CÓDIGO REALIZADO DURANTE LA PRÁCTICA

```
# ================================================
#  Coneccion a la base de datos
# ================================================

import psycopg2
import pandas as pd

# Función para ejecutar una consulta y cargar los resultados en un DataFrame
def execute_query(query):
    try:
        # Establecer la conexión
        conn = psycopg2.connect(
            dbname='LAR_TablasStock',
            user='postgres',
            password='123',
            host='localhost',
            port='5432'
        )

        # Ejecuto la consulta y cargo los datos en un DataFrame
        df = pd.read_sql_query(query, conn)
        conn.close()
        return df
    except Exception as e:
        print(f"Error al conectarse a la base de datos: {e}")



# ================================================
#  Consulta creada para obtener el total de los artículos en oferta
# ================================================

query_consulta_final = """
WITH ofertas AS (
    SELECT
        a.codigo,
        lo.barras,
        a.descripcion,
        lo.fecha_inicio,
        lo.fecha_fin,
        0.0 AS porcentaje_descuento
    FROM
        "DBA".listas_oferta lo
    JOIN
        "DBA".articulos a ON a.barras = lo.barras
```

```
    WHERE
        lo.fecha_inicio >= '2024-08-01'
        AND lo.fecha_fin <= '2024-08-31'
),
precio_minorista AS (
    SELECT
        barras,
        fecha_vigencia_desde,
        fecha_vigencia_hasta,
        precio_minorista
    FROM
        "DBA"."Precios_y_costos"
    WHERE
        fecha_vigencia_desde <= '2024-08-31'
        AND fecha_vigencia_hasta >= '2024-08-01'
)

SELECT
    o.codigo,
    o.barras,
    o.descripcion,
    o.fecha_inicio,
    o.fecha_fin,
    CASE
        WHEN p.precio_minorista IS NULL OR lo.precio IS NULL THEN NULL
        ELSE ROUND(((p.precio_minorista - lo.precio) / p.precio_minorista) * 100, 2)
    END AS porcentaje_descuento,
    COALESCE(ss.sucursal, '') AS sucursal,
    ROUND(COALESCE(ss.existencia, 0)) AS stock
FROM
    ofertas o
LEFT JOIN
    "DBA".stock_sucursal ss ON o.codigo = ss.codigo_art AND ss.fecha = o.fecha_inicio - 1
LEFT JOIN
    precio_minorista p ON o.barras = p.barras AND o.fecha_inicio - 1 BETWEEN
p.fecha_vigencia_desde AND p.fecha_vigencia_hasta
LEFT JOIN
    "DBA".listas_oferta lo ON o.barras = lo.barras AND o.fecha_inicio = lo.fecha_inicio
ORDER BY
    o.descripcion;

"""

# Ejecuto la consulta combinada
result_final = execute_query(query_consulta_final)

# Muestro el número de filas
print(f"Número de filas: {len(result_final)}")
```

```
# Exporto los resultados a un archivo csv
result_final.to_csv('Articulos_oferta.csv', index=False)


# ========================================================
#  Consulta para obtener la cantidad de articulos diferentes
# ========================================================

cant_articulos  = """
WITH ofertas AS (
    SELECT
        a.codigo,
        lo.barras,
        a.descripcion,
        lo.fecha_inicio,
        lo.fecha_fin,
        0.0 AS porcentaje_descuento
    FROM
        "DBA".listas_oferta lo
    JOIN
        "DBA".articulos a ON a.barras = lo.barras
    WHERE
        lo.fecha_inicio >= '2024-08-01'
        AND lo.fecha_fin <= '2024-08-31'
)

SELECT
    COUNT(DISTINCT o.barras) AS total_articulos_oferta
FROM
    ofertas o;
"""


# ================================================
#  Cargar los datos de stock y articulos oferta
# ================================================

# Cargo los archivos CSV
articulos_of = pd.read_csv("Articulos_oferta.csv")


# ================================================================
#  Cambiar nombres de sucursal para que coincidan con el csv de venta
# ================================================================

# Cambios de nombres
cambios_nombres = {
```

```
    'CRESPO-CCC': 'Crespo',
    'HERNANDEZ-SMINO': 'Hernandez',
    'NOGOYA SUR-CONSUMO': 'Nogoya Sur',
    'NOGOYA ITALIA-SMINO': 'Nogoya Italia'
}

# Cambio los nombres en la columna 'sucursal' de articulos_of
if 'sucursal' in articulos_of.columns:
    articulos_of['sucursal'] = articulos_of['sucursal'].replace(cambios_nombres)

print("Nombres de sucursales actualizados correctamente.")




# ================================================
#  Consulta creada para obtener las ventas de los artículos de oferta.     CORREGIR
# ================================================

query_ventas = """

WITH ventas_filtradas AS (
    SELECT
        a.codigo,
        a.descripcion AS Descripcion_Articulo_Oferta,
        a.sucursal,
        a.fecha_inicio,
        a.fecha_fin,
        COALESCE(SUM(v."total_ventas_unidad"), 0) AS Total_Ventas_Unidad,
        a.stock
    FROM
        "DBA"."articulos_of" a
    INNER JOIN
        "DBA"."ventas" v
        ON a.codigo = v.codigointerno AND a.sucursal = v.sucursal
    WHERE
        v.fecha >= a.fecha_inicio AND v.fecha <= a.fecha_fin
    GROUP BY
        a.codigo,
        a.descripcion,
        a.sucursal,
        a.fecha_inicio,
        a.fecha_fin,
        a.stock
)
SELECT *
FROM ventas_filtradas
ORDER BY Descripcion_Articulo_Oferta, sucursal;
```

"""

```
# ============================================================
#  Código en python creado para obtener las ventas de los artículos de oferta.
# ============================================================

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# ================================================
#  Carga de datos
# ================================================

# Cargo los datos desde los archivos CSV
Ventas = pd.read_csv("Ventas_Combinadas_Agosto.csv")
articulos_oferta = pd.read_csv("Articulos_oferta.csv")

# Convierto las columnas 'CodigoInterno' y 'codigo' a string para evitar inconsistencias
Ventas['CodigoInterno'] = Ventas['CodigoInterno'].astype(str)
articulos_oferta['codigo'] = articulos_oferta['codigo'].apply(
    lambda x: str(int(float(x))) if isinstance(x, (int, float)) else str(x)
)

# Renombro columna 'DescripcionMarca' a 'Marca_producto' en el DataFrame de Ventas
Ventas.rename(columns={'DescripcionMarca': 'Marca_producto'}, inplace=True)

# Convierto las fechas a formato datetime
Ventas['Fecha'] = pd.to_datetime(Ventas['Fecha'])
articulos_oferta['fecha_inicio'] = pd.to_datetime(articulos_oferta['fecha_inicio'])
articulos_oferta['fecha_fin'] = pd.to_datetime(articulos_oferta['fecha_fin'])

# ================================================
#  Combinación y filtrado de datos
# ================================================

# Realizo el merge entre las ventas y los artículos en oferta
merged_df = pd.merge(
    articulos_oferta,
    Ventas,
    left_on=["codigo", "sucursal"],
    right_on=["CodigoInterno", "Sucursal"],
    how="inner"  # Inner join para mantener correspondencias en ambas tablas
)

# Relleno valores NaN en columnas relevantes
```

```
merged_df['Total de Ventas por Unidad'].fillna(0, inplace=True)  # Asigno 0 a ventas sin registros
merged_df['Descripcion'] = merged_df['Descripcion'].fillna('Sin descripción')
merged_df['Marca_producto'] = merged_df['Marca_producto'].fillna('Sin marca')

# Filtro las ventas que están dentro del rango de fechas de la oferta
filtered_df = merged_df[
    (merged_df['Fecha'] >= merged_df['fecha_inicio']) &
    (merged_df['Fecha'] <= merged_df['fecha_fin'])
]

# ================================================
#  Agrupamiento y agregación
# ================================================

# Agrupo por producto, sucursal y período de oferta
result_df = filtered_df.groupby(
    ['codigo', 'Descripcion', 'Marca_producto', 'sucursal', 'fecha_inicio', 'fecha_fin', 'stock'],
    as_index=False
).agg({'Total de Ventas por Unidad': 'sum'})

# Renombro la columna para claridad
result_df.rename(
    columns={
        'Descripcion': 'Descripcion_Articulo_Oferta',
        'Total de Ventas por Unidad': 'Total_Ventas_Unidad'
    }, inplace=True
)

# Me aseguro de que el total de ventas sea un entero
result_df['Total_Ventas_Unidad'] = result_df['Total_Ventas_Unidad'].astype(int)

# Guardo el DataFrame resultante en un archivo CSV
result_df.sort_values(by='Descripcion_Articulo_Oferta', inplace=True)
result_df.to_csv("resultado_ventas_oferta.csv", index=False, encoding='utf-8')


# ================================================
#  Lectura del archivo generado
# ================================================

# Leo el archivo guardado para graficar las ventas por período del mes
ventas_oferta = pd.read_csv("resultado_ventas_oferta.csv")
ventas_oferta['fecha_inicio'] = pd.to_datetime(ventas_oferta['fecha_inicio'])


# ================================================
#  Gráfico de barras: Ventas totales por período
```

```
# ================================================

# Filtro las ventas de agosto
result_df['fecha_inicio'] = pd.to_datetime(result_df['fecha_inicio'])
result_df = result_df[result_df['fecha_inicio'].dt.month == 8]

# Creo una nueva columna para clasificar los períodos
def clasificar_periodo_fecha(date):
    if date.day <= 10:
        return 'Inicio del mes\n01-10'
    elif 11 <= date.day <= 20:
        return 'Mitad del mes\n11-20'
    else:
        return 'Fin del mes\n21-31'

result_df['Periodo'] = result_df['fecha_inicio'].apply(clasificar_periodo_fecha)

# Agrupo por período y calculo la suma de ventas
ventas_por_periodo =
result_df.groupby('Periodo')['Total_Ventas_Unidad'].sum().reset_index()

# Ordeno los períodos lógicamente
ventas_por_periodo['Periodo'] = pd.Categorical(
    ventas_por_periodo['Periodo'],
    categories=['Inicio del mes\n01-10', 'Mitad del mes\n11-20', 'Fin del mes\n21-31'],
    ordered=True
)
ventas_por_periodo.sort_values('Periodo', inplace=True)

# Gráfico de barras con fechas detalladas
plt.figure(figsize=(12, 6))
plt.bar(ventas_por_periodo['Periodo'], ventas_por_periodo['Total_Ventas_Unidad'],
color=['skyblue', 'lightgreen', 'lightcoral'])
plt.title('Ventas Totales en los Diferentes Períodos de Agosto')
plt.xlabel('Período del Mes')
plt.ylabel('Total de Ventas por Unidad')
plt.xticks(rotation=45, ha='right')  # Roto etiquetas para mejor visualización
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.show()


# ================================================
#  Calcular la tasa de ventas en oferta por sucursal
# ================================================

# Ventas totales de agosto por sucursal
ventas_totales = Ventas.groupby('Sucursal')['Total de Ventas por
Unidad'].sum().reset_index()
```

```
ventas_totales.rename(columns={'Total de Ventas por Unidad': 'Ventas_Totales'},
inplace=True)

# Ventas en oferta por sucursal
ventas_oferta_sucursal =
ventas_oferta.groupby('sucursal')['Total_Ventas_Unidad'].sum().reset_index()
ventas_oferta_sucursal.rename(columns={'Total_Ventas_Unidad': 'Ventas_Oferta'},
inplace=True)

# Combinar ambos DataFrames
tasa_ventas = pd.merge(ventas_oferta_sucursal, ventas_totales, left_on='sucursal',
right_on='Sucursal', how='inner')

# Calcular la tasa de ventas en oferta
tasa_ventas['Tasa_Ventas_Oferta'] = tasa_ventas['Ventas_Oferta'] /
tasa_ventas['Ventas_Totales']

# ================================================
#  Gráfico: Tasa de ventas en oferta por sucursal
# ================================================

# Configuración del gráfico de barras comparativo
plt.figure(figsize=(10, 6))
sns.barplot(
    data=tasa_ventas,
    x='Sucursal',
    y='Tasa_Ventas_Oferta',
    palette='viridis'
)
plt.title('Tasa de Ventas en Oferta por Sucursal (Agosto)', fontsize=14)
plt.xlabel('Sucursal', fontsize=12)
plt.ylabel('Tasa de Ventas en Oferta', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.show()

crespo_oferta = ventas_oferta[ventas_oferta['sucursal'] ==
'Crespo']['Total_Ventas_Unidad'].sum()
crespo_totales = Ventas[Ventas['Sucursal'] == 'Crespo']['Total de Ventas por Unidad'].sum()
tasa_crespo = crespo_oferta / crespo_totales
print(crespo_oferta)
print(crespo_totales)
print(tasa_crespo)

# ======================================================
#  Consulta creada para graficar las ventas en oferta por sucursal
# ======================================================
```

```
ventas_sucursal = """
SELECT
    rvo.sucursal AS Sucursal,
    ROUND(
        SUM(rvo.Total_Ventas_Unidad) / vt.Ventas_Totales,
        4
    ) AS Tasa_Ventas_Oferta
FROM
    "DBA".resultado_ventas_oferta rvo
INNER JOIN (
    SELECT
        Sucursal,
        SUM(total_ventas_unidad) AS Ventas_Totales
    FROM
        "DBA".ventas
    GROUP BY
        Sucursal
) vt ON rvo.sucursal = vt.Sucursal
GROUP BY
    rvo.sucursal, vt.Ventas_Totales
ORDER BY
    Tasa_Ventas_Oferta DESC;
"""


# ================================================
#  Gráfico: Top 5 productos más vendidos
# ================================================

# Agrupo por producto y sumo las ventas totales para obtener los 5 productos más vendidos
top_5_productos = result_df.groupby(['codigo', 'Descripcion_Articulo_Oferta'])[
    'Total_Ventas_Unidad'
].sum().nlargest(5).reset_index()

# Creo el gráfico de barras
plt.figure(figsize=(12, 6))
sns.barplot(
    data=top_5_productos,
    x='Descripcion_Articulo_Oferta',
    y='Total_Ventas_Unidad',
    palette='viridis'
)
plt.title('Top 5 Productos Más Vendidos')
plt.xlabel('Descripción del Artículo')
plt.ylabel('Total de Ventas por Unidad')
plt.xticks(rotation=45, ha='right')  # Roto etiquetas para mejor visualización
plt.grid(axis='y', linestyle='--', alpha=0.5)
```

```
plt.tight_layout()
plt.show()


# ========================================================
#  Código en python creado para obtener el aumento de ventas respecto a promedio de ventas de días de comercialización normal
# ========================================================
```

$$\text{aumento\_ventas} = \left(\frac{\text{Total Ventas en la Oferta} - \text{Total Ventas Normales}}{\text{Total Ventas Normales}} \right) \times 100$$

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# ================================================
#  Cargar los datos
# ================================================

# Cargo ventas_oferta y Ventas
ventas_oferta = pd.read_csv("resultado_ventas_oferta.csv")
Ventas = pd.read_csv("Ventas_Combinadas_Agosto.csv")

# Convierto las fechas a formato datetime
Ventas['Fecha'] = pd.to_datetime(Ventas['Fecha'])
ventas_oferta['fecha_inicio'] = pd.to_datetime(ventas_oferta['fecha_inicio'])
ventas_oferta['fecha_fin'] = pd.to_datetime(ventas_oferta['fecha_fin'])

# ================================================
#  Calcular ventas normales y aumento de ventas
# ================================================

def calcular_ventas_normales(row):
    """
    Calcula las ventas normales y el aumento de ventas para un producto en una sucursal y periodo dados.
    Compara al menos 5 días si es posible. Si hay menos días, considera los dias disponibles.
    """
    codigo = row['codigo']
    sucursal = row['sucursal']
    fecha_inicio_oferta = row['fecha_inicio']
    fecha_fin_oferta = row['fecha_fin']
```

```
    # Filtro ventas normales (fuera del rango de la oferta)
    ventas_normales = Ventas[
        (Ventas['CodigoInterno'] == str(codigo)) &
        (Ventas['Sucursal'] == sucursal) &
        ((Ventas['Fecha'] < fecha_inicio_oferta) | (Ventas['Fecha'] > fecha_fin_oferta)) &
        (Ventas['Fecha'].dt.year == 2024)
    ]

    # Calculo el número de días en la oferta
    dias_oferta = (fecha_fin_oferta - fecha_inicio_oferta).days + 1

    # Determino el número mínimo de días para comparar
    dias_comparacion = min(dias_oferta, 5)

    # Selecciono las fechas de ventas normales
    fechas_normales = ventas_normales['Fecha'].sort_values().unique()[:dias_comparacion]

    # Verifico si hay suficientes fechas normales
    if len(fechas_normales) < dias_comparacion:
        dias_comparacion = len(fechas_normales)

    # Filtro las ventas normales con las fechas seleccionadas
    ventas_normales = ventas_normales[ventas_normales['Fecha'].isin(fechas_normales)]

    # Calculo la suma total de ventas normales
    total_ventas_normal = ventas_normales['Total de Ventas por Unidad'].sum()

    # Calculo el aumento de ventas
    aumento = ((row['Total_Ventas_Unidad'] - total_ventas_normal) / total_ventas_normal *
100
            if total_ventas_normal > 0 else np.nan)

    return pd.Series({
        'cantidad_ventas_normal': total_ventas_normal,
        'fechas_normal': ', '.join([str(fecha.date()) for fecha in fechas_normales]),
        'aumento_ventas': round(aumento, 2),
        'cantidad_dias': dias_comparacion
    })

# Aplico la función y agrego las columnas al dataset existente
ventas_normales_info = ventas_oferta.apply(calcular_ventas_normales, axis=1)
comparacion_ventas = pd.concat([ventas_oferta, ventas_normales_info], axis=1)

# ================================================
#  Guardo el dataset final
# ================================================

# Guardo el DataFrame final
```

```
comparacion_ventas.to_csv("comparacion_ventas.csv", index=False, encoding='utf-8')
print("Archivo comparacion_ventas.csv generado con éxito.")

# ================================================
#  Verificacion Ventas_combinadas
# ================================================

# Cargo los datos desde el archivo CSV
ventas_oferta = pd.read_csv("resultado_ventas_oferta.csv")

# Cargo los datos desde los archivos CSV
Ventas = pd.read_csv("Ventas_Combinadas_Agosto.csv")

# Filtro los datos según los criterios especificados
filtro = (Ventas['Fecha'] == '2024-08-01') & (Ventas['CodigoInterno'] == '9038') &
(Ventas['Sucursal'] == 'Crespo')
Ventas_filtradas = Ventas[filtro]

# Calculo el total de ventas por unidad
total_ventas_unidad = Ventas_filtradas['Total de Ventas por Unidad'].sum()

print("El Total de Ventas por Unidad es:", total_ventas_unidad)

# ================================================
#  Estadísticas y visualización
# ================================================

# Redondeo los valores de aumento_ventas a 2 decimales
comparacion_ventas['aumento_ventas'] = comparacion_ventas['aumento_ventas'].round(2)

# Cuento las categorías de aumento de ventas
aumentos_positivos = comparacion_ventas[comparacion_ventas['aumento_ventas'] >
0].shape[0]
disminuciones_ventas = comparacion_ventas[comparacion_ventas['aumento_ventas'] <
0].shape[0]
aumentos_sin_dato = comparacion_ventas['aumento_ventas'].isna().sum()
aumento_en_0 = comparacion_ventas[comparacion_ventas['aumento_ventas'] ==
0].shape[0]


# Creo el DataFrame para el gráfico
categorias = pd.DataFrame({
    'Categoria': ['Aumento > 0', 'Aumento < 0', 'Sin Dato', 'Aumento = 0'],
    'Cantidad': [aumentos_positivos, disminuciones_ventas, aumentos_sin_dato,
aumento_en_0]
})

# Grafico
```

```
plt.figure(figsize=(8, 6))
sns.barplot(data=categorias, x='Cantidad', y='Categoria', palette='viridis')
plt.title('Categorías del Aumento de Ventas', fontsize=16)
plt.xlabel('Cantidad')
plt.ylabel('Categoría')
plt.show()


# ================================================
#  Graficar aumento de ventas por sucursal
# ================================================

# Agrupo por sucursal y calculo el promedio del aumento de ventas
aumento_por_sucursal =
comparacion_ventas.groupby('sucursal')['aumento_ventas'].mean().reset_index()

# Redondo los valores a 2 decimales
aumento_por_sucursal['aumento_ventas'] =
aumento_por_sucursal['aumento_ventas'].round(2)

# Ordeno por aumento de ventas
aumento_por_sucursal = aumento_por_sucursal.sort_values(by='aumento_ventas',
ascending=False)

# Creo el gráfico de barras
plt.figure(figsize=(12, 8))
sns.barplot(
    data=aumento_por_sucursal,
    x='aumento_ventas',
    y='sucursal',
    palette='coolwarm'
)

# Configuro título y etiquetas
plt.title('Aumento Promedio de Ventas por Sucursal (%)', fontsize=16)
plt.xlabel('Aumento Promedio en Ventas (%)', fontsize=12)
plt.ylabel('Sucursal', fontsize=12)

plt.show()

# ========================================================
#  Código en python creado para obtener el porcentaje del stock vendido
# ========================================================
```

$$Porcentaje\_Vendido = \left(\frac{Total\_Ventas\_Oferta}{Stock\_Disponible} \right) \times 100$$

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# ======================================================
# Cargar datos y manejar errores
# ======================================================
ventas_oferta = pd.read_csv("resultado_ventas_oferta.csv")

# ======================================================
# Calcular el porcentaje de stock vendido
# ======================================================
ventas_oferta['Porcentaje_Vendido'] = ventas_oferta.apply(
    lambda row: round((row['Total_Ventas_Unidad'] / row['stock']) * 100, 2) if row['stock'] > 0
else 0,
    axis=1
)

# Guardo resultados
ventas_oferta.to_csv("resultado_stock_vendido.csv", index=False, encoding='utf-8')
print("Cálculo de porcentaje vendido completado y guardado en
'resultado_stock_vendido.csv'.")


# ======================================================
# Calculo días de la oferta
# ======================================================
ventas_oferta['fecha_inicio'] = pd.to_datetime(ventas_oferta['fecha_inicio'], errors='coerce')
ventas_oferta['fecha_fin'] = pd.to_datetime(ventas_oferta['fecha_fin'], errors='coerce')
ventas_oferta['Dias_Oferta'] = (ventas_oferta['fecha_fin'] -
ventas_oferta['fecha_inicio']).dt.days + 1

# ============================================================
#Query para el grafico en Metabase de stock_sucursal vendido
# ============================================================

stock_sucursal = """

SELECT
    sucursal,
    CASE
        WHEN SUM(stock) > 0 THEN ROUND((SUM(Total_Ventas_Unidad) / SUM(stock)) *
100, 2)
        ELSE 0
    END AS porcentaje_vendido
FROM
    "DBA".resultado_ventas_oferta
```

```
GROUP BY
    sucursal
ORDER BY
    porcentaje_vendido DESC;

"""

# ============================================================
#  Código en python creado para obtener la efectividad de la oferta por dia
# ============================================================
```

$$\text{Efectividad\_Ventas\_por\_Día} = \frac{\sum \text{Total\_Ventas\_Unidad}}{\sum \text{Dias\_Oferta}}$$

```
# ========================================================
# Cálculo de efectividad de las ventas por dia
# ========================================================
efectividad_por_periodo = ventas_oferta.groupby(
    ['codigo', 'sucursal', 'fecha_inicio', 'fecha_fin']
).apply(
    lambda df: df['Total_Ventas_Unidad'].sum() / df['Dias_Oferta'].sum() if
df['Dias_Oferta'].sum() > 0 else 0
).reset_index(name='Efectividad_Ventas_por_Dia')

efectividad_por_periodo['Efectividad_Ventas_por_Dia'] =
efectividad_por_periodo['Efectividad_Ventas_por_Dia'].round(2)

ventas_oferta = ventas_oferta.merge(
    efectividad_por_periodo,
    on=['codigo', 'sucursal', 'fecha_inicio', 'fecha_fin'],
    how='left'
)

# ============================================================
# Elimino la columna Porcentaje_Vendido que no debe aparecer
# ============================================================

ventas_oferta = ventas_oferta.drop(columns=['Porcentaje_Vendido'])

# ============================================================
# Guardo el archivo
# ============================================================

ventas_oferta.to_csv("resultado_efectividad_por_dia.csv", index=False, encoding='utf-8')
print("Archivo 'resultado_efectividad_por_dia.csv' generado exitosamente.")
```