Proyecto Final: Ingeniería de Datos en Python

Luis Tun

15 de abril de 2024

Creación de la Base de Datos.

Se importan las librerías necesarias

Creación de la Base de Datos en AWS.

```
import boto3
import pandas as pd
import numpy as np
import psycopg2
import configparser
import sql_queries
import binascii
from faker import Faker
import random
import os
```

Creación de la Base de Datos.

Se lee el archivo de configuración. Este archivo no se incluye en el repositorio

Creación de la Base de Datos en AWS.

```
config = configparser.ConfigParser()
```

Creación de la Base de Datos.

Nos identificamos con AWS.

Creación de la base de Datos en AWS.

```
aws_rds_connection = boto3.client('rds',
aws_access_key_id=config.get('IAM', 'ACCESS_KEY'),
                  'SECRET_ACCESS_KEY'),
                   region_name='us-east-2')
```

Creación de la Base de Datos.

Se verifica instancias de AWS disponibles en el usuario.

Creación de la base de Datos en AWS.

```
rds_instances_ids = []

aws_response = aws_rds_connection.describe_db_instances()

for response in aws_response['DBInstances']:
    rds_instances_ids.append(response['DBInstanceIdentifier'])

print(f"Instancias Disponibles: {rds_instances_ids}")
```

Creación de la Base de Datos.

Se crea una instancia de Base de Datos en AWS con PostGres.

Creación de la base de Datos en AWS.

```
try:
  response = aws_rds_connection.create_db_instance(
    DBInstanceIdentifier=config.get('TRANSACC', 'DB_INSTANCE_ID'),
    DBName=config.get('TRANSACC', 'DB_NAME'),
    DBInstanceClass='db.t3.micro',
    Engine='postgres',
    MasterUsername=config.get('TRANSACC', 'DB_USER'),
    MasterUserPassword=config.get('TRANSACC', 'DB_PASSWORD'),
    Port=int(config.get('TRANSACC', 'DB_PORT')),
    PubliclyAccessible=True,
    VpcSecurityGroupIds=[config.get('VPC', 'SECURITY_GROUP')],
    AllocatedStorage=15
    )
    print(response)
except aws_rds_connection.exceptions.DBInstanceAlreadyExists
Fault as ex:
    print("La instancia ya existe")
except Exception as ex:
    print("Error!", ex)
```

Creación de la Base de Datos.

Se obtiene el hostname de la instancia.

Creación de la base de Datos en AWS.

```
try:
    instance = aws_rds_connection.describe_db_instan
    ces(DBInstanceIdentifier=config.get('TRANSACC',
    'DB_INSTANCE_ID'))
    RDS_HOSTNAME = instance.get('DBInstances')
    [0].get('Endpoint').get('Address')
    print(RDS_HOSTNAME)
except Exception as ex:
    print("Error!!", ex)
```

Creación de la Base de Datos.

Se hace la conexión a la Base de Datos.

Creación de la base de Datos en AWS.

```
try:
    db_pg_conn = psycopg2.connect(
        database=config.get('TRANSACC', 'DB_NAME'),
        user=config.get('TRANSACC', 'DB_USER'),
        password=config.get('TRANSACC', 'DB_PASSWORD'),
        host=RDS_HOSTNAME,
        port=int(config.get('TRANSACC', 'DB_PORT'))
    )
    cursor = db_pg_conn.cursor()
    cursor.execute(sql_queries.DDL_QUERY)
    db_pg_conn.commit()
    print("Base de Datos Creada Exitosamente")
except Exception as ex:
    print("Error!", ex)
```

Creación de la Base de Datos.

Se crean funciones para inserción de datos.

Creación de la Base de Datos.

```
def insertData2SQL(data_dict, table_name, driver):

    df_data = pd.DataFrame.from_records(data_dict)
    try:
        response = df_data.to_sql(table_name, driver, index=False,
        if_exists='append')
        print(f"Se han insertado {response} nuevos registros")
    except Exception as ex:
        print(ex)
```

Creación de la Base de Datos.

```
def insertData2SQL_categoria(data_dict, table_name, driver):
    df_data = pd.DataFrame.from_records(data_dict)
    try:
        for index, row in df_data.iterrows():
            estado_value = "'1'" if row.to_dict()['estado'] else
            "'0'"
            query = f"INSERT INTO {table_name} VALUES
            ({row.to_dict()['idcategoria']}, '{row.to_dict()
            ['nombre']}', '{row.to_dict()['descripcion']}',
            CAST({estado_value} AS BIT))"
            cursor.execute(query)
        db_pg_conn.commit()
        print(f"Se han insertado {len(df_data)} nuevos registros")
    except Exception as ex:
        print(ex)
```

Creación de la Base de Datos.

```
def insertData2SQL_rol(data_dict, table_name, driver):
    df_data = pd.DataFrame.from_records(data_dict)
    try:
        for index, row in df_data.iterrows():
            estado_value = "'1'" if row.to_dict()['estado'] else
            "'0'"
            query = f"INSERT INTO {table_name} VALUES
            ({row.to_dict()['idrol']}, '{row.to_dict()
            ['nombre']}', '{row.to_dict()['descripcion']}',
            CAST({estado_value} AS BIT))"
            cursor.execute(query)
        db_pg_conn.commit()
        print(f"Se han insertado {len(df_data)} nuevos registros e
    except Exception as ex:
        print(ex)
```

Creación de la Base de Datos.

```
def insertData2SQL_articulo(data_dict, table_name, driver):
    df_data = pd.DataFrame.from_records(data_dict)
    try:
        for index, row in df_data.iterrows():
            estado_value = "'1'" if row.to_dict()['estado'] else
            "'0'"
            query = f"INSERT INTO {table_name} VALUES
            ({row.to_dict()['idarticulo']}, '{row.to_dict()
            ['idcategoria']}', '{row.to_dict()['codigo']}',
            '{row.to_dict()['nombre']}', '{row.to_dict()
            ['precio_venta']}', '{row.to_dict()['stock']}',
            '{row.to_dict()['descripcion']}', '{row.to_dict()
            ['imagen']}', CAST({estado_value} AS BIT))"
            cursor.execute(query)
        db_pg_conn.commit()
        print(f"Se han insertado {len(df_data)} nuevos registros
        en la tabla {table_name}")
    except Exception as ex:
        print(ex)
```

Creación de la Base de Datos.

```
def insertData2SQL_usuario(data_dict, table_name, driver):
    try:
        for row in data_dict:
            estado_value = 1 if row['estado'] else 0
            query = f"INSERT INTO {table_name} VALUES (%s, %s,
            %s, %s, %s, %s, %s, %s, CAST(%s AS BIT))"
            cursor.execute(query, (
                row['idusuario'],
                row['idrol'],
                row['nombre'],
                row['tipo_documento'],
                row['num_documento'],
                row['direccion'],
                row['telefono'],
                row['email'],
                binascii.hexlify(row['clave']).decode('utf-8'),
                estado_value
            ))
        db_pg_conn.commit()
        print(f"Se han insertado {len(data_dict)} nuevos
        registros en la tabla {table_name}")
    except Exception as ex:
```

Esta función genera números aleatorios únicos para usar en algunas columnas como el número de documento, número de serie de comprobantes, id, etc.

Creación de la Base de Datos.

```
def generate_unique_random_numbers(start, end, count):
    unique_numbers = set()
    while len(unique_numbers) < count:
        random_number = np.random.randint(start, end)
        unique_numbers.add(random_number)
    return list(unique_numbers)
```

Creación de la Base de Datos.

Esta función genera números de teléfono aleatorios de determinados dígitos.

Creación de la Base de Datos.

```
def generate_phone_number(digits):
    phone_number = fake.phone_number()[:digits]
    return phone_number
```

Creación de la Base de Datos.

Se crea variable driver para que lea las credenciales y para usarse en las funciones de inserción de datos.

Creación de la Base de Datos.

```
driver = f"""postgresql://{config.get('TRANSACC', 'DB_USER')}:
{config.get('TRANSACC', 'DB_PASSWORD')}@{RDS_HOSTNAME}:
{config.get('TRANSACC', 'DB_PORT')}/{config.get('TRANSACC',
'DB_NAME')}"""
```

Creación de la Base de Datos.

Se agregan datos a la tabla categoria.

Creación de la Base de Datos.

```
data_categoria = [
    {'idcategoria': 39582, 'nombre': 'Ropa y Accesorios',
    'descripcion': 'Incluye prendas de vestir para hombres,
    mujeres y ninos, asi como accesorios como bolsos, sombreros,
    bufandas, etc.', 'estado': True},
    {'idcategoria': 82016, 'nombre': 'Calzado', 'descripcion':
    'Incluye zapatos formales, tenis, botas, sandalias, para
    hombres, mujeres y ninos.', 'estado': True},
    {'idcategoria': 51470, 'nombre': 'Juguetes y Juegos',
    'descripcion': 'Una amplia gama de juguetes para ninos de
    todas las edades, asi como juegos de mesa y articulos
    deportivos.', 'estado': True},
    {'idcategoria': 75369, 'nombre': 'Electronicos',
    'descripcion': 'Desde telefonos moviles, computadoras
    portatiles y tabletas hasta accesorios como auriculares,
    cargadores y fundas.', 'estado': True},
    {'idcategoria': 20934, 'nombre': 'Mascotas', 'descripcion':
    'Ofrece productos para el cuidado de mascotas, alimentos,
    juguetes y accesorios para perros, gatos y otras mascotas.',
    'estado': True},
```

Creación de la Base de Datos.

```
{'idcategoria': 64218, 'nombre': 'Alimentos y Bebidas',
'descripcion': 'Desde alimentos enlatados y productos secos
hasta bebidas como vinos, cervezas artesanales y licores.',
'estado': True},
{'idcategoria': 13750, 'nombre': 'Libros y Papeleria',
'descripcion': 'Ofrece una selección de libros para todas las edad
{'idcategoria': 40879, 'nombre': 'Belleza y Cuidado Personal',
'descripcion': 'Incluye productos de cuidado de la piel,
maquillaje, perfumes, asi como herramientas y accesorios para el
cuidado del cabello.', 'estado': True},
{'idcategoria': 97521, 'nombre': 'Joyeria', 'descripcion': 'Desde
anillos, pulseras y collares hasta relojes y joyeria
personalizada.', 'estado': True},
{'idcategoria': 32684, 'nombre': 'Hogar y Decoracion',
'descripcion': 'Ofrece muebles para el hogar, articulos de
decoracion como cuadros y jarrones, asi como utensilios de cocina
]

insertData2SQL_categoria(data_categoria, 'categoria', driver)
```

Creación de la Base de Datos.

Se agregan datos a la tabla rol.

Creación de la Base de Datos.

```
data_roles = [
    {'idrol': 10001, 'nombre': 'Administrador', 'descripcion':
    'Este rol tiene acceso completo al sistema y puede realizar
    todas las acciones.', 'estado': True},
    {'idrol': 10002, 'nombre': 'Moderador', 'descripcion': 'Este
    rol tiene permisos para moderar contenido y realizar acciones
    específicas.', 'estado': True},
    {'idrol': 10003, 'nombre': 'Usuario Estándar', 'descripcion':
    'Este rol tiene acceso limitado y puede realizar acciones
    básicas en el sistema.', 'estado': True},
    {'idrol': 10004, 'nombre': 'Soporte Técnico', 'descripcion':
    'Este rol proporciona soporte técnico y tiene acceso a
    herramientas de diagnóstico.', 'estado': True},
    {'idrol': 10005, 'nombre': 'Invitado', 'descripcion': 'Este
    rol tiene acceso limitado y solo puede ver contenido
    público.', 'estado': True}
]

insertData2SQL_rol(data_roles, 'rol', driver)
```

Creación de la Base de Datos.

Se agregan datos a la tabla articulo.

Creación de la Base de Datos.

```
fake = Faker()

categorias_por_palabra_clave = {
    "camisa": 39582,
    "pantalón": 39582,
    "zapatos": 82016,
    "tenis": 82016,
    "vestido": 39582,
    "reloj": 97521,
    "teléfono": 75369,
    "portátil": 75369,
    "tableta": 75369,
    "juguete": 51470,
    "libro": 13750,
    "silla": 32684,
    "mesa": 32684,
```

Creación de la Base de Datos.


```
    "lámpara": 32684,
    "cuadro": 32684,
    "cocina": 32684,
    "cama": 32684,
    "alfombra": 32684,
    "jarrón": 32684,
    "perfume": 40879,
    "maquillaje": 40879,
    "galleta": 64218,
    "pelota": 51470,
    "anillo": 97521,
    "suéter": 39582,
    "bolígrafo": 13750
}
```

Creación de la Base de Datos.

```
palabras_clave = categorias_por_palabra_clave.keys()

articulos_categoria = {}

while len(articulos_categoria) < 1000:
    palabra_clave = random.choice(list(palabras_clave))
    palabra_aleatoria = fake.word().capitalize()
    nombre_articulo = palabra_clave.capitalize() + " " +
    palabra_aleatoria
    id_categoria = categorias_por_palabra_clave[palabra_clave]
    articulos_categoria[nombre_articulo] = id_categoria
```

Creación de la Base de Datos.

```
def generar_codigo(length):
    caracteres = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOP
    QRSTUVWXYZ0123456789'
    codigo = ''.join(random.choice(caracteres) for _ in
    range(length))
    return codigo
```

Creación de la Base de Datos.

```
cantidad_articulos = 1000

data_articulos = []

start_number_articulo = 100000
end_number_articulo = 999999
number_count_articulo = cantidad_articulos

unique_random_numbers_idarticulos = generate_unique_random_numbers
end_number_articulo, number_count_articulo)

lista_nombre_articulo = list(articulos_categoria.keys())
```

Creación de la Base de Datos.

```
for index, articulo in enumerate(range(cantidad_articulos)):
    nombre_articulo = lista_nombre_articulo[index]
    nuevo_articulo = {
        'idarticulo': unique_random_numbers_idarticulos[index],
        'idcategoria': articulos_categoria[nombre_articulo],
        'codigo': generar_codigo(8),
        'nombre': nombre_articulo,
        'precio_venta': round(random.uniform(10, 1000),2),
        'stock': np.random.randint(1, 100),
        'descripcion': f'Artículo {nombre_articulo} en venta en
        tienda',
        'imagen': 'fig_' + str(index),
        'estado': True
    }
    data_articulos.append(nuevo_articulo)
insertData2SQL_articulo(data_articulos, 'articulo', driver)
```

Creación de la Base de Datos.

Se agregan datos a la tabla persona.

Creación de la Base de Datos.

```
tipo_persona = ['cliente', 'proveedor']
tipo_documento = ['DNI', 'NIE', 'Pasaporte', 'Tarjeta Residencia']

cantidad_personas = 1000

start_number_persona = 100000000
end_number_persona = 999999999
number_count_persona = cantidad_personas

unique_random_numbers_idpersona = generate_unique_random_numbers(s
end_number_persona, number_count_persona)

start_number_num_doc_persona = 100000000000000
end_number_num_doc_persona = 199999999999999
number_count_num_doc_persona = cantidad_personas

unique_random_numbers_num_doc_persona = generate_unique_random_num
doc_persona, end_number_num_doc_persona,
number_count_num_doc_persona)
```

Creación de la Base de Datos.

```
data_personas = []
for index, persona in enumerate(range(cantidad_personas)):
    perfil_persona = fake.profile()
    nombre_persona = perfil_persona['name']
    direccion_persona = perfil_persona['address']
    direccion_persona = direccion_persona.replace("\n", " ")
    email_persona = perfil_persona['mail']
    # Generar un número de teléfono
    numero_telefono = generate_phone_number(8)
    nueva_persona = {
        'idpersona': unique_random_numbers_idpersona[index],
        'tipo_persona': tipo_persona[np.random.randint(0,2)],
        'nombre': nombre_persona,
        'tipo_documento': tipo_documento[np.random.randint(0,4)],
        'num_documento':
        unique_random_numbers_num_doc_persona[index],
        'direccion': direccion_persona,
        'telefono': numero_telefono,
        'email': email_persona,
    }
    data_personas.append(nueva_persona)
insertData2SQL(data_personas, 'persona', driver)
```

Creación de la Base de Datos.

Se agregan datos a la tabla usuario.

Creación de la Base de Datos.

```
cantidad_usuarios = 1000
estados = [True, False]

start_number_usuario = 1000000
end_number_usuario = 9999999
number_count_usuario = cantidad_usuarios

unique_random_numbers_idusuario =
generate_unique_random_numbers(start_number_usuario,
end_number_usuario, number_count_usuario)

start_number_num_doc_usuario = 100000000000000
end_number_num_doc_usuario = 199999999999999
number_count_num_doc_usuario = cantidad_usuarios

unique_random_numbers_num_doc_usuario =
generate_unique_random_numbers(start_number_num_doc_usuario,
end_number_num_doc_usuario, number_count_num_doc_usuario)

data_usuarios = []
```

Creación de la Base de Datos.

```
for index, usuario in enumerate(range(cantidad_usuarios)):
    perfil_persona = fake.profile()
    nombre_persona = perfil_persona['name']
    direccion_persona = perfil_persona['address']
    direccion_persona = direccion_persona.replace("\n", " ")
    email_persona = perfil_persona['mail']
    longitud_bytes = 10
    dato_binario = os.urandom(longitud_bytes)
    numero_telefono = generate_phone_number(8)
    nuevo_usuario = {
        'idusuario': unique_random_numbers_idusuario[index],
        'idrol': random.choices(data_roles, weights=(0.1, 0.2, 0.4
        'nombre': nombre_persona,
        'tipo_documento': tipo_documento[np.random.randint(0,3)],
        'num_documento': unique_random_numbers_num_doc_usuario[ind
        'direccion': direccion_persona,
        'telefono': numero_telefono,
        'email': email_persona,
        'clave': dato_binario,
        'estado': estados[np.random.randint(0,2)],
    }
    data_usuarios.append(nuevo_usuario)
```

Creación de la Base de Datos.

Se agregan datos a la tabla venta.

Creación de la Base de Datos.

```
cantidad_venta = 1000
tipos_comprobante = ['Factura', 'Boleta de Venta', 'Ticket de
Venta', 'Nota de Crédito', 'Nota de Débito', 'Recibo']
estados_venta = ['Pendiente', 'En proceso', 'Completada',
'Cancelada', 'Reembolsada']

fecha_inicio = datetime(2015, 1, 1)
fecha_fin = datetime(2023, 12, 31)
fecha_aleatoria = fecha_inicio + timedelta(days=random.randint(0,
(fecha_fin - fecha_inicio).days))
start_number_venta = 111111
end_number_venta = 999999

number_count_venta = cantidad_venta
unique_random_numbers_idventa =

generate_unique_random_numbers(start_number_venta,
end_number_venta, number_count_venta)
start_number_serie_comprob = 1000000
end_number_serie_comprob = 9999999
number_count_serie_comprob = cantidad_venta
```

Creación de la Base de Datos.

```
unique_random_numbers_serie_comprob = generate_unique_random_numbe
end_number_serie_comprob, number_count_serie_comprob)

start_number_num_comprob = 10000000
end_number_num_comprob = 99999999
number_count_num_comprob = cantidad_venta

unique_random_numbers_num_comprob = generate_unique_random_numbers
end_number_num_comprob, number_count_num_comprob)

data_venta = []
```

Paso 6.

```
for index, venta in enumerate(range(cantidad_venta)):
    index_usuario = np.random.randint(0, len(data_usuarios))
    idusuario = data_usuarios[index_usuario]['idusuario']
    fecha_aleatoria = fecha_inicio + timedelta(days=random.randint
    nueva_venta = {
        'idventa': unique_random_numbers_idventa[index],
        'idcliente': random.sample(data_personas, 1)[0]
        ['idpersona'],
        'idusuario': random.sample(data_usuarios, 1)[0]
        ['idusuario'],
        'tipo_comprobante':
        tipos_comprobante[np.random.randint(0,6)],
        'serie_comprobante':
        unique_random_numbers_serie_comprob[index],
        'num_comprobante':
        unique_random_numbers_num_comprob[index],
        'fecha': fecha_aleatoria,
        'impuesto': round(random.uniform(0, 99.99), 2),
        'total': round(random.uniform(0, 99999.99), 2),
        'estado': estados_venta[np.random.randint(0,5)]
    }
    data_venta.append(nueva_venta)
```

Creación de la Base de Datos.

Se agregan datos a la tabla detalles venta.

Creación de la Base de Datos.

```
cantidad_detalles_venta = 5000

start_number_detalle_venta = 3333333
end_number_detalle_venta = 9999999
number_count_detalle_venta = cantidad_detalles_venta

unique_random_numbers_iddetalle_venta =
generate_unique_random_numbers(start_number_detalle_venta,
end_number_detalle_venta, number_count_detalle_venta)
```

Creación de la Base de Datos.

```
data_detalles_venta = []
for index, detalles_venta in
enumerate(range(cantidad_detalles_venta)):
    muestra_tabla_articulo = random.sample(data_articulos, 1)
    idarticulo_muestra = muestra_tabla_articulo[0]['idarticulo']
    precio_muestra = muestra_tabla_articulo[0]['precio_venta']
    nuevo_detalle_venta = {
        'iddetalle_venta':
        unique_random_numbers_iddetalle_venta[index],
        'idventa': random.sample(data_venta, 1)[0]['idventa'],
        'idarticulo': idarticulo_muestra,
        'cantidad': np.random.randint(0,100),
        'precio': precio_muestra,
        'descuento': round(random.uniform(0, 999.99), 2)
    }
    data_detalles_venta.append(nuevo_detalle_venta)
insertData2SQL(data_detalles_venta, 'detalle_venta', driver)
```

Creación de la Base de Datos.

Se agregan datos a la tabla ingreso.

Creación de la Base de Datos.

```
cantidad_ingresos = 1000
estados_ingreso = ['Pendiente', 'En proceso', 'Completada',
'Cancelada', 'Reembolsada']

start_number_ingreso = 100000000
end_number_ingreso = 999999999
number_count_ingreso = cantidad_ingresos

unique_random_numbers_idingreso =
generate_unique_random_numbers(start_number_ingreso,
end_number_ingreso, number_count_ingreso)

start_number_serie_comprob_ingreso = 1000000
end_number_serie_comprob_ingreso = 9999999
number_count_serie_comprob_ingreso = cantidad_ingresos

unique_random_numbers_serie_comprob_ingreso =
generate_unique_random_numbers(start_number_serie_comprob_ingreso,
end_number_serie_comprob_ingreso,
number_count_serie_comprob_ingreso)
```

Creación de la Base de Datos.

```
data_ingresos = []
for index, ingreso in enumerate(range(cantidad_ingresos)):
    fecha_aleatoria = fecha_inicio +
    timedelta(days=random.randint(0, (fecha_fin -
    fecha_inicio).days))
    nuevo_ingreso = {
        'idingreso': unique_random_numbers_idingreso[index],
        'idproveedor': random.sample(data_personas, 1)[0]
        ['idpersona'],
        'idusuario': random.sample(data_usuarios, 1)[0]
        ['idusuario'],
        'tipo_comprobante':
        tipos_comprobante[np.random.randint(0,6)],
        'serie_comprobante':
        unique_random_numbers_serie_comprob_ingreso[index],
        'fecha': fecha_aleatoria,
        'impuesto': round(random.uniform(0, 99.99), 2),
        'total': round(random.uniform(0, 99999.99), 2),
        'estado': estados_ingreso[np.random.randint(0,5)]
    }
    data_ingresos.append(nuevo_ingreso)
insertData2SQL(data_ingresos, 'ingreso', driver)
```

Se agregan datos a la tabla detalle ingreso.

Creación de la Base de Datos.

```
cantidad_detalles_ingreso = 1000

start_number_detalle_ingreso = 100000000
end_number_detalle_ingreso = 999999999
number_count_detalle_ingreso = cantidad_detalles_ingreso

unique_random_numbers_iddetalle_ingreso =
generate_unique_random_numbers(start_number_detalle_ingreso,
end_number_detalle_ingreso, number_count_detalle_ingreso)
```

Creación de la Base de Datos.

```
data_detalles_ingreso = []
for index, detalles_ingreso in
enumerate(range(cantidad_detalles_ingreso)):
    muestra_tabla_articulo = random.sample(data_articulos, 1)
    idarticulo_muestra = muestra_tabla_articulo[0]['idarticulo']
    precio_muestra = muestra_tabla_articulo[0]['precio_venta']
    nuevo_detalle_ingreso = {
        'iddetalle_ingreso':
        unique_random_numbers_iddetalle_ingreso[index],
        'idingreso': random.sample(data_ingresos, 1)[0]
        ['idingreso'],
        'idarticulo': idarticulo_muestra,
        'cantidad': np.random.randint(0,100),
        'precio': precio_muestra
    }
    data_detalles_ingreso.append(nuevo_detalle_ingreso)
insertData2SQL(data_detalles_ingreso, 'detalle_ingreso', driver)
```

Se crea la dimensión de la tabla persona.

Creación de Data Warehouse.

```
sql_query = 'SELECT * FROM persona;'
dimPersonas = pd.read_sql(sql_query, postgres_driver)
dimPersonas.to_csv('dimPersonas.csv', index=False)
dimPersonas.head()
```

Se crea la dimensión de la tabla usuario.

Creación de Data Warehouse.

```
sql_query = 'SELECT * FROM usuario;'
df_usuario = pd.read_sql(sql_query, postgres_driver)
df_usuario = df_usuario.rename(columns={'nombre':
'nombre_usuario', 'estado': 'estado_usuario'})

sql_query = 'SELECT * FROM rol;'
df_rol = pd.read_sql(sql_query, postgres_driver)
df_rol = df_rol.rename(columns={'nombre': 'nombre_rol', 'estado':
'estado_rol'})
```

Creación de Data Warehouse.

```
df_usuario = df_usuario.merge(df_rol, how='inner', on='idrol')
dimUsuario = df_usuario.drop(['idrol'], axis=1)
dimUsuario.to_csv('dimUsuario.csv', index=False)
```

Se crea la dimensión de la tabla articulo.

Creación de Data Warehouse.

```
sql_query = 'SELECT * FROM articulo;'
df_articulo = pd.read_sql(sql_query, postgres_driver)
df_articulo = df_articulo.rename(columns={'nombre':
'nombre_articulo', 'descripcion': 'descripcion_articulo',
'estado': 'estado_articulo'})

sql_query = 'SELECT * FROM categoria;'
df_categoria = pd.read_sql(sql_query, postgres_driver)
df_categoria = df_categoria.rename(columns={'nombre':
'nombre_categoria', 'descripcion': 'descripcion_categoria',
'estado': 'estado_categoria'})
```

Creación de Data Warehouse.

```
df_articulo = df_articulo.merge(df_categoria, how='inner',
on='idcategoria')
dimArticulo = df_articulo.drop(['idcategoria'], axis=1)
dimArticulo.to_csv('dimArticul.csv', index=False)
```

Se crea la dimensión de la tabla venta.

Creación de Data Warehouse.

```
sql_query = 'SELECT * FROM venta;'
df_venta = pd.read_sql(sql_query, postgres_driver)
df_venta = df_venta.rename(columns={'estado': 'estado_venta'})

sql_query = 'SELECT * FROM persona'
df_cliente = pd.read_sql(sql_query, postgres_driver)
df_cliente = df_cliente[df_cliente['tipo_persona']=='cliente']
df_cliente = df_cliente.rename(columns={'idpersona': 'idcliente',
'nombre': 'nombre_cliente', 'tipo_documento':
'tipo_documento_cliente', 'num_documento':
'num_documento_cliente','direccion': 'direccion_cliente',
'telefono': 'telefono_cliente', 'email': 'email_cliente'})
```

Creación de Data Warehouse.

```
df_venta_cliente = df_venta.merge(df_cliente, how='inner',
on='idcliente')

sql_query = 'SELECT * FROM usuario;'
df_usuario = pd.read_sql(sql_query, postgres_driver)
df_usuario = df_usuario.rename(columns={'nombre':
'nombre_usuario', 'tipo_documento': 'tipo_documento_usuario',
'num_documento': 'num_documento_usuario','direccion':
'direccion_usuario', 'telefono': 'telefono_usuario', 'email':
'email_usuario', 'estado': 'estado_cliente'})
```

Creación de Data Warehouse.

```
df_venta_cliente_usuario = df_venta_cliente.merge(df_usuario,
how='inner', on='idusuario')
dimVenta = df_venta_cliente_usuario.drop(['idcliente',
'idusuario', 'idrol'], axis=1)
dimVenta.to_csv('dimVenta.csv', index=False)
```

Se crea la dimensión de la tabla ingreso.

Creación de Data Warehouse.

```
sql_query = 'SELECT * FROM ingreso;'
df_ingreso = pd.read_sql(sql_query, postgres_driver)
df_ingreso = df_ingreso.rename(columns={'estado':
'estado_ingreso'})

sql_query = 'SELECT * FROM persona;'
df_proveedor = pd.read_sql(sql_query, postgres_driver)
df_proveedor =
df_proveedor[df_proveedor['tipo_persona']=='proveedor']
df_proveedor = df_proveedor.rename(columns={'idpersona':
'idproveedor', 'nombre': 'nombre_proveedor', 'tipo_documento':
'tipo_documento_proveedor', 'num_documento':
'num_documento_proveedor', 'direccion': 'direccion_proveedor',
'telefono': 'telefono_proveedor', 'email': 'email_proveedor'})
```

Creación de Data Warehouse.

```
df_ingreso_proveedor = df_ingreso.merge(df_proveedor,
how='inner', on='idproveedor')

sql_query = 'SELECT * FROM usuario'
df_usuario = pd.read_sql(sql_query, postgres_driver)
df_usuario = df_usuario.rename(columns={'nombre':
'nombre_usuario', 'tipo_documento': 'tipo_documento_usuario',
'num_documento': 'num_documento_usuario', 'direccion':
'direccion_usuario', 'telefono': 'telefono_usuario', 'email':
'email_usuario', 'estado': 'estado_usuario'})
```

Creación de Data Warehouse.

```
dimIngreso = df_ingreso_proveedor.merge(df_usuario, how='inner',
on='idusuario')
dimIngreso.drop(['idproveedor', 'idusuario', 'idrol'], axis=1,
inplace=True)
dimIngreso.to_csv('dimIngreso.csv', index=False)
```

Se crea la dimensión de la tabla detalle venta.

Creación de Data Warehouse.

```
sql_query = 'SELECT * FROM detalle_venta;'
df_detalle_venta = pd.read_sql(sql_query, postgres_driver)

sql_query = 'SELECT * FROM venta;'
df_venta = pd.read_sql(sql_query, postgres_driver)
df_venta = df_venta.rename(columns={'estado': 'estado_venta'})
```

Creación de Data Warehouse.

```
df_detalle_venta_venta = df_detalle_venta.merge(df_venta,
how='inner', on='idventa')

sql_query = 'SELECT *FROM articulo;'
df_articulo = pd.read_sql(sql_query, postgres_driver)
df_articulo = df_articulo.rename(columns={'estado':
'estado_articulo'})
```

Creación de Data Warehouse.

```
dimDetalleVenta = df_detalle_venta_venta.merge(df_articulo,
how='inner', on='idarticulo')
dimDetalleVenta.drop(['idventa', 'idarticulo', 'idcliente',
'idusuario'], axis=1,  inplace=True)
dimDetalleVenta.to_csv('dimDetalleventa.csv', index=False)
```

Se crea la dimensión de la tabla detalle ingreso.

Creación de Data Warehouse.

```
sql_query = 'SELECT * FROM detalle_ingreso;'
df_detalle_ingreso = pd.read_sql(sql_query, postgres_driver)

sql_query = 'SELECT * FROM ingreso;'
df_ingreso = pd.read_sql(sql_query, postgres_driver)
df_ingreso = df_ingreso.rename(columns={'estado':
'estado_ingreso'})
```

Creación de Data Warehouse.

```
df_detalle_ingreso_ingreso =
df_detalle_ingreso.merge(df_ingreso, how='inner', on='idingreso')

sql_query = 'SELECT * FROM articulo;'
df_articulo = pd.read_sql(sql_query, postgres_driver)
df_articulo = df_articulo.rename(columns={'estado':
'estado_articulo', 'nombre': 'nombre_articulo'})
```

Creación de Data Warehouse.

```
dimDetalleIngreso = df_detalle_ingreso_ingreso.merge(df_articulo,
on='idarticulo')
dimDetalleIngreso.drop(['idingreso', 'idarticulo',
'idproveedor', 'idusuario'], axis=1,  inplace=True)
dimDetalleIngreso.to_csv('dimDetalleIngreso.csv', index=False)
```

Se crea la dimensión de la tabla rol.

Creación de Data Warehouse.

```
sql_query = 'SELECT * FROM rol;'
dimRol = pd.read_sql(sql_query, postgres_driver)
dimRol.to_csv('dimRol.csv', index=False)
```

Se crea la dimensión de la tabla categoria.

Creación de Data Warehouse.

```
sql_query = 'SELECT * FROM categoria;'
dimCategoria = pd.read_sql(sql_query, postgres_driver)
dimCategoria.to_csv('dimCategoria.csv', index=False)
```

Se crea la dimensión de la tabla de tiempo.

Creación de Data Warehouse.

```
sql_query = 'SELECT fecha FROM venta;'
dimFecha = pd.read_sql(sql_query, postgres_driver)

dimFecha['year'] = pd.DatetimeIndex(dimFecha['fecha']).year
dimFecha['month'] = pd.DatetimeIndex(dimFecha['fecha']).month
dimFecha['quarter'] = pd.DatetimeIndex(dimFecha['fecha']).quarter
dimFecha['day'] = pd.DatetimeIndex(dimFecha['fecha']).day
dimFecha['week'] = pd.DatetimeIndex(dimFecha['fecha']).week
dimFecha['dayofweek'] =
pd.DatetimeIndex(dimFecha['fecha']).dayofweek
dimFecha['is_weekend'] = dimFecha['dayofweek'].apply(lambda x: 1
if x > 5 else 0)
```

Creación de Data Warehouse.

```
dimFecha['id_fecha'] = dimFecha['year'].astype(str) +
dimFecha['month'].astype(str)
dimFecha['id_fecha'] = dimFecha['id_fecha'].astype(str) +
dimFecha['day'].astype(str)

dimFecha.drop_duplicates(inplace=True)
dimFecha.to_csv('dimFecha.csv', index=False)
```

Se crea la tabla de hechos.

Creación de Data Warehouse.

```
sql_query = '''SELECT idventa, fecha, idcliente,
            idusuario, tipo_comprobante, serie_comprobante,
            num_comprobante, impuesto, total,
            estado FROM venta;'''
df_factTable = pd.read_sql(sql_query, postgres_driver)
df_factTable.to_csv('Tabla_de_Hechos.csv', index=False)
```

Preguntas de Negocio.

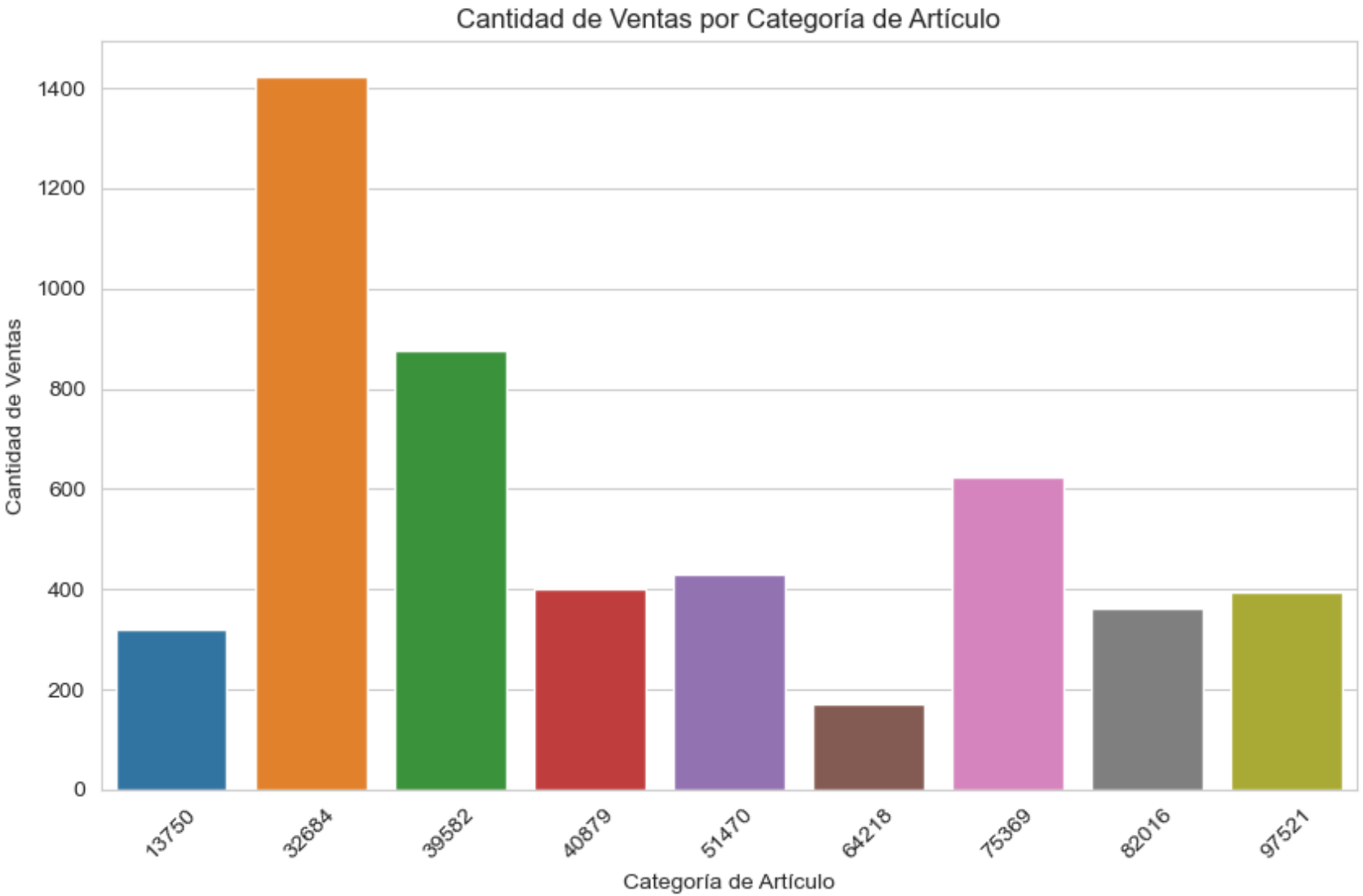
Pregunta No.1

¿Cuáles son las categorías de artículo más vendidas?

Preguntas de Negocio.

```
data_Detalles_Venta = pd.read_csv('dimDetalleVenta.csv')

plt.figure(figsize=(10, 6))
sns.countplot(data=data_Detalles_Venta, x='idcategoria')
plt.title('Cantidad de Ventas por Categoría de Artículo')
plt.xlabel('Categoría de Artículo')
plt.ylabel('Cantidad de Ventas')
plt.xticks(rotation=45)
plt.show()
```

Preguntas de Negocio.



Pregunta No.2

¿Cómo varía el total de ventas entre estas categorías?

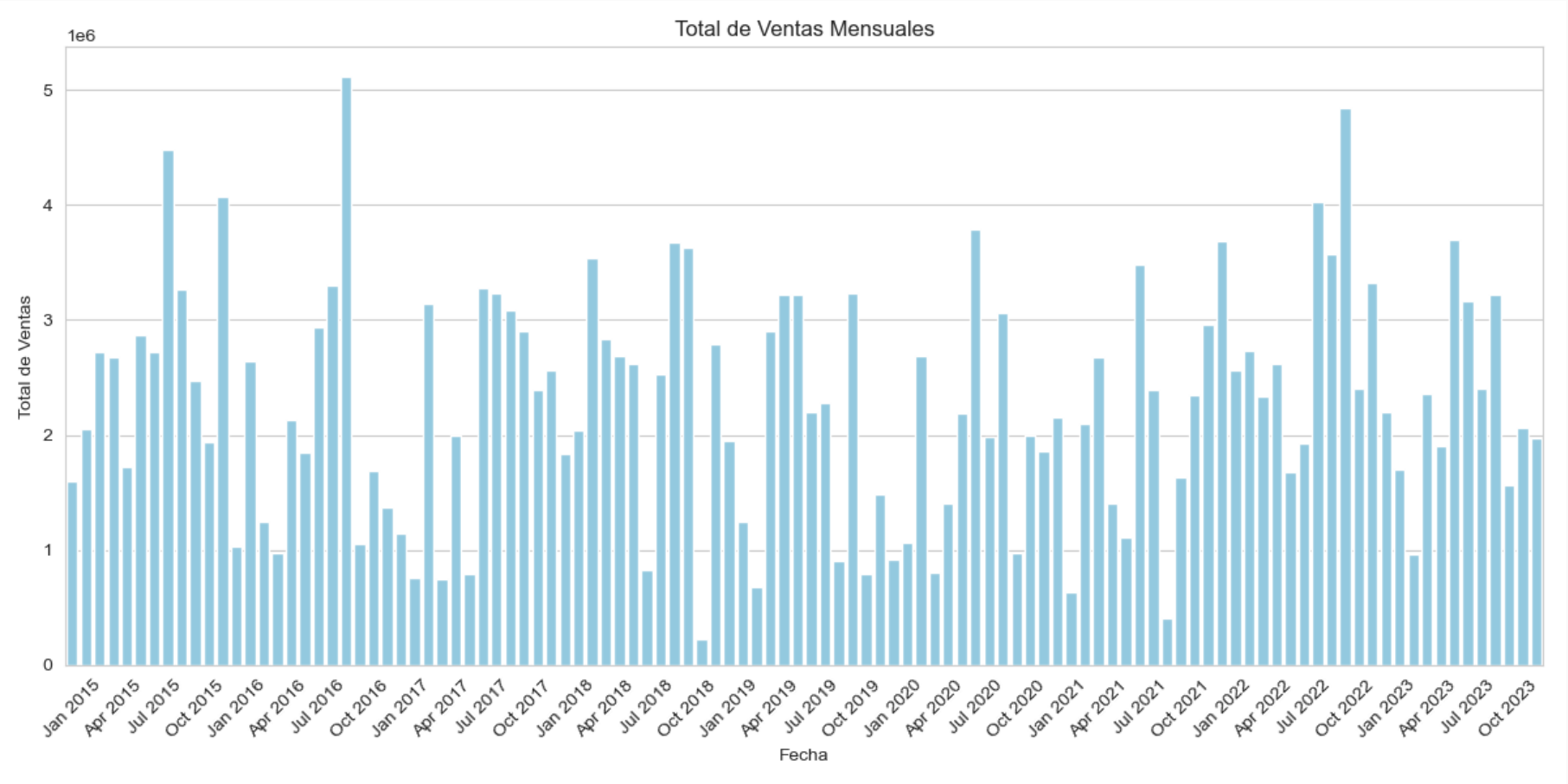
Preguntas de Negocio.

```
data_Detalles_Venta['fecha'] =
pd.to_datetime(data_Detalles_Venta['fecha'])

ventas_mensuales =
data_Detalles_Venta.groupby(pd.Grouper(key='fecha',
freq='M')).agg({'total': 'sum'}).reset_index()

ventas_trimestrales =
data_Detalles_Venta.groupby(pd.Grouper(key='fecha',
freq='Q')).agg({'total': 'sum'}).reset_index()

ventas_anuales =
data_Detalles_Venta.groupby(pd.Grouper(key='fecha',
freq='Y')).agg({'total': 'sum'}).reset_index()
```

Preguntas de Negocio.

```
sns.set_style("whitegrid")
ticks_seleccionados = ventas_mensuales['fecha'][::3]
plt.figure(figsize=(12, 6))
sns.barplot(x=ventas_mensuales['fecha'].dt.strftime('%b %Y'),
y=ventas_mensuales['total'], color='skyblue')
plt.title('Total de Ventas Mensuales')
plt.xlabel('Fecha')
plt.ylabel('Total de Ventas')
# Establecer los ticks del eje x
plt.xticks(ticks=ticks_seleccionados.index,
labels=ticks_seleccionados.dt.strftime('%b %Y'), rotation=45,
ha='center')
plt.tight_layout()
plt.show()
```

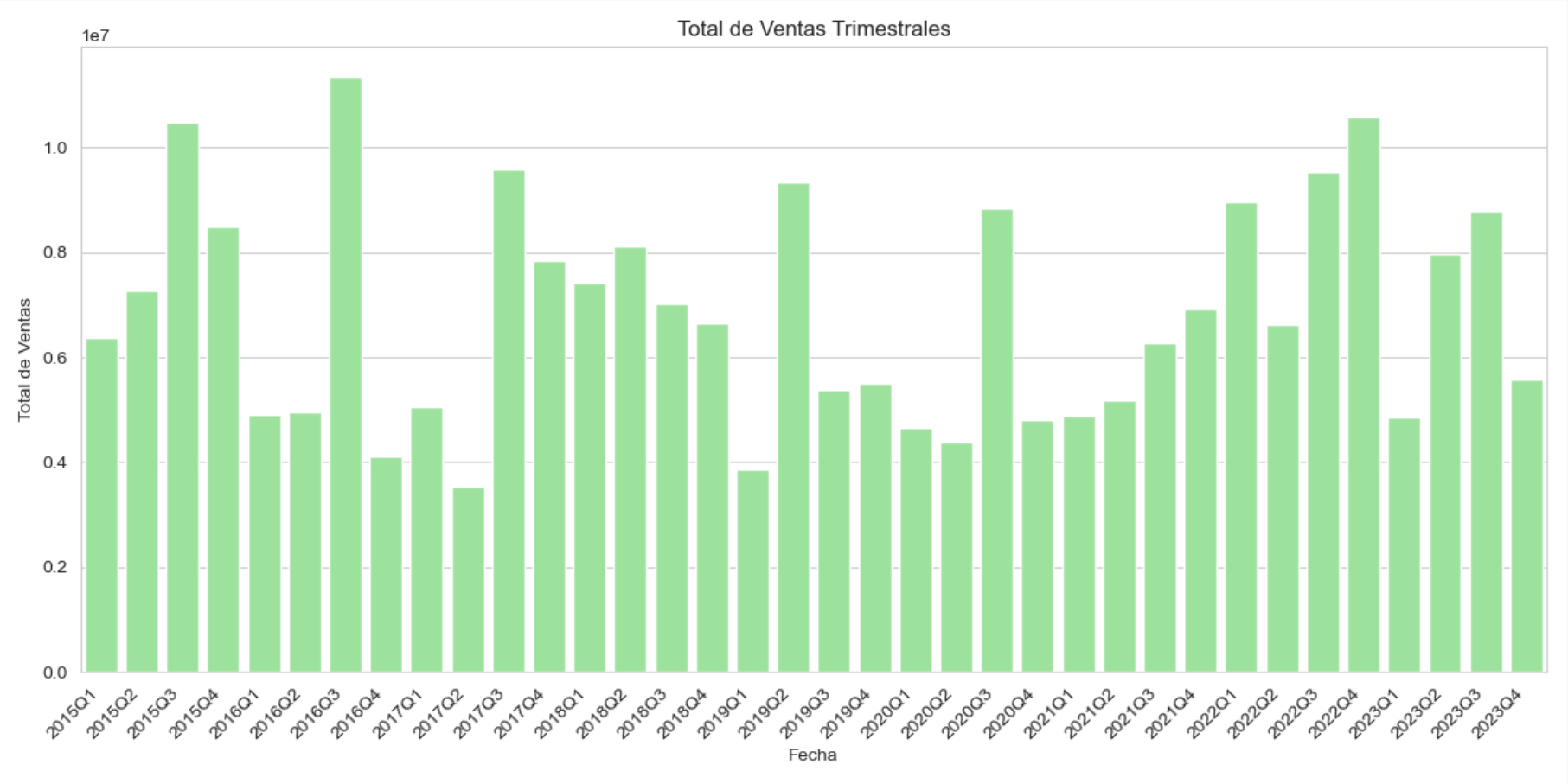
Preguntas de Negocio.



Preguntas de Negocio.

```
plt.figure(figsize=(12, 6))
sns.barplot(x=ventas_trimestrales['fecha'].dt.to_period('Q').asty
pe(str), y=ventas_trimestrales['total'], color='lightgreen')
plt.title('Total de Ventas Trimestrales')
plt.xlabel('Fecha')
plt.ylabel('Total de Ventas')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

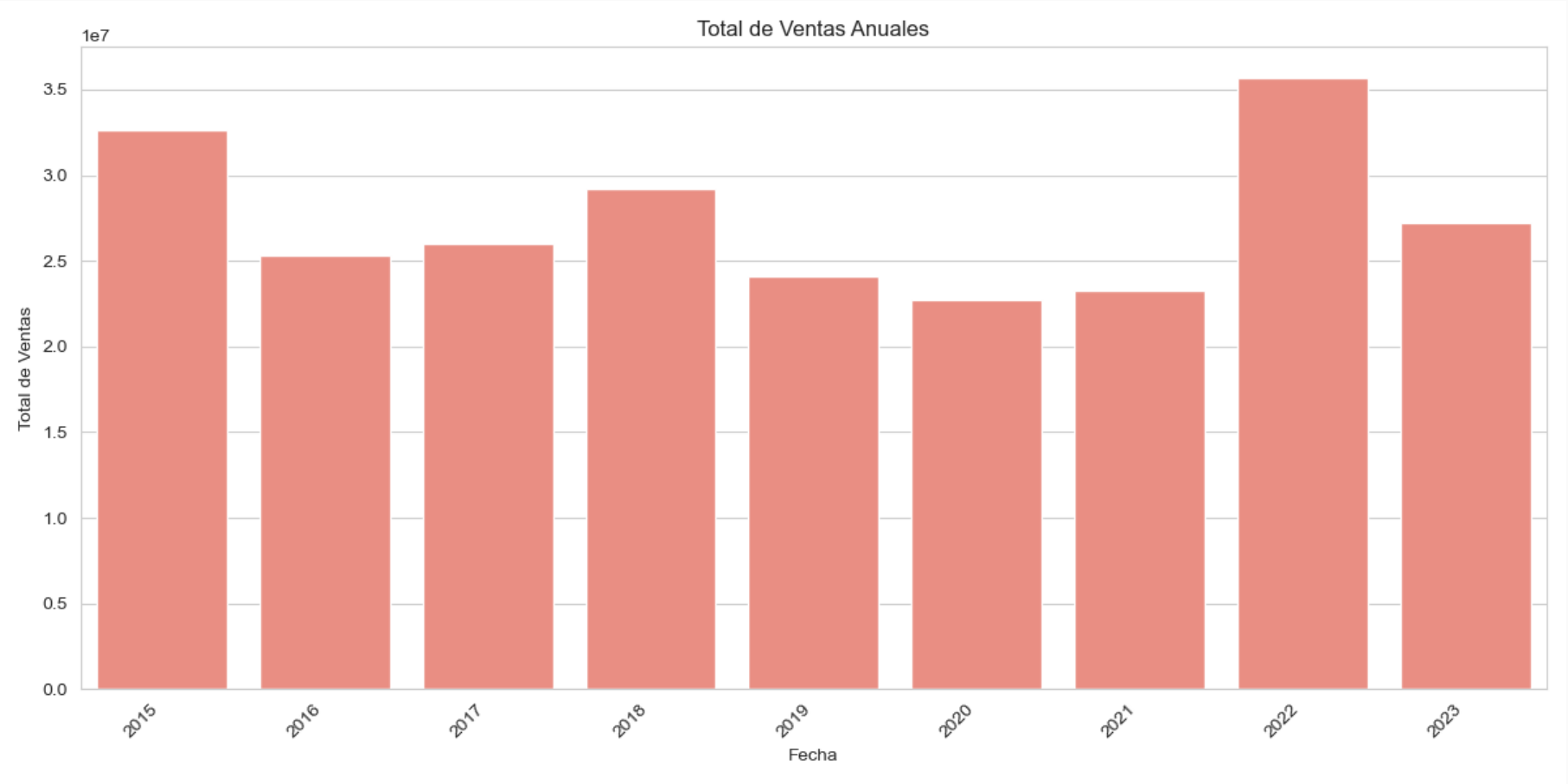
Preguntas de Negocio.



Preguntas de Negocio.

```
plt.figure(figsize=(12, 6))
sns.barplot(x=ventas_anuales['fecha'].dt.strftime('%Y'),
y=ventas_anuales['total'], color='salmon')
plt.title('Total de Ventas Anuales')
plt.xlabel('Fecha')
plt.ylabel('Total de Ventas')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

Preguntas de Negocio.



Pregunta No.3

¿Qué rol o roles tienden a tener un mayor impacto en el resultado final de las ventas?

Preguntas de Negocio.

```
sql_query = 'SELECT * FROM venta;'
df_venta = pd.read_sql(sql_query, postgres_driver)
df_venta = df_venta.rename(columns={'estado': 'estado_venta'})
```

Preguntas de Negocio.

```
sql_query = 'SELECT * FROM persona'
df_cliente = pd.read_sql(sql_query, postgres_driver)
df_cliente = df_cliente[df_cliente['tipo_persona']=='cliente']
df_cliente = df_cliente.rename(columns={'idpersona':
'idcliente', 'nombre': 'nombre_cliente', 'tipo_documento':
'tipo_documento_cliente', 'num_documento':
'num_documento_cliente','direccion': 'direccion_cliente',
'telefono': 'telefono_cliente', 'email': 'email_cliente'})
```

Preguntas de Negocio.

```
df_venta_cliente = df_venta.merge(df_cliente, how='inner',
on='idcliente')
```

Preguntas de Negocio.

```
sql_query = 'SELECT * FROM usuario;'
df_usuario = pd.read_sql(sql_query, postgres_driver)
df_usuario = df_usuario.rename(columns={'nombre':
'nombre_usuario', 'tipo_documento': 'tipo_documento_usuario',
'num_documento': 'num_documento_usuario','direccion':
'direccion_usuario', 'telefono': 'telefono_usuario', 'email':
'email_usuario', 'estado': 'estado_cliente'})
```

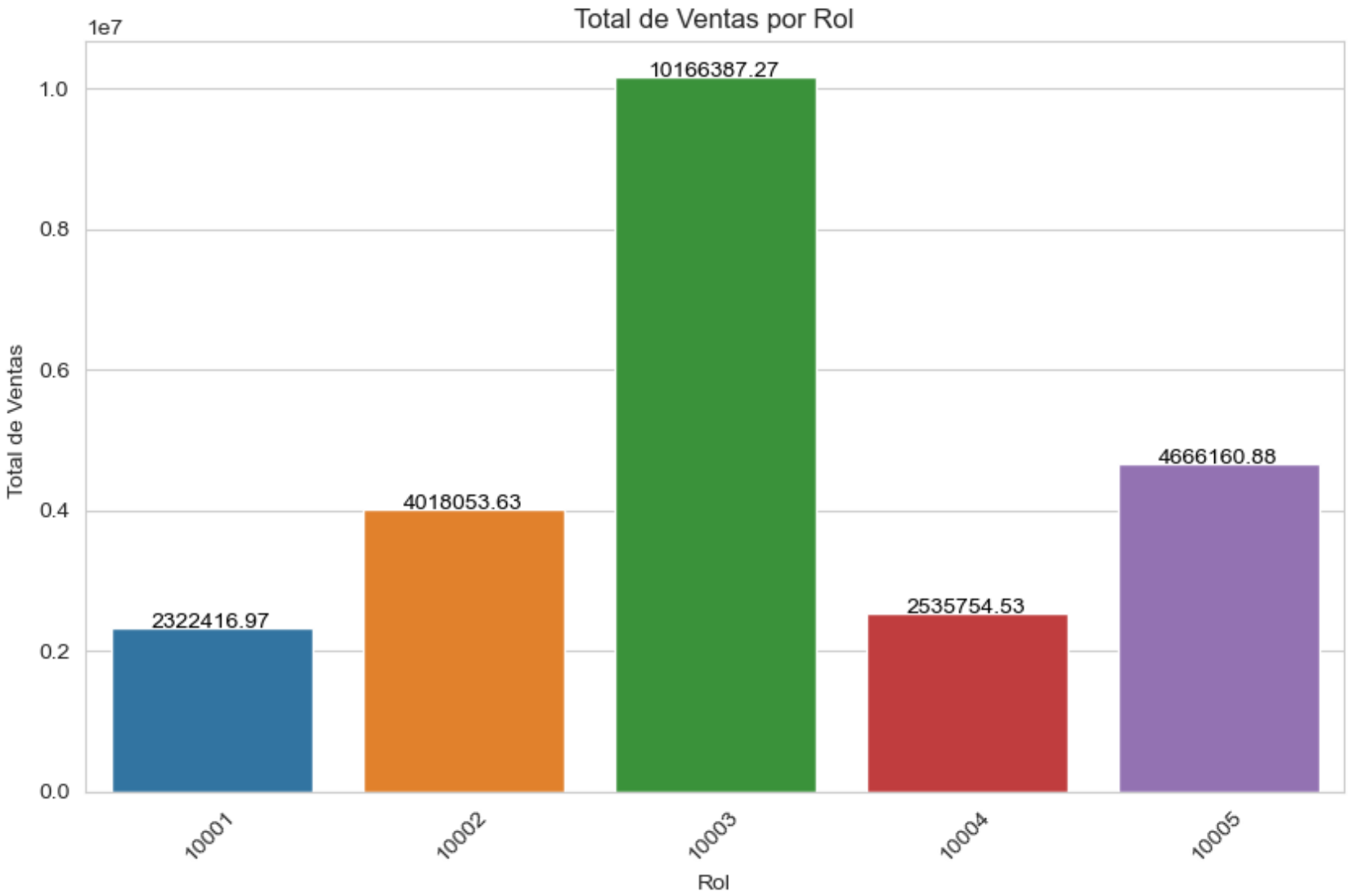
Preguntas de Negocio.

```
df_venta_cliente_usuario = df_venta_cliente.merge(df_usuario,
how='inner', on='idusuario')
df_venta_cliente_usuario =
df_venta_cliente_usuario.drop(['idcliente'], axis=1)
```

Preguntas de Negocio.

```
data_roles = pd.read_csv('dimRol.csv')

usuarios_roles = pd.merge(df_venta_cliente_usuario, data_roles,
how='left', on='idrol')

ventas_por_rol = usuarios_roles.groupby('idrol')
['total'].sum().reset_index()

plt.figure(figsize=(10, 6))
ax = sns.barplot(x='idrol', y='total',
data=ventas_por_rol.sort_values(by='total', ascending=False))
plt.xlabel('Rol')
plt.ylabel('Total de Ventas ')
plt.title('Total de Ventas por Rol')
plt.xticks(rotation=45)

for index, row in ventas_por_rol.iterrows():
    ax.text(index, row['total'], str(round(row['total'], 2)),
    color='black', ha="center")
plt.show()
```

Preguntas de Negocio.



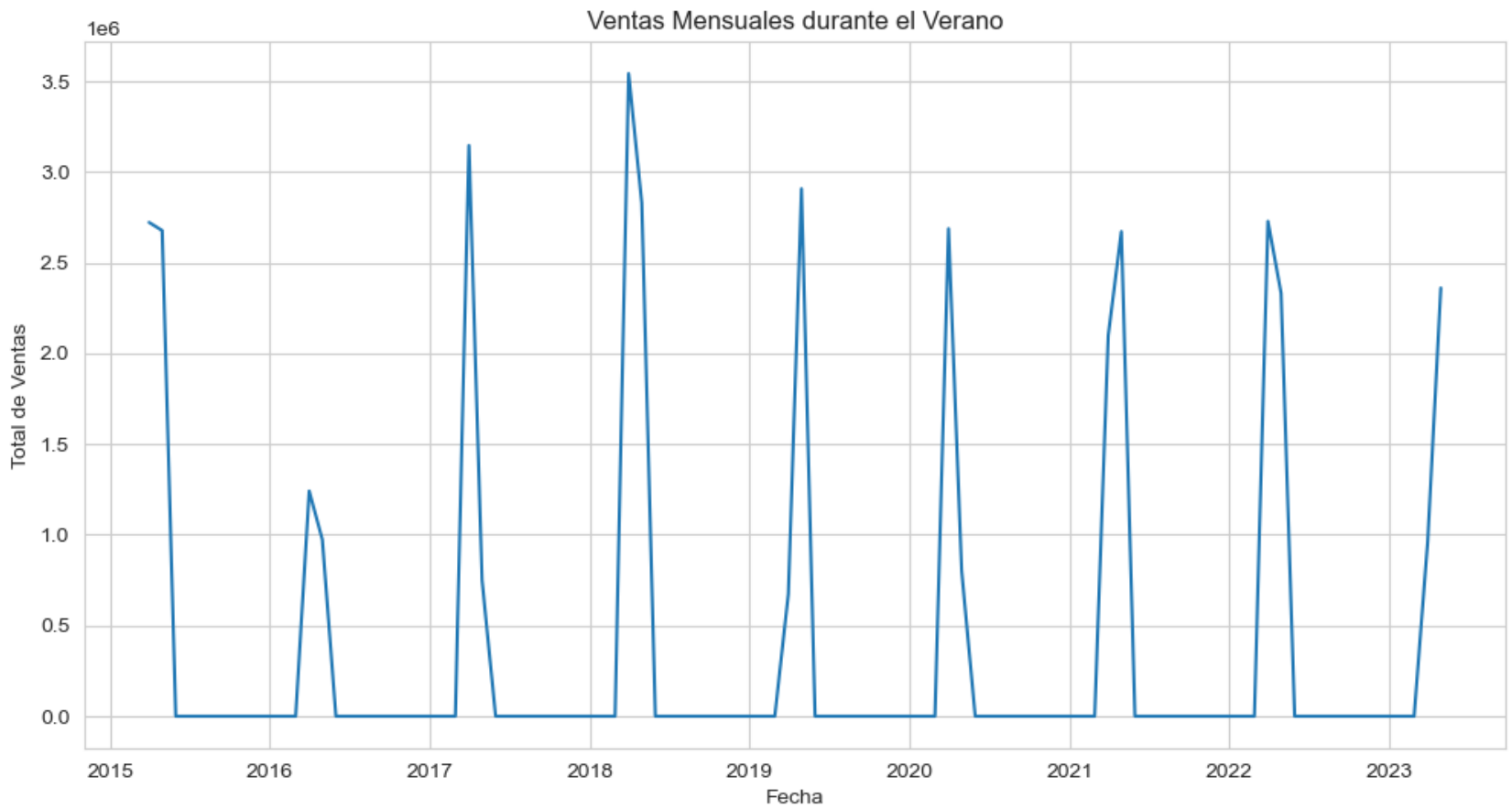
Pregunta No.4

¿Cómo varían las ventas en diferentes momentos del año, como durante las vacaciones o eventos especiales?

Preguntas de Negocio.

```
ventas_verano =
data_Detalles_Venta[(data_Detalles_Venta['fecha'].dt.month >= 3)
& (data_Detalles_Venta['fecha'].dt.month <= 4)]

ventas_verano_mensuales =
ventas_verano.groupby(pd.Grouper(key='fecha',
freq='M')).agg({'total': 'sum'}).reset_index()

plt.figure(figsize=(12, 6))
sns.lineplot(x='fecha', y='total', data=ventas_verano_mensuales)
plt.title('Ventas Mensuales durante el Verano')
plt.xlabel('Fecha')
plt.ylabel('Total de Ventas')
plt.show()
```

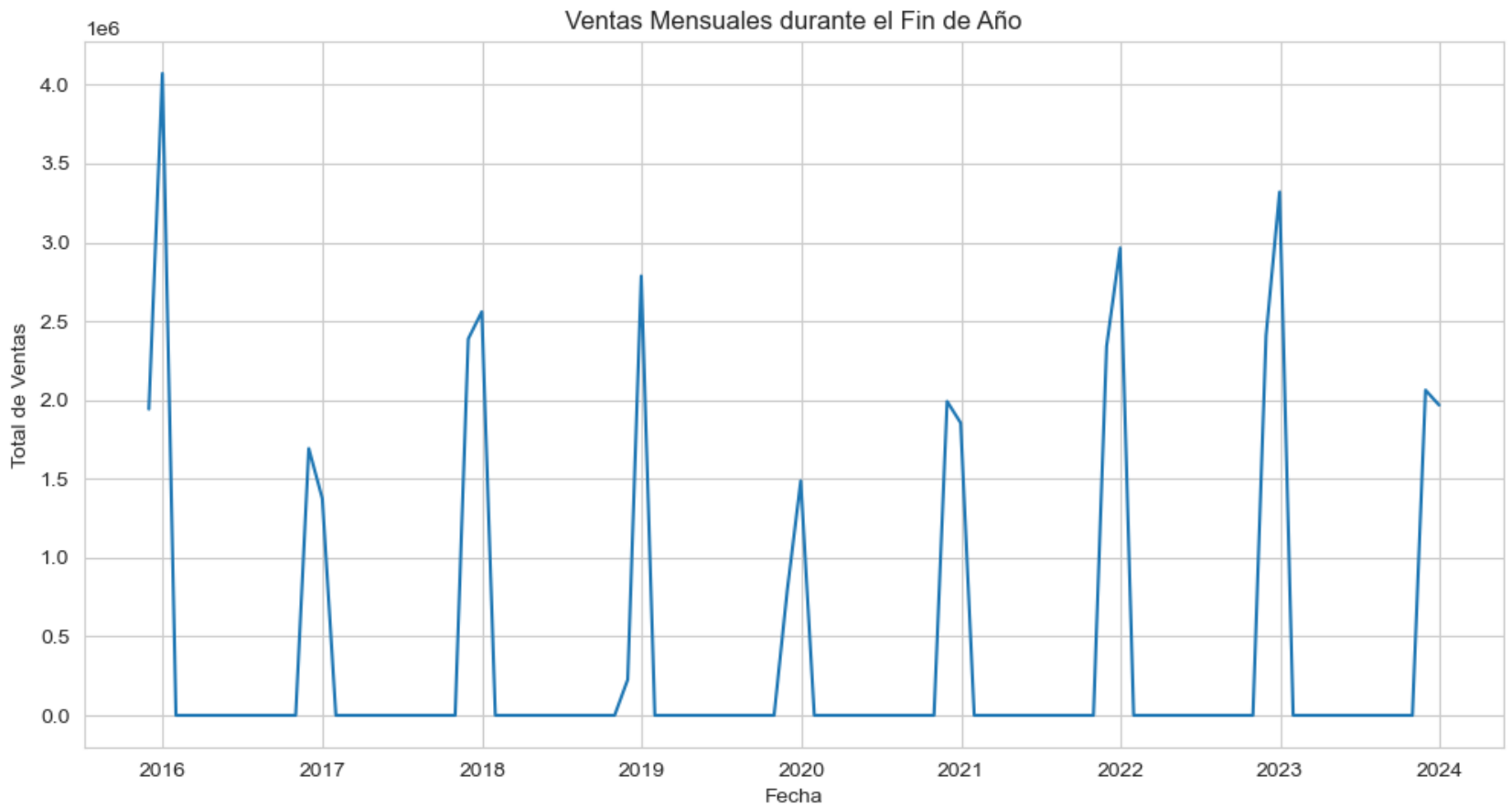
Preguntas de Negocio.



Preguntas de Negocio.

```
ventas_fin_de_anio =
data_Detalles_Venta[(data_Detalles_Venta['fecha'].dt.month >=
11) & (data_Detalles_Venta['fecha'].dt.month <= 12)]

ventas_fin_de_anio_mensuales =
ventas_fin_de_anio.groupby(pd.Grouper(key='fecha',
freq='M')).agg({'total': 'sum'}).reset_index()

plt.figure(figsize=(12, 6))
sns.lineplot(x='fecha', y='total',
data=ventas_fin_de_anio_mensuales)
plt.title('Ventas Mensuales durante el Fin de Año')
plt.xlabel('Fecha')
plt.ylabel('Total de Ventas')
plt.show()
```

Preguntas de Negocio.



Pregunta No.5

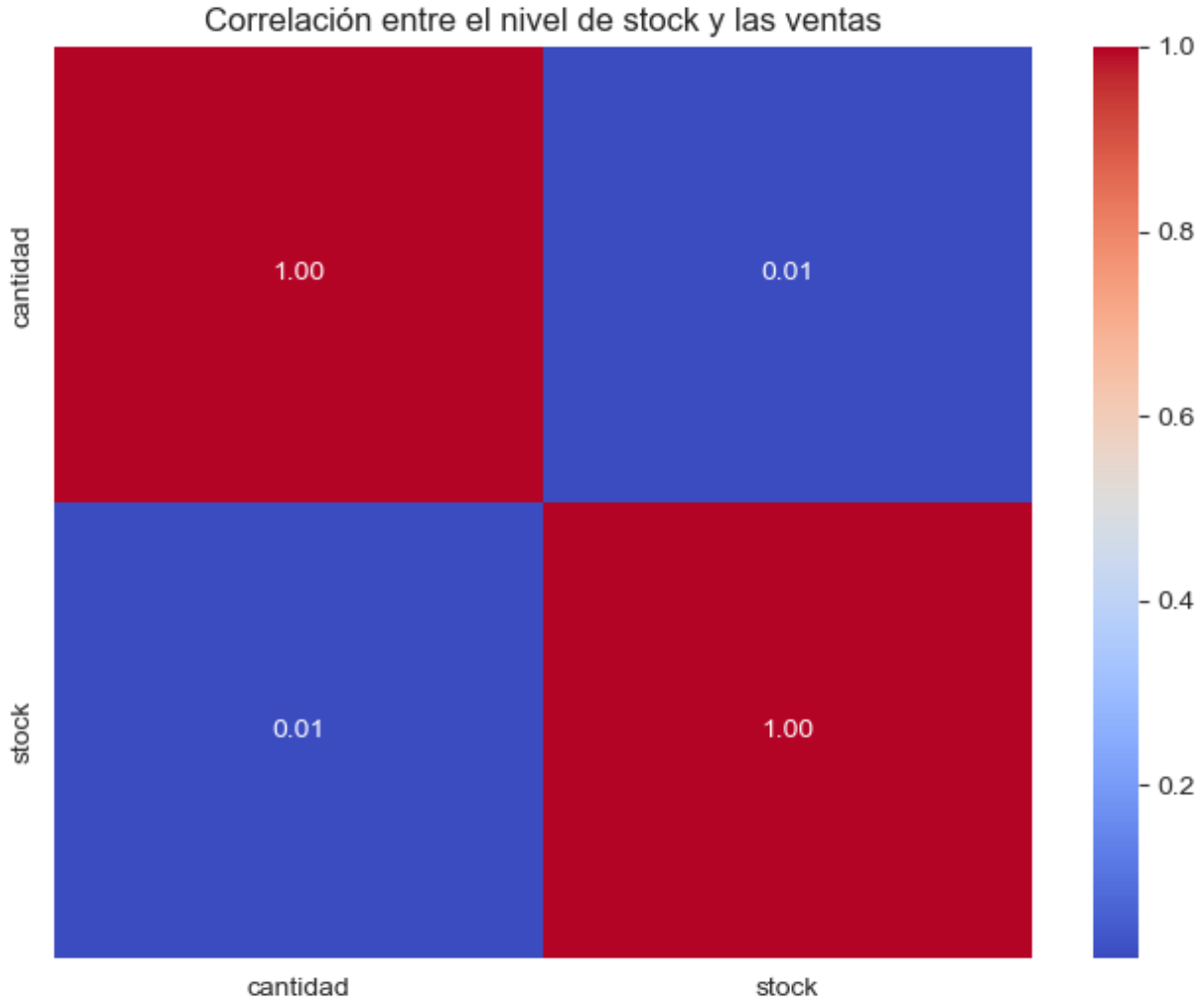
¿Cómo se correlaciona el nivel de stock de los artículos con las ventas realizadas?

¿Existen artículos que constantemente tienen un bajo nivel de stock pero alta demanda?

Preguntas de Negocio.

```
correlation = data_Detalles_Venta[['cantidad', 'stock']].corr()

low_stock_items =
data_Detalles_Venta[data_Detalles_Venta['stock'] < 10]

high_demand_items =
data_Detalles_Venta[data_Detalles_Venta['cantidad'] >
data_Detalles_Venta['cantidad'].mean()]

plt.figure(figsize=(8, 6))
sns.heatmap(correlation, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlación entre el nivel de stock y las ventas')
plt.show()
```

Preguntas de Negocio.



Preguntas de Negocio.

```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=low_stock_items, x='stock', y='cantidad',
hue='nombre')
plt.title('Artículos con bajo nivel de stock pero alta demanda')
plt.xlabel('Stock')
plt.ylabel('Cantidad Vendida')
plt.legend(title='Artículo', bbox_to_anchor=(1.05, 1), loc='upper
plt.show()
```

Preguntas de Negocio.