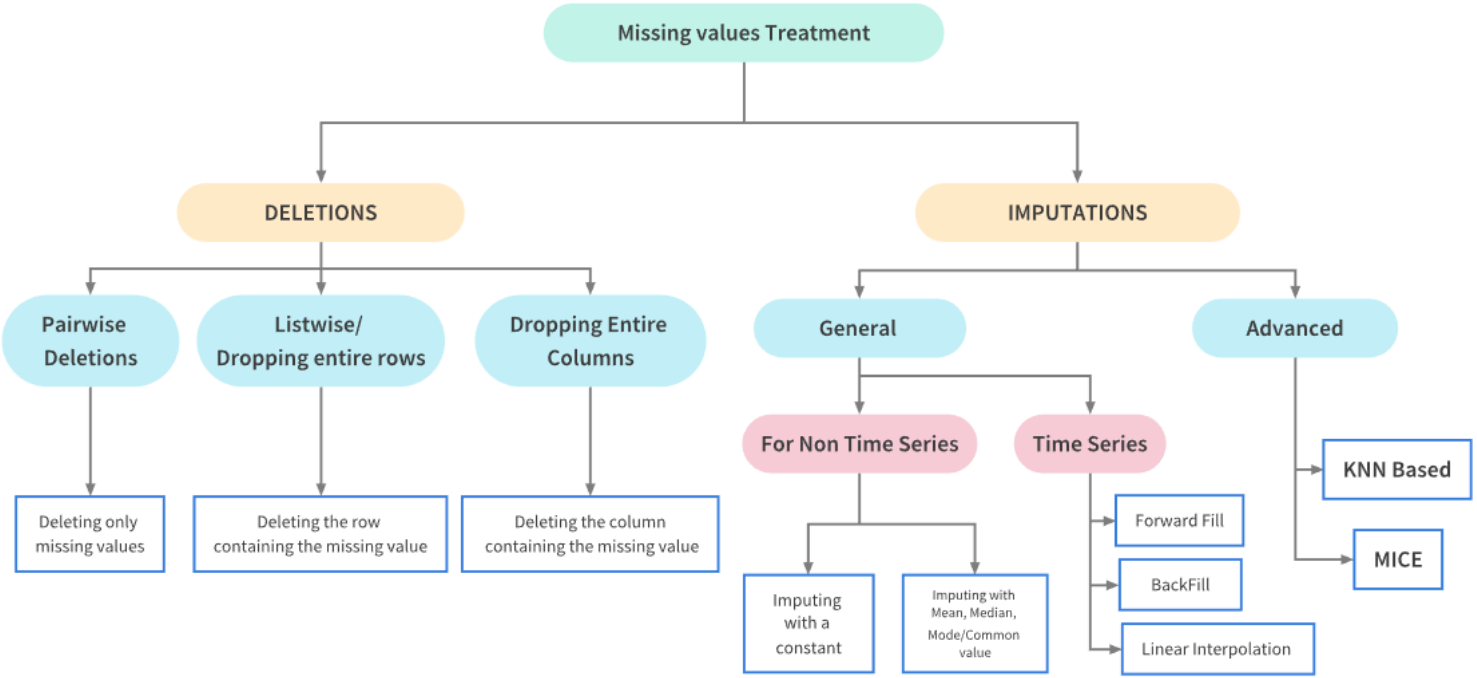
Detección y gestión de datos nulos y atípicos en series temporales

La detección y gestión de datos nulos o atípicos es un punto clave para el análisis y posterior modelaje, para evitar tener resultados erróneos o poco interpretables.

Los pasos son similares a los usados para otros tipos de datos:

- Identificación de datos nulos/anómalos: funciones (isna(), isnull()) o gráficos para la visualización.
- Tratamiento de datos nulos/anómalos

Detección y gestión de datos nulos y atípicos en series temporales



Detección y gestión de datos nulos y atípicos en series temporales

Las técnicas de gestión de datos nulos para series temporales más comunes son:

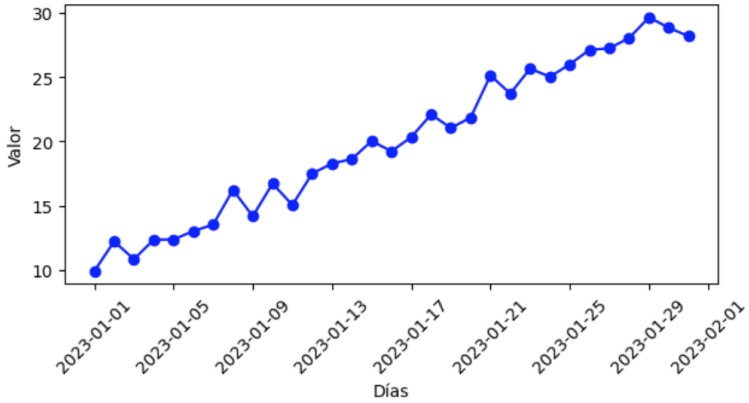
- **Last Observation Carried Forward** (LOCF): Reemplaza los valores faltantes con el último valor conocido.
- **Next Observation Carried Backward** (NOCB): Reemplaza los valores faltantes con el próximo valor conocido.
- **Interpolación Lineal**: Estima los valores nulos trazando una línea recta entre los dos puntos de datos conocidos más cercanos.
- **Interpolación Spline**: Estima los valores nulos ajustando una línea curva flexible a través de los puntos de datos.

Detección y gestión de datos nulos y atípicos en series temporales

Vamos a generar unos datos aleatorios con valores nulos para testear las distintas técnicas.

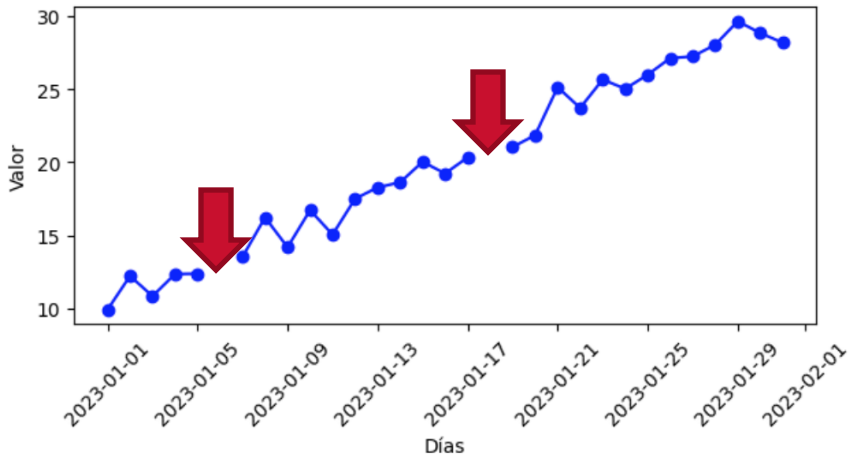
```python
import pandas as pd
import numpy as np
import datetime
import matplotlib.pyplot as plt

# Creamos fechas para la serie temporal
# que representan los 31 días de Enero
fechas = pd.date_range(start='2023-01-01', periods=31, freq='D')

# Generamos valores con tendencia ascendente
# Valores linealmente crecientes de 10 a 30
# con ruido
valores = np.linspace(10, 30, num=31) + np.random.normal(0, 0.8, size=31)


# Creamos el DataFrame con fechas y valores
enero_df = pd.DataFrame({'Fecha': fechas, 'Valor': valores})

# Mostramos la serie temporal
enero_df.head(5)
```

	Fecha	Valor
0	2023-01-01	9.901301
1	2023-01-02	12.209411
2	2023-01-03	10.827082
3	2023-01-04	12.343811
4	2023-01-05	12.368013



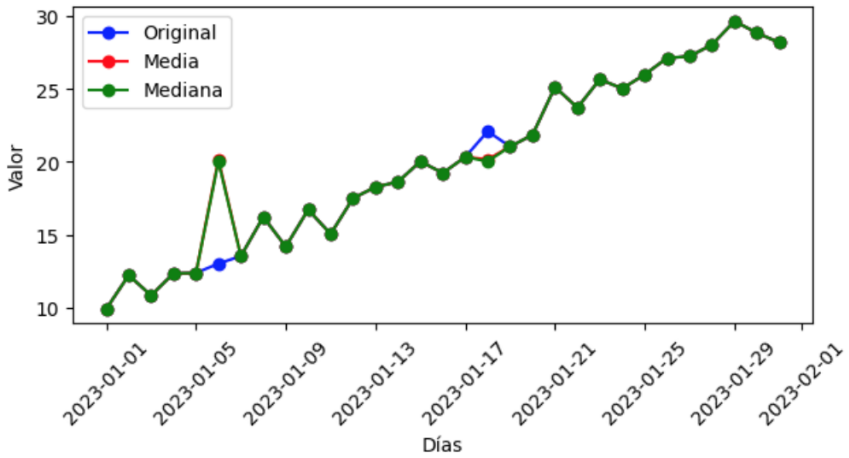
Detección y gestión de datos nulos y atípicos en series temporales

```
# Añadimos algunos valores missing
# para testear las distintas técnicas

# Escogemos dos índices aleatorios
idx_rand = [5, 17]

# Sustituímos el valor de esos índices por NaNs
enero_df.loc[idx_rand, 'Valor'] = np.nan
```



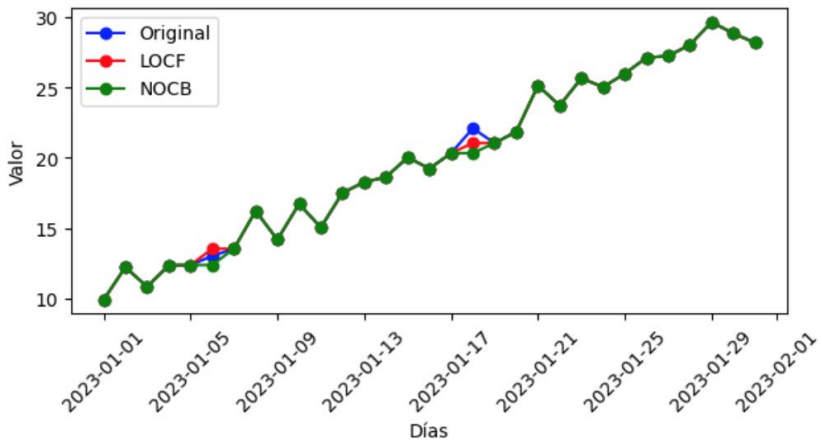
Detección y gestión de datos nulos y atípicos en series temporales

```
# Ahora vamos a testear las distintas técnicas

# Media & Mediana

# Sustituímos los NaNs por la media/mediana de la columna
enero_mean_df = enero_df.assign(Valor=enero_df.Valor.fillna(enero_df.Valor.mean()))
enero_median_df = enero_df.assign(Valor=enero_df.Valor.fillna(enero_df.Valor.median()))
```



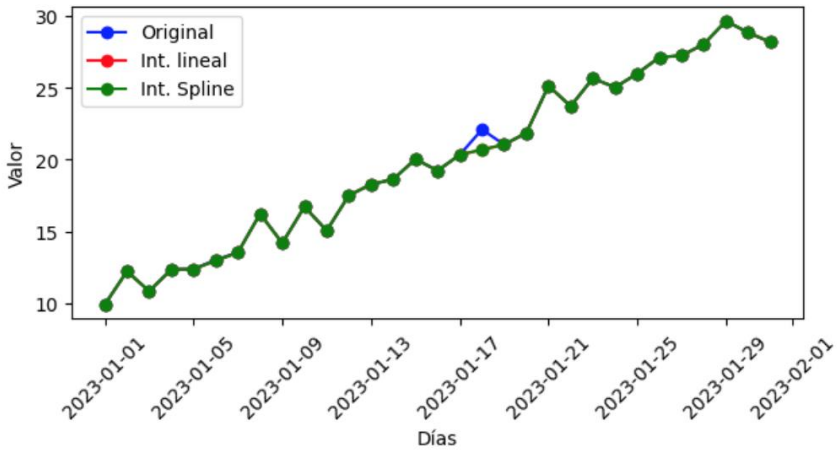
Detección y gestión de datos nulos y atípicos en series temporales

```
# LOCF & NOCB

# Creamos nuevos df para guardar los resultados
enero_locf_df = enero_df.copy()
enero_nocb_df = enero_df.copy()

# Usamos los métodos LOCF & NOCB
enero_locf_df['Valor']= enero_locf_df['Valor'].fillna(method ='bfill')
enero_nocb_df['Valor']= enero_nocb_df['Valor'].fillna(method ='ffill')
```



Detección y gestión de datos nulos y atípicos en series temporales

```
# Interpolación lineal y spline

# Creamos nuevos df para guardar los resultados
enero_int_lin_df = enero_df.copy()
enero_int_spline_df = enero_df.copy()

# Usamos la interpolación lineal y spline
enero_int_lin_df['Valor']= enero_int_lin_df['Valor'].interpolate(method='linear')
enero_int_spline_df['Valor']= enero_int_spline_df['Valor'].interpolate(option='spline')
```



5

Suavizado de series temporales

El **suavizado de series temporales** es un proceso que tiene como objetivo reducir el ruido o variabilidad en los datos para mostrar patrones o tendencias más claras.

Existen varias técnicas de suavizado, y la elección de una depende de la naturaleza de los datos y del objetivo del análisis.

- Promedio Móvil
- Suavizado Exponencial
- Transformaciones Logarítmicas o Box-Cox
- Filtros FIR
- Spline Cúbico
- Transformaciones Logarítmicas o Box-Cox

5

Suavizado de series temporales

El **promedio móvil** (rolling window) calcula el promedio de un conjunto de puntos de datos adyacentes. Puede ser un promedio simple o ponderado.

El **suavizado exponencial** (expontential smoothing) asigna pesos decrecientes exponencialmente a los puntos de datos anteriores. Es útil para capturar tendencias a largo plazo.

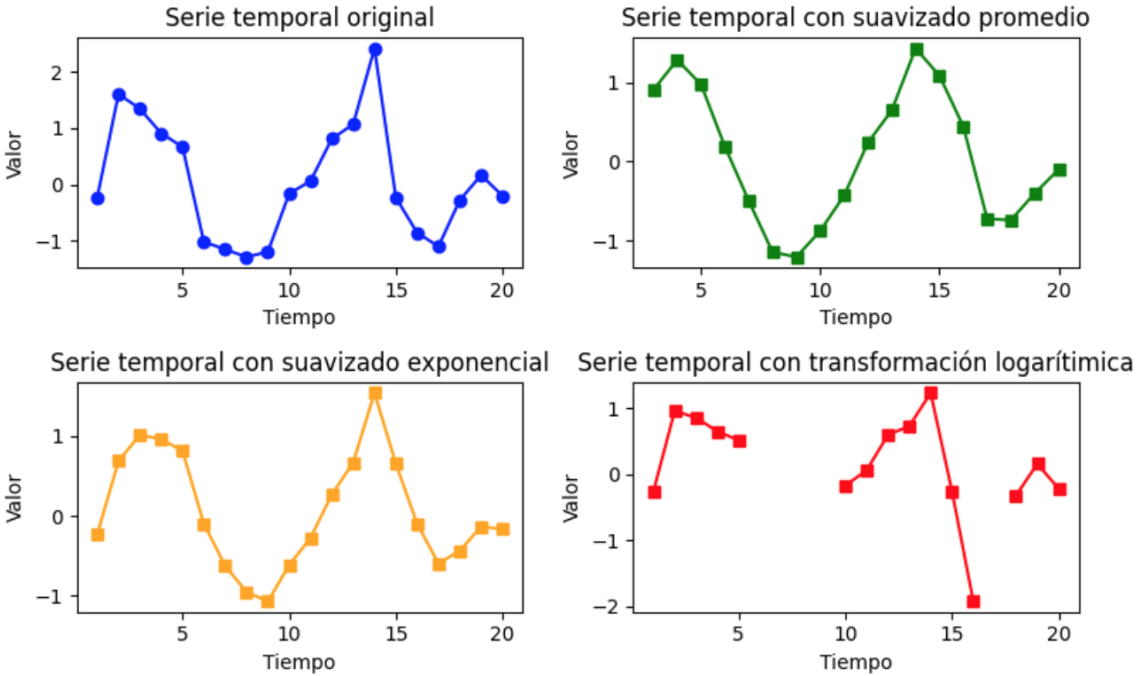
Las **transformaciones Logarítmicas** o **Box-Cox** consisten en aplicar transformaciones como logaritmos puede estabilizar la varianza y suavizar la serie.

5

Suavizado de series temporales

```
# Vamos a ver estas técnicas de suavizado con la serie temporal con ruido

# Definimos el eje temporal
tiempo = np.arange(1, 21)

# Creamos una serie temporal con ruido
st_ruido_alto = np.sin(2 * np.pi * tiempo / 10) + np.random.normal(0, 0.8, size=20)

# Transformamos el array en dataframe para poder usar las funciones
st_ruido_alto_df = pd.DataFrame(st_ruido_alto)

# Aplicamos un promedio móvil simple
serie_suavizada_promedio = st_ruido_alto_df.rolling(window=3).mean()

# Aplicamos un suavizado exponencial
serie_suavizada_exp = st_ruido_alto_df.ewm(span=3, adjust=False).mean()

# Aplicamos una transformación logarítmica
serie_transformada_log = np.log1p(st_ruido_alto_df)
```

Suavizado de series temporales



5

Suavizado de series temporales

Los **filtros FIR** (Finite Impulse Response) aplican coeficientes a los datos para suavizar la serie. Pueden ser diseñados para suavizar específicamente frecuencias de interés.

Los **splines cúbicos** interpolan los datos con polinomios cúbicos suaves. Son útiles para suavizar la serie mientras preservan mejor la forma de los datos originales.

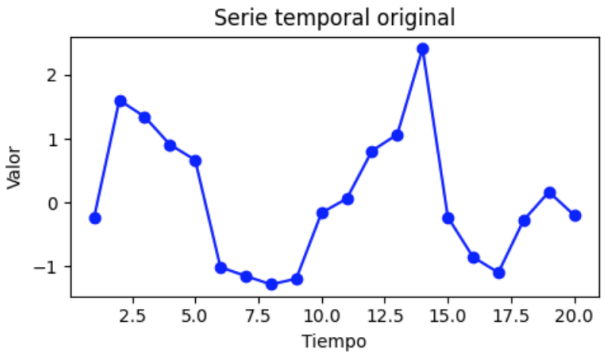
5

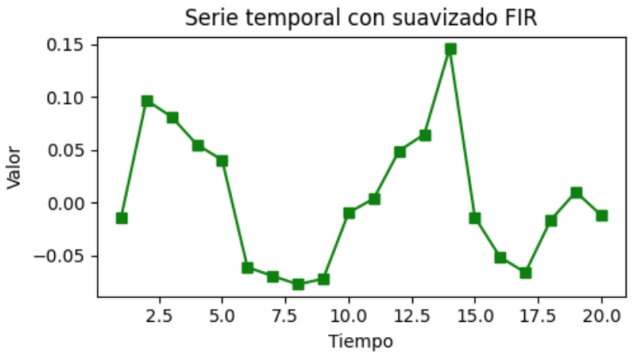
Suavizado de series temporales

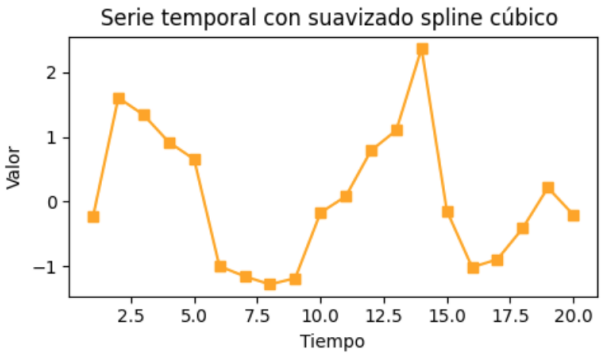
```
# Cargamos las funciones necesarias
from scipy.signal import firwin, lfilter
from scipy.interpolate import UnivariateSpline

# Aplicamos un filtro FIR
coeficientes_fir = firwin(numtaps=3, cutoff=0.3)
serie_suavizada_fir = lfilter(coeficientes_fir, 1.0, st_ruido_alto_df)

# Aplicamos un spline cúbico
spline = UnivariateSpline(tiempo, st_ruido_alto_df, s=0.1)
serie_suavizada_cub = spline(tiempo)
```

5

Suavizado de series temporales







upf. BARCELONA SCHOOL OF MANAGEMENT

Extra: Cómo trabajar con NaNs

	Fecha	Valor
0	2023-01-01	9.916944
1	2023-01-02	8.534285
2	2023-01-03	10.818799
3	2023-01-04	10.921833
4	2023-01-05	12.758009

```python
import pandas as pd
import numpy as np
import datetime
import matplotlib.pyplot as plt

# Creamos fechas para la serie temporal
# que representan los 31 días de Enero
fechas = pd.date_range(start='2023-01-01', periods=31, freq='D')

# Generamos valores con tendencia ascendente
# Valores linealmente crecientes de 10 a 30
# con ruido
valores = np.linspace(10, 30, num=31) + np.random.normal(0, 0.8, size=31)


# Creamos el DataFrame con fechas y valores
enero_df = pd.DataFrame({'Fecha': fechas, 'Valor': valores})

# Mostramos la serie temporal
enero_df.head(5)
```

6

Extra: Cómo trabajar con NaNs

```python
# Especificamos un nuevo formato
formato_personalizado = "%d$%m$%Y"

# Transformamos las fechas a este nuevo formato
list_dates = enero_df['Fecha']
enero_df['Fecha'] = [date.strftime(formato_personalizado) for date in list_dates]

# Mostramos el nuevo formato de Fecha
enero_df.head(5)
```

	Fecha	Valor
0	2023-01-01	9.916944
1	2023-01-02	8.534285
2	2023-01-03	10.818799
3	2023-01-04	10.921833
4	2023-01-05	12.758009

Formato original

	Fecha	Valor
0	01$01$2023	9.916944
1	02$01$2023	8.534285
2	03$01$2023	10.818799
3	04$01$2023	10.921833
4	05$01$2023	12.758009

Formato nuevo

upf. BARCELONA SCHOOL OF MANAGEMENT

6

Extra: Cómo trabajar con NaNs

```
enero1_df = enero_df.copy()

# Añadimos NaNs a la fecha
enero1_df.iloc[1, 0] = np.nan

# Mostramos el df
enero1_df.head(5)
```

	Fecha	Valor
0	01$01$2023	9.916944
1	02$01$2023	8.534285
2	03$01$2023	10.818799
3	04$01$2023	10.921833
4	05$01$2023	12.758009

Formato original

	Fecha	Valor
0	01$01$2023	9.916944
1	NaN	8.534285
2	03$01$2023	10.818799
3	04$01$2023	10.921833
4	05$01$2023	12.758009

Formato nuevo

upf. BARCELONA SCHOOL OF MANAGEMENT

6

Extra: Cómo trabajar con NaNs

```
## Opción 1
# Podemos transformar las fechas al formato correcto sin problemas
enero1_df['Fecha'] = pd.to_datetime(enero1_df['Fecha'], format="%d$%m$%Y")
enero1_df.head(5)
```

	Fecha	Valor
0	2023-01-01	9.916944
1	NaT	8.534285
2	2023-01-03	10.818799
3	2023-01-04	10.921833
4	2023-01-05	12.758009

No hay problemas con los NaNs!

6

Extra: Cómo trabajar con NaNs

```
## Opción 2

# Seleccionamos las filas que no son NaNs
idx_dates = enero1_df[~enero1_df.Fecha.isna()].index

# Aplicamos la transformación solo en esas filas
enero1_df.loc[idx_dates, 'Fecha'] = pd.to_datetime(enero1_df.loc[idx_dates, 'Fecha'], format="%d$%m$%Y")

# Mostramos el df
enero1_df.head(5)
```

	Fecha	Valor
0	2023-01-01 00:00:00	9.916944
1	NaN	8.534285
2	2023-01-03 00:00:00	10.818799
3	2023-01-04 00:00:00	10.921833
4	2023-01-05 00:00:00	12.758009

Opción más larga

upf. BARCELONA SCHOOL OF MANAGEMENT

6

Extra: Cómo trabajar con NaNs

```
enero2_df = enero_df.copy()

# Añadimos un string a la fecha
enero2_df.iloc[2, 0] = "IDK"

# Mostramos el df
enero2_df.head(5)
```

	Fecha	Valor
0	01$01$2023	9.916944
1	02$01$2023	8.534285
2	IDK	10.818799
3	04$01$2023	10.921833
4	05$01$2023	12.758009

```
# Podemos transformar ahora las fechas al formato correcto?
try:
  enero2_df['Fecha'] = pd.to_datetime(enero2_df['Fecha'], format="%d$%m$%Y")
  enero2_df.head(5)
except ValueError as e:
  print("[ERROR] ¡No se puede transformar!")
  print(str(e))
```

```
[ERROR] ¡No se puede transformar!
time data "IDK" doesn't match format "%d$%m$%Y", at position 2. You might want to try:
```

¡La transformación da error!

6

Extra: Cómo trabajar con NaNs

```
## Opcion 1
# Seleccionamos las filas que no tienen el valor IDK
idx_dates = enero2_df[enero2_df.Fecha!='IDK'].index

# Transformamos las filas que no son IDK
enero2_df.loc[idx_dates, 'Fecha'] = pd.to_datetime(enero2_df.loc[idx_dates, 'Fecha'], format="%d$%m$%Y")

# Mostramos el df
enero2_df.head(5)
```

	Fecha	Valor
0	2023-01-01 00:00:00	9.916944
1	2023-01-02 00:00:00	8.534285
2	IDK	10.818799
3	2023-01-04 00:00:00	10.921833
4	2023-01-05 00:00:00	12.758009

¡Ya no da error!

upf. BARCELONA SCHOOL OF MANAGEMENT

6

Extra: Cómo trabajar con NaNs

```
## Opcion 2
# Seleccionamos las filas que no tienen el valor IDK
enero2_df.Fecha = enero2_df.Fecha.replace('IDK', np.nan)

# Ahora podemos transformar las fechas al formato correcto sin problemas
enero2_df['Fecha'] = pd.to_datetime(enero2_df['Fecha'], format="%d$%m$%Y")

# Mostramos el df
enero2_df.head(5)
```

	Fecha	Valor
0	2023-01-01	9.916944
1	2023-01-02	8.534285
2	NaT	10.818799
3	2023-01-04	10.921833
4	2023-01-05	12.758009

¡Ya no da error!

upf. BARCELONA SCHOOL OF MANAGEMENT

Parte práctica

¿Qué vamos a ver?

- 1 Ejercicio para preprocesar el dataset con datos temporales.
- 1 Ejercicio para analizar las series temporales.

Parte práctica

Ejercicio 1

En este ejercicio vamos a trabajar con datos temporales de suscripciones de móvil de distintos países.

a) Carga los datos del archivo `mobile_subscriptions.csv` que encontrarás en la carpeta `data` en forma de DataFrame. Muestra el tamaño del archivo, sus columnas, y las 5 primeras filas. **Opcional**

b) Revisa los datos de las tres primeras columnas (`Location`, `Location Name`, `Location - RegionId`) y, si encuentras valores anómalos o missings, corrígelos. Razona el procedimiento que has escogido. **Opcional**

c) Tenemos la información de las suscripciones por año en columnas. Este formato de los datos (conocido como *wide*) no es muy óptimo para los análisis posteriores que queremos hacer. Por eso, vamos a reorganizar los datos para tener la información en formato *long*, es decir, con una columna que indique los años y una columna con los valores de las suscripciones.

Revisa este link de stackoverflow para ver como usar la función **melt** de la librería Pandas. **Opcional**

Podéis saltar directamente al apartado d)

Parte práctica

d) Si no has hecho los apartados a)-c), carga los datos del archivo `mobile_subscriptions_long.csv` que encontrarás en la carpeta `data` en forma de DataFrame.

Ahora que tenemos el df en formato long, queremos corregir los missings de la información temporal. Como vimos en teoría, hay distintas maneras de corrección possibles. En este apartado vamos a comparar los resultados usando dos técnicas:

- Reemplazar el NaN con la media de la serie temporal.
- Usar una interpolación lineal.

Aplica estas dos correcciones en las series temporales por país, es decir, si reemplazamos unos NaNs de Italía, la media o la interpolación se tiene que hacer teniendo en cuenta solo los datos de Italía.

Muestra gráficamente los resultados de las dos técnicas en un gráfico con las dos series temporales corregidas juntas para poder compararlas mejor. Comenta 3 ejemplos diferentes.

¿Qué técnica te parece más correcta en este caso?

Parte práctica

Ejercicio 2

Ahora que tenemos los datos preparados, vamos a analizarlos para responder algunas preguntas:

a) ¿Qué país tiene, en media, el mayor número de suscripciones? ¿Y el menor?

b) Si consideramos todas las suscripciones mundiales (la suma total) por año, hay una tendencia positiva (ascendente) del número de suscripciones? Calcula la pendiente de la serie temporal y muestra gráficamente la evolución anual de las suscripciones.

c) Si miramos a nivel de país, ¿hay algun país con tendencia negativa (descendente) de las suscripciones? ¿Qué país tiene la tendencia mayor? ¿Y la menor? Calcula la pendiente por país y muestra gráficamente la evolución temporal de los países con mayor/menor tendencia.

Nota: Para poder comparar las pendientes, calcula el coeficiente de la pendiente dividido por la media del país.

¿PREGUNTAS?

Alexandra Abós

alexandra.abos@bsm.upf.edu

¡GRACIAS!

BARCELONA SCHOOL OF MANAGEMENT

Alexandra Abós

alexandra.abos@bsm.upf.edu