

# De Ticker a Gráfico

Una Guía Visual para la Generación de  
Gráficos Financieros con Python

```
import matplotlib.pyplot as plt
import pandas as pd
data_frame = pd.read_csv('data.csv')
data_frame = data_frame[['date', 'open', 'close', 'low', 'high']]
data_frame = data_frame[['date', 'open', 'close', 'low', 'high']].set_index('date')
data_frame['color'] = np.where(data_frame['close'] > data_frame['open'], 'green', 'red')
data_frame['lat.inn'] = data_frame['close'].rolling(3).mean()
data_frame['meda'] = data_frame['close'].ewm(span=3, adjust=False).mean()
data_frame['primal.fara'] = "33"
plt.plot(lat.inn)
plt.plot(name, data_frame)
plt.plot("color", data_f
...
}
```

# Nuestro Destino: Un Gráfico Profesional y Automatizado

En esta guía, construiremos un script de Python capaz de generar un gráfico como este.

Aprenderemos no solo a visualizar datos, sino a hacerlo de una manera robusta y eficiente, lista para ser usada en servidores y aplicaciones automatizadas.

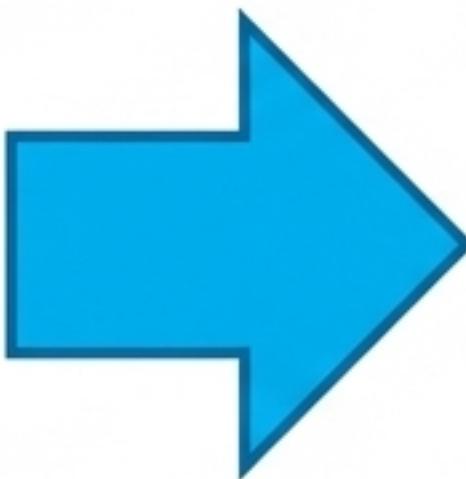


# La Hoja de Ruta: De los Datos Crudos a la Visualización



## Acto I: La Búsqueda de los Datos

Usaremos `yfinance` para conectar con una fuente de datos financieros y obtener la información histórica de una empresa.



## Acto II: El Arte de la Visualización

Con `matplotlib`, transformaremos esos datos en un gráfico claro, informativo y estéticamente pulcro, listo para ser compartido.

# Las Herramientas del Viaje: Nuestras Librerías Clave

```
import matplotlib.pyplot  
as plt
```

El estándar de oro para la visualización de datos en Python. Es nuestro pincel y nuestro lienzo para crear el gráfico.

```
import yfinance as yf
```

Una interfaz simple y potente para descargar datos históricos y metadatos de empresas directamente desde Yahoo Finance.

```
from io import BytesIO
```

Una herramienta avanzada que nos permite tratar la memoria RAM como un archivo. Crucial para la eficiencia en servidores.

# Píldora de Sabiduría: El Modo 'Sin Ventanas'



```
import matplotlib
# Configura el 'backend' a 'Agg'
matplotlib.use('Agg')
```

## El Problema

Por defecto, `matplotlib` intenta mostrar los gráficos en una ventana emergente. Esto falla en un servidor, que no tiene pantalla.

## La Solución

`matplotlib.use('Agg')` le indica que renderice el gráfico directamente a un archivo de imagen (como PNG) – sin intentar abrir ninguna interfaz gráfica.

## Conclusión

Esta línea es **esencial** para cualquier script que se ejecute de forma automática o en un entorno web.

# Acto I: La Búsqueda de los Datos

La función `obtener\_datos\_empresas`

```
def obtener_datos_empresas(empresa, periodo):
    """
    Descarga los datos OHLCV (Open, High, Low, Close, Volume)
    de Yahoo Finance.
    """
```

El primer paso es obtener los datos. Esta función actúa como nuestro agente de bolsa: le damos un ‘ticker’ (el símbolo de una empresa, como ‘AAPL’) y un período de tiempo ('1y' para un año), y nos devuelve toda la información que necesitamos.

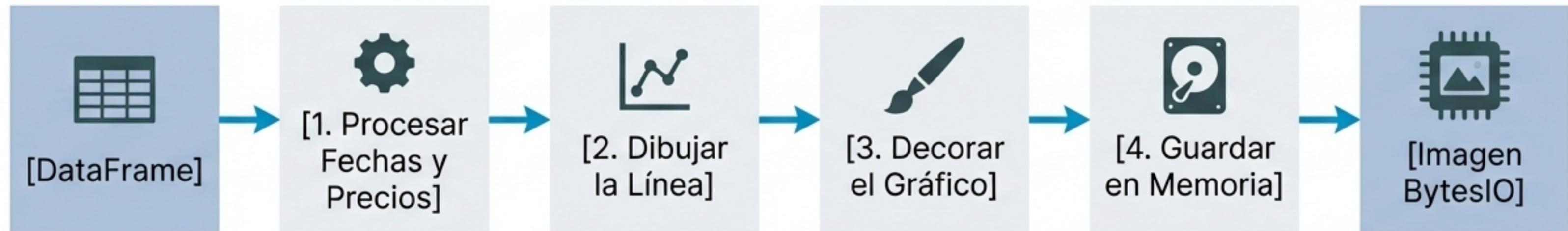
# Anatomía de la Búsqueda

```
def obtener_datos_empresas(empres, periodo):
    # Se llama a la API de yfinance.
    datos = yf.download(empres, period=periodo)
    # Retorna un DataFrame de Pandas.
    return datos
```

1. `yf.download(empresa, period=periodo)`  
Esta es la llamada clave. `yfinance` se encarga de toda la complejidad de la comunicación con la API de Yahoo Finance.
2. `datos = ...`  
El resultado se almacena en una variable llamada `datos`.
3. `return datos`  
La función devuelve un objeto `DataFrame` de la librería Pandas. Piensa en él como una hoja de cálculo superpotente, con fechas como índice y columnas para "Open", "Close", "High", etc.

# Acto II: Dando Vida a los Datos

## La función `generar\_grafico\_empresas`



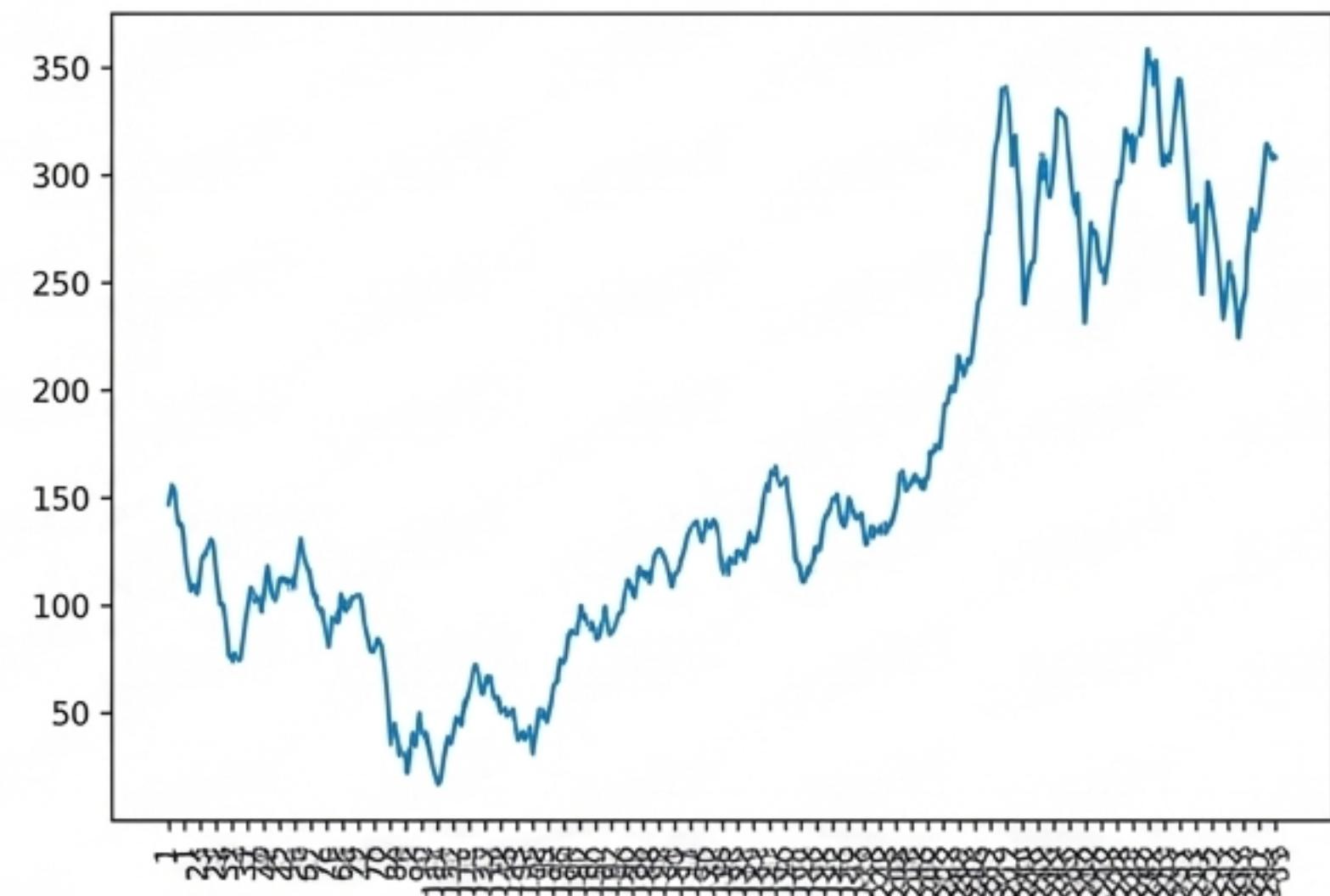
Ahora que tenemos los datos, el siguiente paso es convertirlos en una imagen. Esta función orquesta todo el proceso de visualización, desde la preparación de los datos hasta el guardado final de la imagen en memoria RAM.

# El Lienzo en Blanco y el Primer Trazo

```
# 1. Preparación de datos (explicado brevemente)
fechas = [...]
precios_cierre = [...]

# 2. Configuración del Lienzo
plt.figure(figsize=(10, 6.5))

# 3. Dibujo de la línea
plt.plot(fechas, precios_cierre,
          label='Precios de Cierre', color='b')
```



Con solo unas pocas líneas, `matplotlib` ya puede trazar nuestros datos. Ahora, vamos a refinarlo.

# Añadiendo Contexto: Títulos y Ejes

```
# Título principal en la parte superior  
plt.title(f'Evolución de {empresa}',  
          fontsize=16)  
  
# Etiqueta del eje horizontal  
plt.xlabel('Fecha', fontsize=14)  
  
# Etiqueta del eje vertical  
plt.ylabel(f'Precio ({moneda})', fontsize=14)
```



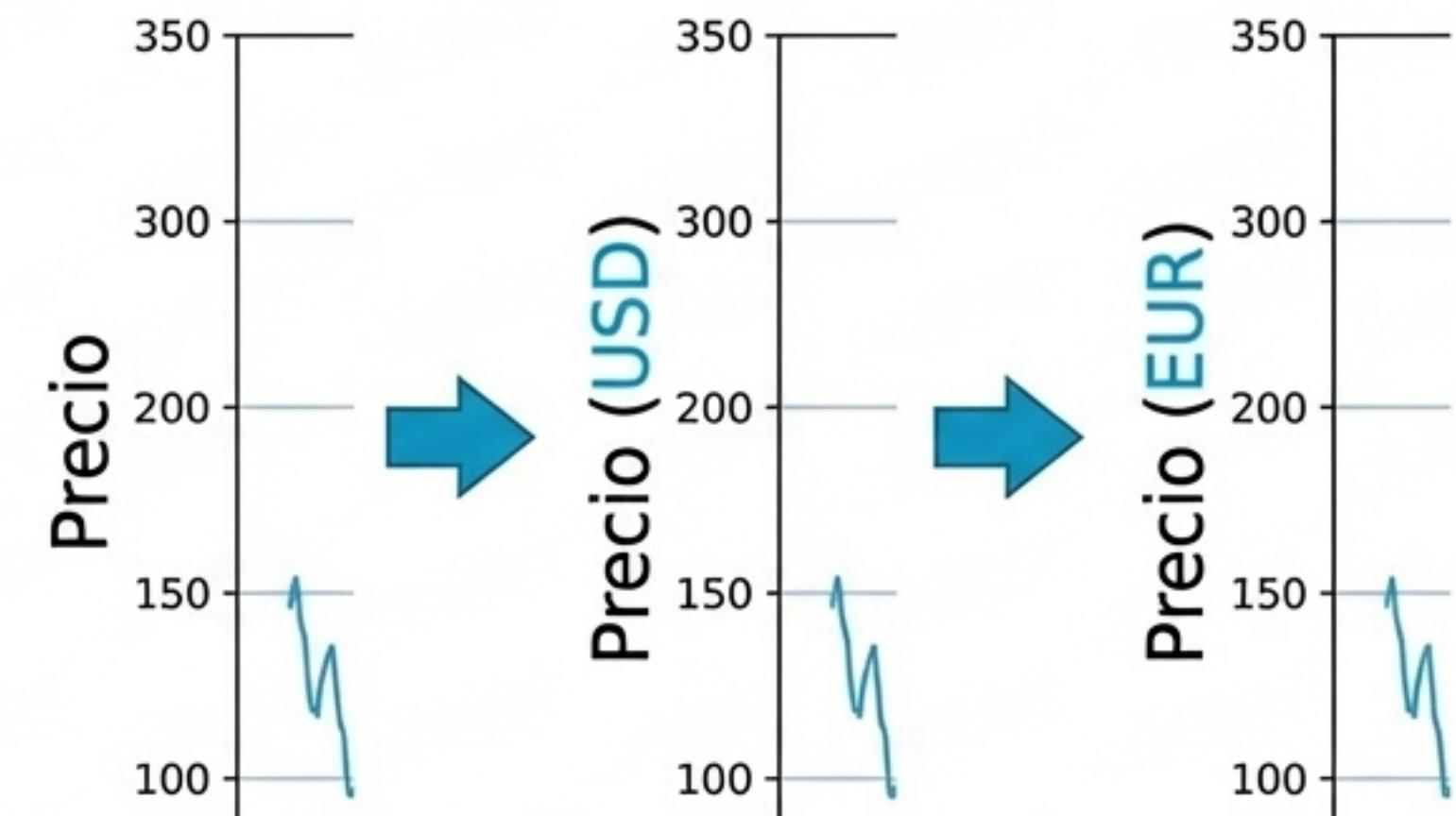
Las nuevas líneas de código añaden un título y etiquetas a los ejes, proporcionando contexto esencial. Los ticks del eje x aún necesitan ser ajustados.

# Un Detalle Profesional: La Moneda Dinámica

```
# Creamos un objeto Ticker para pedir metadatos
ticker_obj = yf.Ticker(empresia)

# Extraemos la moneda del diccionario .info
moneda = ticker_obj.info['currency']

# Se usa en la etiqueta del eje Y
# Se usa en la etiqueta del eje Y
plt.ylabel(f'Precio ({moneda})', fontsize=14)
```



- En lugar de asumir que el precio es en USD, hacemos una consulta adicional para obtener la moneda real de la empresa (USD, EUR, etc.).
- El objeto `yf.Ticker` nos da acceso a metadatos valiosos.
- Este pequeño paso hace que nuestro gráfico sea más preciso y adaptable a cualquier activo.

# Dominando el Eje X: Fechas Legibles

```
# Queremos mostrar aproximadamente 10 etiquetas
num_etiquetas = 10
step = max(1, len(fechas) // num_etiquetas)

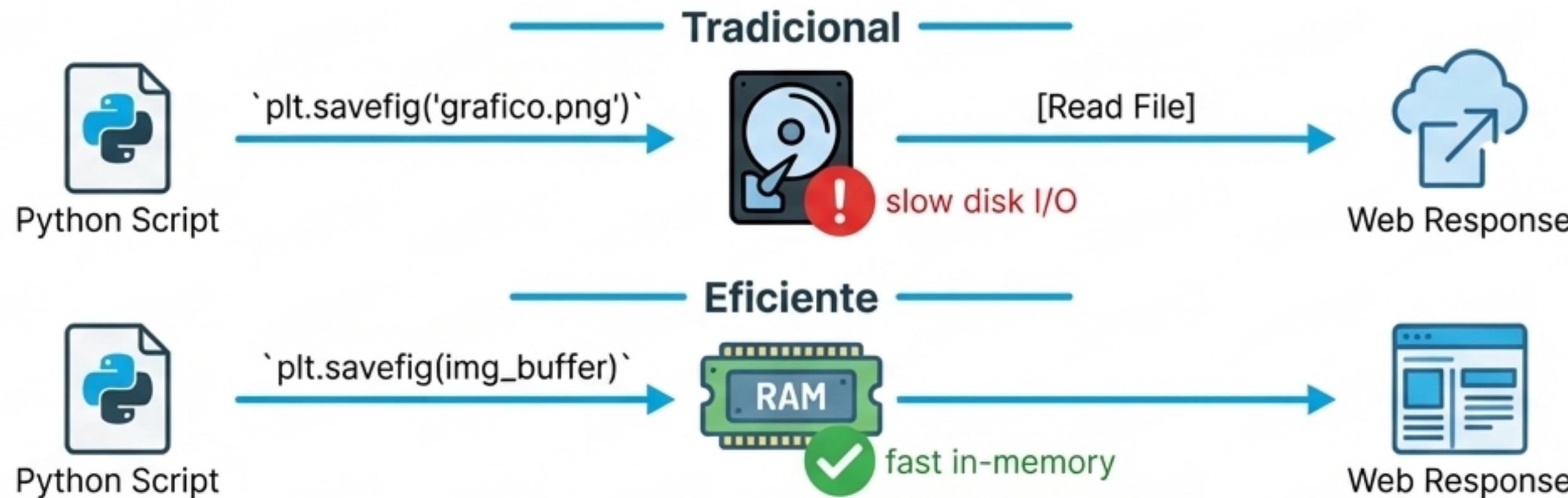
# Aplicamos la selección y rotamos las etiquetas
plt.xticks(fechas[::step], rotation=45, ha='right')
)

# Añadimos una cuadrícula y ajustamos márgenes
plt.grid(True)
plt.tight_layout()
```



# Píldora de Sabiduría: La Magia de la Memoria

Guardando el gráfico sin tocar el disco duro



```
from io import BytesIO  
  
# Crea un archivo virtual en la memoria RAM  
img_buffer = BytesIO()  
  
# Guarda el gráfico en ese buffer  
plt.savefig(img_buffer, format='png')
```

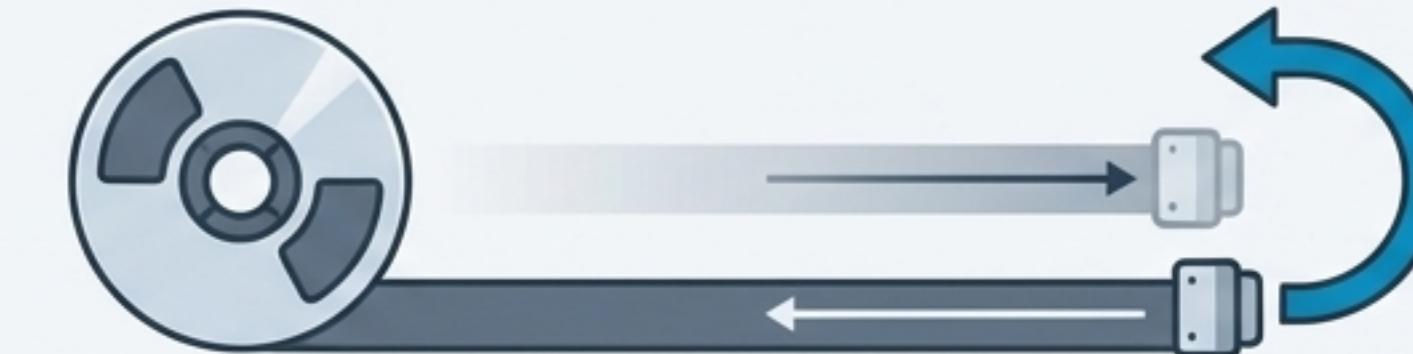
Guardar y leer desde el disco es lento. `BytesIO` crea un 'archivo' temporal directamente en la RAM, eliminando el cuello de botella del disco y haciendo el proceso mucho más rápido y limpio para servidores.

# El Acto Final: Limpieza y Eficiencia

Dos líneas de código que evitan fugas de memoria

```
img.seek(0)
```

```
img_buffer.seek(0)
```



Después de escribir en nuestro archivo en memoria, el 'cursor' está al final.  
`seek(0)` lo rebobina al principio, para que quien lo lea pueda ver la imagen completa desde el byte cero.

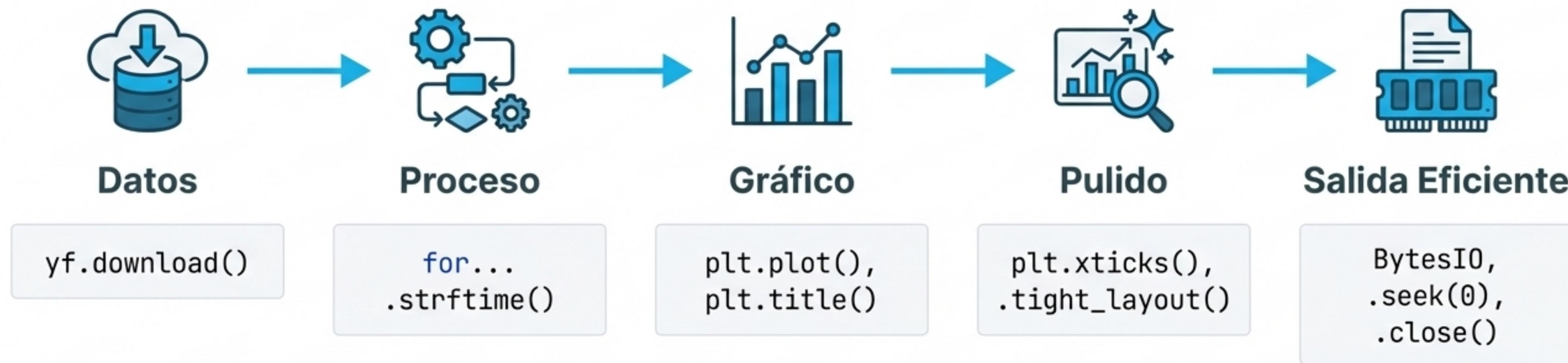
```
plt.close('all')
```

```
plt.close('all')
```



**¡CRÍTICO!** Cada vez que se crea un gráfico, `matplotlib` consume memoria. Si no cerramos explícitamente la figura, esta memoria nunca se libera. En un servidor que genera cientos de gráficos, esto provocaría un colapso por falta de RAM.

# El Viaje Completo: De Ticker a Imagen en Memoria



- Hemos construido un pipeline completo y automatizado.
- Hemos aprendido a obtener y procesar datos financieros de forma programática.
- Hemos dominado no solo cómo crear un gráfico, sino cómo hacerlo de manera profesional.
- Conocemos las mejores prácticas para entornos de servidor, asegurando rendimiento (`BytesIO`) y estabilidad (`plt.close`).