Módulo 0 (Sección 5): El Dolor y la Gloria de las Fechas

Filosofía: Antes de poder "muestrear", "agregar" o "interpolar" series temporales en Pandas, necesitas entender la naturaleza de un instante de tiempo. En Python, esto se gestiona con el módulo datetime. El 90% de los errores de datos están en el parsing de fechas o en la gestión de zonas horarias.

Parte 1: La Explicación Rigurosa

1. El Módulo datetime

Python proporciona varios objetos para manejar fechas y tiempo. Los cuatro fundamentales que usaremos son:

1. datetime.date: Almacena solo la fecha (año, mes, día).
2. datetime.time: Almacena solo la hora (hora, minuto, segundo, microsegundo).
3. datetime.datetime: El más usado. Almacena fecha y hora juntas.
4. datetime.timedelta: **Crucial**. Representa una *duración* o *diferencia* de tiempo (p.ej., "2 días y 5 horas").

2. El Concepto Clave: Fechas "Naive" vs. "Aware" (Ingenuas vs. Conscientes)

Este es el concepto más importante y el que más problemas causa.

- **Naive (Ingenua):** Es un objeto datetime que **NO** tiene información de zona horaria (Timezone). Es solo una etiqueta. "Las 10:00". ¿Dónde? ¿En Madrid? ¿En Londres? No se sabe.
    - datetime.now() y datetime.utcnow() (¡obsoleto!) devuelven datetimes *naive*.
- **Aware (Consciente):** Es un objeto datetime que **SÍ** tiene información de zona horaria. Sabe a qué parte del mundo se refiere. "Las 10:00 en Europa/Madrid (UTC+2)" es un instante de tiempo *inequívoco*.

Regla de Oro del Data Scientist: NUNCA operes con fechas *naive*. Siempre que ingestes datos, conviértelos a "aware", y trabaja **siempre** en UTC (Tiempo Universal Coordinado).

```
from datetime import datetime, timezone, timedelta
import json # Lo necesitaremos

# --- Naive (Ingenua) ---
dt_naive = datetime.now()
print(f"NAIVE: {dt_naive}")
# Imprime la hora local, pero sin 'tzinfo'. Es ambiguo.

# --- Aware (Consciente) - El método CORRECTO ---
```

```
# Usamos el objeto timezone.utc
dt_aware_utc = datetime.now(timezone.utc)
print(f"AWARE (UTC): {dt_aware_utc}")
# Nota el '+00:00'. Esto es un instante de tiempo inequívoco.
```

3. Los 4 Formatos (El "Casting" de Fechas)

Un instante de tiempo puede existir en 4 formatos. Tu trabajo es saber "castear" (convertir) de uno a otro.

1. **Objeto datetime:** El objeto nativo de Python. datetime(2025, 10, 26, 20, 15, 0, tzinfo=timezone.utc)
2. **Timestamp (Unix Epoch):** Un float o int. El número de segundos transcurridos desde el "Epoch" (1 de enero de 1970, 00:00:00 UTC). Es el formato preferido por las máquinas y bases de datos. 1761538500.0
3. **String (ISO 8601):** El estándar de oro para intercambiar fechas como texto. Es legible por humanos e inequívoco. 2025-10-26T20:15:00+00:00
4. **String (Custom/Sucia):** Cualquier otro formato. 26/10/2025 20:15 (formato europeo), 10/26/25 8:15 PM (formato americano). El 99% de tu trabajo de limpieza será esto.

4. El Taller: Convertir entre Formatos

```
from datetime import datetime, timezone

# Tomemos un instante de tiempo como referencia (ahora mismo, en UTC)
dt = datetime.now(timezone.utc)
print(f"Objeto Original (Aware): {dt}\n")

# --- 1. De Objeto a Timestamp (Unix) ---
ts = dt.timestamp()
print(f"Formato Timestamp (float): {ts}")

# --- 2. De Timestamp a Objeto ---
# Siempre devuelve un objeto aware en la zona horaria LOCAL.
# ¡CUIDADO! Es mejor convertirlo a UTC inmediatamente.
dt_from_ts_local = datetime.fromtimestamp(ts)
dt_from_ts_utc = datetime.fromtimestamp(ts, tz=timezone.utc)
print(f"De Timestamp a Objeto (UTC): {dt_from_ts_utc}\n")


# --- 3. De Objeto a String (ISO 8601) ---
iso_string = dt.isoformat()
print(f"Formato String (ISO 8601): {iso_string}")

# --- 4. De String (ISO 8601) a Objeto ---
```

```
dt_from_iso = datetime.fromisoformat(iso_string)
print(f"De ISO a Objeto: {dt_from_iso}\n")
# .fromisoformat() es inteligente y preserva el timezone.


# --- 5. De Objeto a String (Custom) - strftime ---
# str[F]ormat - "Formatear a String"
# %Y = Año 4 dígitos, %m = mes, %d = día, %H = hora (24h), %M = minuto, %S = segundo
custom_string = dt.strftime("%d/%m/%Y (%H:%M)")
print(f"Formato String (Custom): {custom_string}")

# --- 6. De String (Custom) a Objeto - strptime ---
# str[P]arse - "Parsear desde String"
# Debe coincidir EXACTAMENTE con el formato.
s = "26/10/2025 (20:15)"
formato = "%d/%m/%Y (%H:%M)"
dt_from_custom = datetime.strptime(s, formato)
print(f"De Custom a Objeto (Naive!): {dt_from_custom}\n")
# ¡OJO! strptime crea objetos NAIVE. Debes asignarles un timezone.
dt_aware = dt_from_custom.replace(tzinfo=timezone.utc)
print(f"De Custom a Objeto (Aware): {dt_aware}\n")
```

5. Aritmética de Fechas (timedelta)

No puedes sumar datetime + datetime. Puedes restar datetime - datetime (te da un timedelta) o sumar/restar datetime + timedelta.

```
from datetime import datetime, timedelta, timezone

ahora = datetime.now(timezone.utc)
print(f"Ahora: {ahora}")

# --- 1. Calcular fechas futuras/pasadas ---
delta = timedelta(days=5, hours=3, minutes=30)
en_5_dias = ahora + delta
print(f"Futuro: {en_5_dias}")

hace_1_semana = ahora - timedelta(weeks=1)
print(f"Pasado: {hace_1_semana}")


# --- 2. Calcular diferencias ---
t_inicio = datetime.now(timezone.utc)
```

```
# (simulamos un proceso largo)
for _ in range(100000):
    pass
t_fin = datetime.now(timezone.utc)

duracion = t_fin - t_inicio
print(f"\nTipo de 'duracion': {type(duracion)}")
print(f"Proceso tardó (objeto): {duracion}")
print(f"Proceso tardó (segundos): {duracion.total_seconds()}")
```

Parte 2: Desafíos (Módulo 0 + Fechas)

¡A combinarlo todo!

Desafío 1: Parsing de Logs de Formato Mixto

Escenario: Tienes una lista de logs de servidor. Algunos usan ISO 8601, otros usan un formato custom "legacy". Queremos una lista única de objetos datetime (aware, en UTC) ordenados cronológicamente.

```
from datetime import datetime, timezone

logs_sucios = [
    "2025-10-26T12:00:00+00:00 [INFO] Server Start",
    "27/10/2025 09:30:15 [WARN] Low memory", # Formato: dd/mm/YYYY HH:MM:SS
    "2025-10-26T12:01:30Z [INFO] User login", # 'Z' (Zulu) es UTC
    "28/10/2025 10:00:00 [ERROR] Crash"
]

# --- Solución ---

# 1. Crear una función de parsing "robusta"
def parsear_fecha_flexible(log_line):
    # Extraemos solo la parte de la fecha (antes del '[')
    timestamp_str = log_line.split(' [')[0]

    # Intento 1: Parsear formato ISO (el más rápido)
    try:
        # fromisoformat maneja 'Z' y '+00:00' automáticamente
        return datetime.fromisoformat(timestamp_str)
    except ValueError:
        pass # Si falla, probamos el siguiente

    # Intento 2: Parsear formato custom
```

```
    try:
        formato_custom = "%d/%m/%Y %H:%M:%S"
        dt_naive = datetime.strptime(timestamp_str, formato_custom)
        # Asumimos que los logs 'legacy' también estaban en UTC
        return dt_naive.replace(tzinfo=timezone.utc)
    except ValueError:
        pass # Si falla, no podemos hacer nada

    return None # Devolver None si todos fallan


print("--- Desafío 1: Parsing Flexible ---")
# 2. Usar la función en una list comprehension (y filtrar Nones)
datetimes_parseados = [
    dt for log in logs_sucios
    if (dt := parsear_fecha_flexible(log)) is not None
]

# 3. Ordenar la lista final
datetimes_ordenados = sorted(datetimes_parseados)

print("Logs parseados y ordenados:")
for dt in datetimes_ordenados:
    print(dt)
```

Desafío 2: Cálculo de Duración de Sesiones (JSON + Timestamp)

Escenario: Tienes un JSON con eventos de sesión. Debes correlacionar 'LOGIN' y 'LOGOUT' por session_id y calcular la duración media de las sesiones *válidas* (que tienen login y logout) en *minutos*.

```
import json
from datetime import datetime, timezone

json_eventos = """
[
    {"ts": 1678886400, "session": "SESS_A", "event": "LOGIN"},
    {"ts": 1678886460, "session": "SESS_B", "event": "LOGIN"},
    {"ts": 1678886520, "session": "SESS_A", "event": "LOGOUT"},
    {"ts": 1678886580, "session": "SESS_C", "event": "LOGIN"},
    {"ts": 1678886700, "session": "SESS_B", "event": "LOGOUT"},
    {"ts": 1678886800, "session": "SESS_D", "event": "LOGIN"}
]
"""
```

```
# SESS_C y SESS_D son huérfanas

print("\n--- Desafío 2: Duración de Sesiones ---")

# --- Solución ---
eventos = json.loads(json_eventos)

# 1. Separar logins y logouts en dicts para búsqueda O(1)
#    Clave = session_id, Valor = objeto datetime (desde timestamp)
logins = {
    e['session']: datetime.fromtimestamp(e['ts'], tz=timezone.utc)
    for e in eventos if e['event'] == 'LOGIN'
}

logouts = {
    e['session']: datetime.fromtimestamp(e['ts'], tz=timezone.utc)
    for e in eventos if e['event'] == 'LOGOUT'
}

# 2. Encontrar sesiones comunes (intersección de claves)
sesiones_completas = logins.keys() & logouts.keys()
print(f"Sesiones completas encontradas: {sesiones_completas}")

# 3. Calcular duraciones en minutos (usando dict comprehension)
duraciones_sesiones = {
    sess_id: (logouts[sess_id] - logins[sess_id]).total_seconds() / 60
    for sess_id in sesiones_completas
}

print(f"Duraciones (minutos): {duraciones_sesiones}")

# 4. Calcular la media
media_minutos = sum(duraciones_sesiones.values()) / len(duraciones_sesiones)
print(f"Duración media: {media_minutos:.2f} minutos")
```

Desafío 3: "GroupBy" Manual por Día (Agregación de TimeSeries)

Escenario: Tienes una larga lista de lecturas de un sensor. Queremos agregar esto para obtener la temperatura *máxima* y *media* por *día*. (Esto es un resample('D').agg(['mean', 'max']) en Pandas).

```
from datetime import datetime, timezone

# (Datos de ejemplo, en un caso real serían miles)
```

```
lecturas = [
    {'ts': 1678886400, 'temp': 12.0}, # 2023-03-15 13:20:00
    {'ts': 1678890000, 'temp': 13.0}, # 2023-03-15 14:20:00
    {'ts': 1678971600, 'temp': 15.0}, # 2023-03-16 13:00:00
    {'ts': 1678975200, 'temp': 18.0}, # 2023-03-16 14:00:00
    {'ts': 1678975800, 'temp': 16.5}, # 2023-03-16 14:10:00
    {'ts': 1678886460, 'temp': 12.5}  # 2023-03-15 13:21:00
]

print("\n--- Desafío 3: 'GroupBy' por Día ---")

# 1. Convertir todos los datos, añadiendo el objeto 'date'
#    Usamos .date() para extraer solo la parte de la fecha
datos_procesados = [
    {
        'dt': datetime.fromtimestamp(l['ts'], tz=timezone.utc),
        'fecha': datetime.fromtimestamp(l['ts'], tz=timezone.utc).date(),
        'temp': l['temp']
    }
    for l in lecturas
]

# 2. Encontrar los días únicos (set comprehension)
dias_unicos = {d['fecha'] for d in datos_procesados}
print(f"Días encontrados: {dias_unicos}")

# 3. Agregar usando una dict comprehension (anidada)
agregado_por_dia = {
    dia: {
        'temp_media': sum(d['temp'] for d in datos_procesados if d['fecha'] == dia) /
                      sum(1 for d in datos_procesados if d['fecha'] == dia),
        'temp_max': max(d['temp'] for d in datos_procesados if d['fecha'] == dia)
    }
    for dia in dias_unicos
}

print("Datos agregados por día:")
print(json.dumps(agregado_por_dia, indent=2, default=str)) # default=str para serializar 'date'
```

Desafío 4: Lambdas y Ordenación por Fechas "Sucias"

Escenario: Tienes una lista de artículos (diccionarios). La clave 'publicado' es un string en formato dd-mm-YYYY. Quieres ordenar la lista por fecha de publicación, de más reciente a

más antiguo.

```
from datetime import datetime

articulos = [
    {'titulo': 'Articulo A', 'publicado': '15-06-2024'},
    {'titulo': 'Articulo C', 'publicado': '01-01-2025'},
    {'titulo': 'Articulo B', 'publicado': '30-11-2023'}
]

print("\n--- Desafío 4: Ordenación con Lambda + strptime ---")

# --- Solución ---
# No podemos ordenar alfabéticamente ("30-11-2023" iría después de "15-06-2024")
# Debemos convertir el string a un objeto datetime *dentro* de la 'key'
articulos_ordenados = sorted(
    articulos,
    key=lambda art: datetime.strptime(art['publicado'], "%d-%m-%Y"),
    reverse=True # De más reciente a más antiguo
)

print(json.dumps(articulos_ordenados, indent=2))
```