HITO 2 DEL 1º TRIMESTRE DE SISTEMAS DE GESTIÓN EMPRESARIAL

Jorge Bernal Barriga 20/11/2024

ÍNDICE

INTRODUCCIÓN

El presente proyecto tiene como objetivo desarrollar una solución informática para la gestión y análisis de datos relacionados con el consumo de alcohol y sus impactos en la salud, en respuesta a los requerimientos del Dr. Fernando. Este sistema ha sido diseñado para facilitar la interacción con una base de datos clínica, permitiendo realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) de manera eficiente, así como proporcionar herramientas avanzadas de análisis y visualización.

La aplicación integra una interfaz gráfica de usuario desarrollada en **Tkinter**, que brinda una experiencia intuitiva y funcional para la manipulación de datos. Asimismo, incorpora conexiones estables a una base de datos **MySQL** para garantizar la fiabilidad y el manejo de grandes volúmenes de información. Además, el sistema permite exportar consultas realizadas a un archivo **Excel**, facilitando así el análisis externo de los datos para una mejor comprensión y aprovechamiento de la información.

Por otro lado, el sistema ofrece capacidades de consulta y generación de gráficos para presentar de forma clara las correlaciones y patrones relevantes en los datos, como la relación entre el consumo de alcohol y problemas de salud específicos.

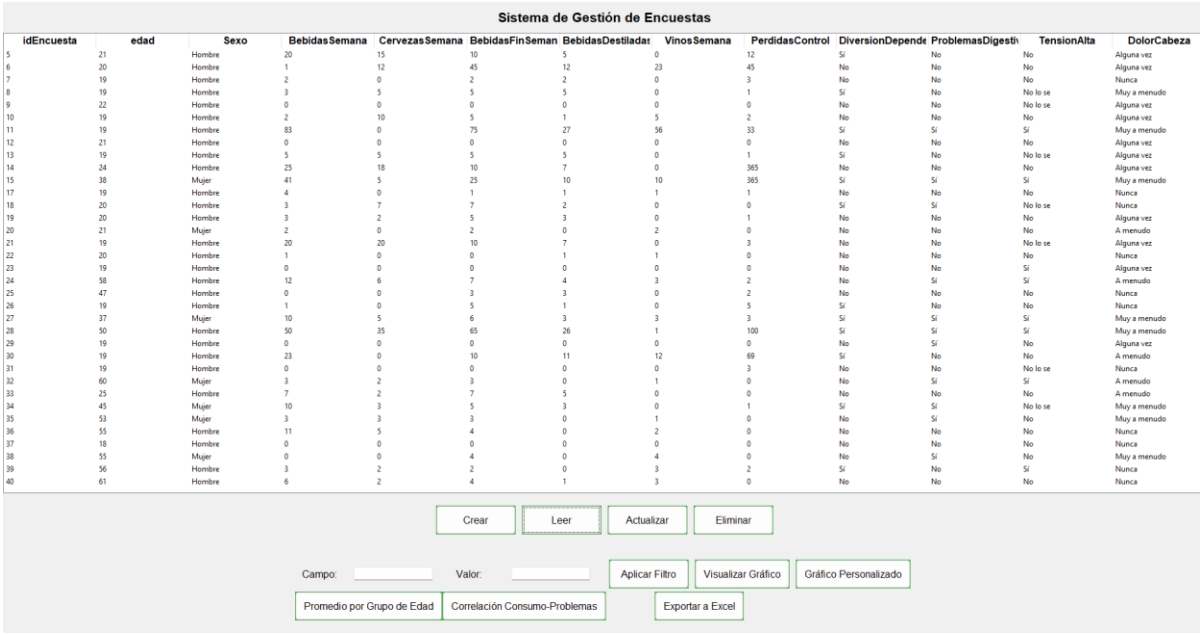


Ilustración 1:Interfaz gráfica

DESCRIPCIÓN DE LA INTERFAZ Y CONEXIÓN

Interfaz de Usuario

La interfaz de usuario ha sido desarrollada utilizando el módulo **Tkinter** de Python, el cual proporciona un entorno amigable, intuitivo y fácil de usar. La interfaz permite a los usuarios realizar todas las operaciones necesarias relacionadas con la gestión de los datos de encuestas sobre el consumo de alcohol y los indicadores de salud.

Entre los elementos principales de la interfaz, se incluyen:

- **Menús desplegables**: Para facilitar el acceso a diferentes funciones como la gestión de datos, consultas y generación de gráficos.
- **Botones interactivos**: Para ejecutar acciones específicas como agregar, modificar, eliminar o consultar registros.
- **Campos de entrada**: Que permiten ingresar la información requerida, como la edad, el consumo semanal de alcohol, problemas de salud reportados, entre otros.
- **Visualización de tablas**: Mediante widgets como `Treeview`, los datos se presentan en un formato tabular que permite ordenar y filtrar registros de manera sencilla.
- **Exportación a Excel**: Una funcionalidad adicional que permite guardar los datos consultados en un archivo Excel para su análisis externo.

A continuación, se presentan capturas de pantalla que ilustran la interfaz de usuario:

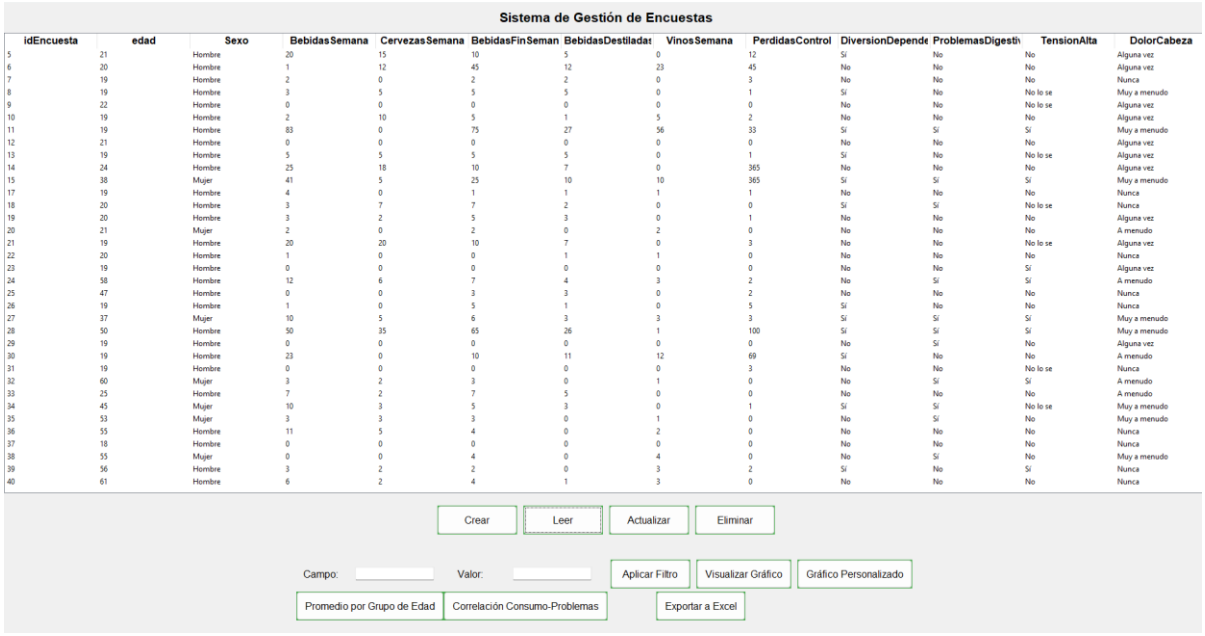


Ilustración 2: Pantalla principal

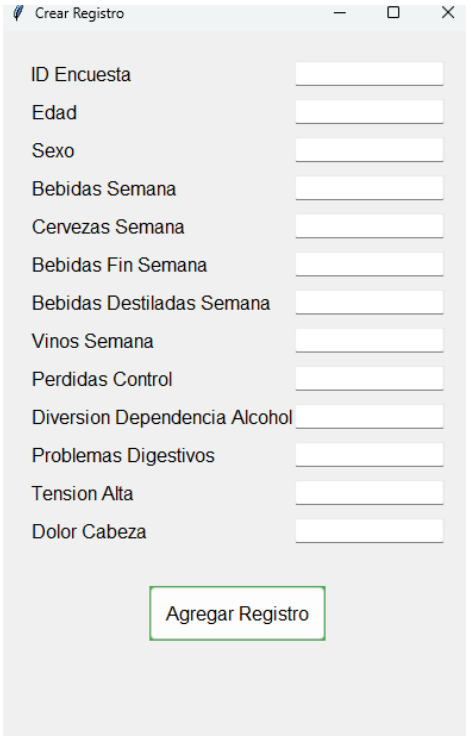


Ilustración 3: Formulario creación de encuestas

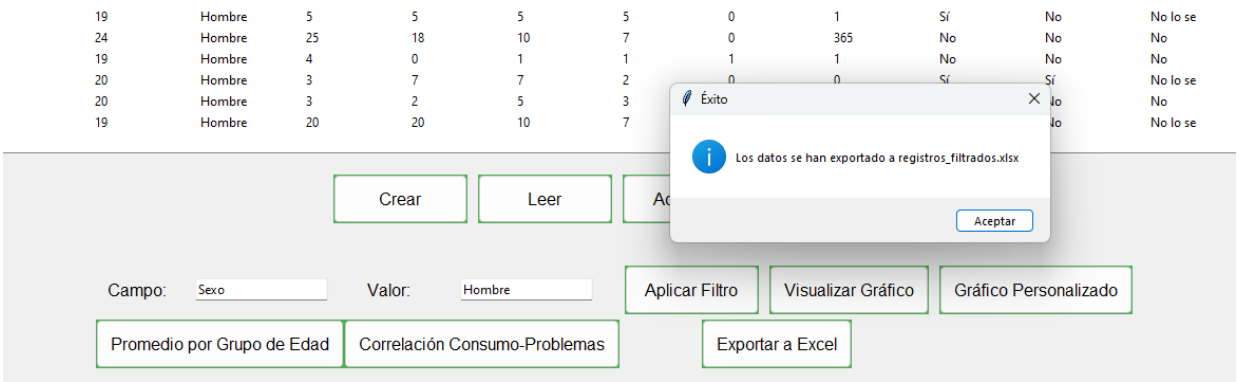


Ilustración 4: Tabla de datos y exportaciones

Conexión a la Base de Datos

La aplicación establece una conexión estable y segura con una base de datos **MySQL** utilizando la librería **pymysql**, que permite interactuar con la base de datos de forma eficiente. Todas las operaciones CRUD (Crear, Leer, Actualizar y Eliminar) se realizan de manera fluida y sin errores, asegurando la integridad de los datos.

Flujo de conexión

1. **Librería pymysql**: La conexión con MySQL se gestiona mediante esta librería, conocida por ser ligera y eficaz.
2. **Credenciales seguras**: Los parámetros de configuración, como usuario, contraseña, host y base de datos, están correctamente protegidos para garantizar la seguridad.
3. **Mensajes de estado**: La interfaz informa al usuario si la conexión ha sido exitosa o si se ha producido algún error, facilitando el diagnóstico de problemas.

Operaciones CRUD implementadas

- **Crear**: Inserción de nuevas encuestas mediante formularios en la interfaz gráfica.
- **Leer**: Recuperación de registros desde la base de datos y visualización en tablas interactivas.
- **Actualizar**: Modificación de datos existentes con validación previa para evitar inconsistencias.
- **Eliminar**: Eliminación de registros tras solicitar confirmación al usuario para prevenir errores accidentales.

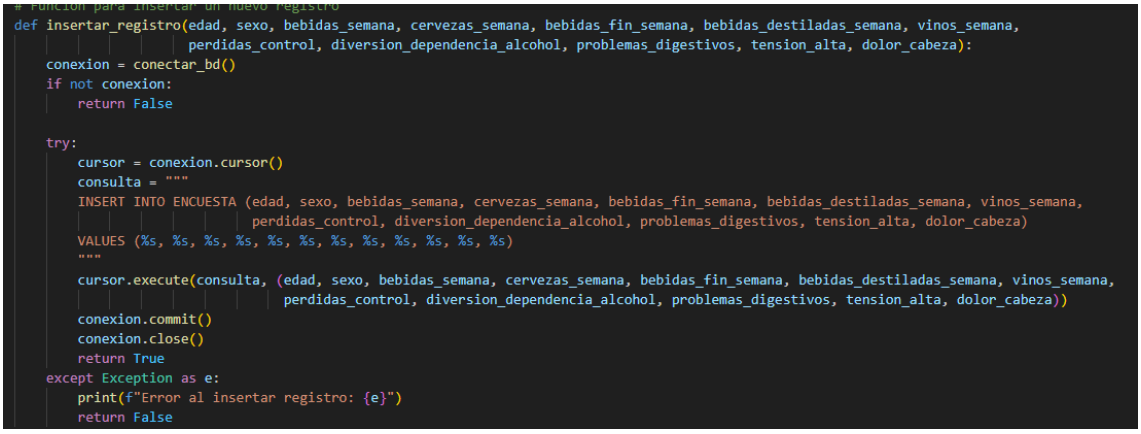
OPERACIONES CRUD

1. Crear (Insertar Registro)

Descripción: La operación de creación permite agregar un nuevo registro a la base de datos. El usuario ingresa los datos a través de un formulario en la interfaz gráfica y, al hacer clic en el botón "Agregar Registro", los datos se envían a la base de datos.

Lógica:

- El usuario abre una ventana secundaria para ingresar los datos del nuevo registro.
- Los datos se recogen de los campos de entrada y se pasan a la función insertar_registro.
- La función insertar_registro utiliza una consulta SQL parametrizada para insertar los datos en la tabla ENCUESTA



```
# Función para insertar un nuevo registro
def insertar_registro(edad, sexo, bebidas_semana, cervezas_semana, bebidas_fin_semana, bebidas_destiladas_semana, vinos_semana,
                      perdidas_control, diversion_dependencia_alcohol, problemas_digestivos, tension_alta, dolor_cabeza):
    conexion = conectar_bd()
    if not conexion:
        return False

    try:
        cursor = conexion.cursor()
        consulta = """
        INSERT INTO ENCUESTA (edad, sexo, bebidas_semana, cervezas_semana, bebidas_fin_semana, bebidas_destiladas_semana, vinos_semana,
                              perdidas_control, diversion_dependencia_alcohol, problemas_digestivos, tension_alta, dolor_cabeza)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
        """
        cursor.execute(consulta, (edad, sexo, bebidas_semana, cervezas_semana, bebidas_fin_semana, bebidas_destiladas_semana, vinos_semana,
                                  perdidas_control, diversion_dependencia_alcohol, problemas_digestivos, tension_alta, dolor_cabeza))
        conexion.commit()
        conexion.close()
        return True
    except Exception as e:
        print(f"Error al insertar registro: {e}")
        return False
```

2. Leer (Leer Registros)

Descripción: La operación de lectura permite obtener y mostrar los registros almacenados en la base de datos. Los registros se muestran en una tabla en la interfaz gráfica.

Lógica:

- Al iniciar la aplicación o al hacer clic en el botón "Leer", se llama a la función leer_registros.
- La función leer_registros ejecuta una consulta SQL para obtener todos los registros de la tabla ENCUESTA.

- Los registros obtenidos se muestran en la tabla de la interfaz gráfica

```python
def leer_registros(ordenar_por=None):
    conexion = conectar_bd()
    if not conexion:
        return []

    try:
        cursor = conexion.cursor()

        # Si se especifica un campo de ordenación, agregarlo a la consulta SQL
        if ordenar_por:
            consulta = f"SELECT * FROM ENCUESTA ORDER BY {ordenar_por}"
        else:
            consulta = "SELECT * FROM ENCUESTA"  # Si no se especifica, no ordena

        cursor.execute(consulta)
        registros = cursor.fetchall()
        conexion.close()
        return registros
    except Exception as e:
        print(f"Error al leer registros: {e}")
        return []
```

3. Actualizar (Actualizar Registro)

Descripción: La operación de actualización permite modificar un registro existente en la base de datos. El usuario selecciona un registro, abre una ventana secundaria para editar los datos y, al hacer clic en el botón "Actualizar Registro", los cambios se guardan en la base de datos.

Lógica:

- El usuario selecciona un registro y abre una ventana secundaria para editar los datos.
- Los datos modificados se recogen de los campos de entrada y se pasan a la función actualizar_registro.
- La función actualizar_registro utiliza una consulta SQL parametrizada para actualizar los datos en la tabla ENCUESTA.

```python
def actualizar_registro(id, edad, sexo, consumo_semanal, problemas_salud, perdidas_control):
    conexion = conectar_bd()
    if not conexion:
        return False

    try:
        cursor = conexion.cursor()
        consulta = "UPDATE ENCUESTA SET edad=%s, sexo=%s, consumo_semanal=%s, problemas_salud=%s, perdidas_control=%s
        cursor.execute(consulta, (edad, sexo, consumo_semanal, problemas_salud, perdidas_control, id))
        conexion.commit()
        conexion.close()
        return True
    except Exception as e:
        print(f"Error al actualizar registro: {e}")
        return False
```

4. Eliminar (Eliminar Registro)

Descripción: La operación de eliminación permite borrar un registro existente de la base de datos. El usuario selecciona un registro y, al hacer clic en el botón "Eliminar", el registro se elimina de la base de datos.

Lógica:

- El usuario selecciona un registro y hace clic en el botón "Eliminar".
- El ID del registro seleccionado se pasa a la función eliminar_registro.
- La función eliminar_registro utiliza una consulta SQL parametrizada para eliminar el registro de la tabla ENCUESTA.

```python
def eliminar_registro(id):
    conexion = conectar_bd()
    if not conexion:
        return False

    try:
        cursor = conexion.cursor()
        consulta = "DELETE FROM ENCUESTA WHERE id=%s"
        cursor.execute(consulta, (id,))
        conexion.commit()
        conexion.close()
        return True
    except Exception as e:
        print(f"Error al eliminar registro: {e}")
        return False
```

Conclusión

Estas operaciones CRUD (Crear, Leer, Actualizar, Eliminar) permiten gestionar los datos de la base de datos de manera eficiente y segura. La implementación de consultas parametrizadas y la gestión de mensajes de estado aseguran que la aplicación sea robusta y segura contra vulnerabilidades como las inyecciones SQL

CONSULTAS Y ORDENACIÓN DE DATOS

Descripción: La funcionalidad de consultas y ordenación de datos permite a los usuarios buscar y ordenar los registros de la base de datos según diferentes criterios. Esto facilita la visualización de subgrupos específicos y la organización de los datos de manera significativa.

1. Consultas Personalizadas
Descripción: Los usuarios pueden realizar consultas personalizadas especificando un campo y un valor. Esto permite filtrar los registros que coinciden con los criterios de búsqueda.

Lógica:

- El usuario ingresa el campo y el valor en los campos de entrada correspondientes.
- Al hacer clic en el botón "Aplicar Filtro", se llama a la función filtrar_registros.
- La función filtrar_registros filtra los registros según el campo y el valor especificados y actualiza la tabla con los resultados.

```
def filtrar_registros():
    campo = campo_filtro.get()
    valor = valor_filtro.get()
    if not campo or not valor:
        messagebox.showerror("Error", "Debe especificar un campo y un
        return

    def filtro(registro):
        campos = {
            "idEncuesta": 0,
            "edad": 1,
            "Sexo": 2,
            "BebidasSemana": 3,
            "CervezasSemana": 4,
            "BebidasFinSemana": 5,
            "BebidasDestiladasSemana": 6,
            "VinosSemana": 7,
            "PerdidasControl": 8,
            "DiversionDependenciaAlcohol": 9,
            "ProblemasDigestivos": 10,
            "TensionAlta": 11,
            "DolorCabeza": 12
        }
        indice = campos.get(campo)
        if indice is not None:
            return str(registro[indice]) == valor
        return False
```

2. Filtros Predefinidos

Descripción: Los usuarios pueden aplicar filtros predefinidos para visualizar subgrupos específicos, como encuestados con alta frecuencia de consumo de alcohol, personas que han perdido el control por el consumo de alcohol en más de 3 ocasiones, y pacientes que reportan problemas de salud específicos.

Lógica:

- Al hacer clic en uno de los botones de filtros predefinidos, se llama a la función aplicar_filtro_predefinido con el tipo de filtro correspondiente.
- La función aplicar_filtro_predefinido aplica el filtro y actualiza la tabla con los resultados.

```
def aplicar_filtro_predefinido(tipo):
    if tipo == "alta_frecuencia":
        actualizar_tabla(lambda registro: registro[3] > 10)  # BebidasSemana > 10
    elif tipo == "perdida_control":
        actualizar_tabla(lambda registro: registro[8] > 3)  # PerdidasControl > 3
    elif tipo == "problemas_salud":
        actualizar_tabla(lambda registro: registro[12] == 'Sí' or registro[11] == 'Sí')  # Problemas de
```

3.Ordenación de Resultados

Descripción: Los usuarios pueden ordenar los resultados por cualquier campo, como edad, sexo, consumo semanal, problemas de salud, etc. Esto facilita la organización de los datos de manera significativa.

Lógica:

- El usuario selecciona el campo por el cual desea ordenar los resultados en un menú desplegable.
- Al hacer clic en el botón "Ordenar", se llama a la función actualizar_tabla con el campo de ordenación correspondiente.
- La función actualizar_tabla ordena los registros según el campo seleccionado y actualiza la tabla con los resultados.

```
def actualizar_tabla(filtro=None, orden=None):
    global registros_filtrados
    try:
        for row in tree.get_children():
            tree.delete(row)
        registros = leer_registros()
        if filtro:
            registros = list(filter(filtro, registros))
        if orden:
            registros.sort(key=lambda x: x[orden])
        registros_filtrados = registros  # Almacenar registros filtrados
        for registro in registros:
            tree.insert("", "end", values=registro)
    except Exception as e:
        messagebox.showerror("Error", f"Error al actualizar la tabla: {e}")
```

EXPORTACIÓN DE RESULTADOS A EXCEL

Descripción: La funcionalidad de exportación a Excel permite a los usuarios guardar los resultados de las consultas en un archivo Excel. Esto facilita el análisis de los datos utilizando herramientas externas como Microsoft Excel.

Lógica:

- Los registros filtrados se almacenan en una variable global registros_filtrados.
- Al hacer clic en el botón "Exportar a Excel", se llama a la función exportar_a_excel.
- La función exportar_a_excel convierte los registros filtrados en un DataFrame de pandas y los guarda en un archivo Excel.

```
# Función para exportar registros filtrados a un archivo Excel
def exportar_a_excel():
    if not registros_filtrados:
        messagebox.showerror("Error", "No hay datos para exportar")
        return

    df = pd.DataFrame(registros_filtrados, columns=["ID Encuesta", "Edad", "Sexo", "Bebidas Semana", "Ce
    df.to_excel("registros_filtrados.xlsx", index=False)
    messagebox.showinfo("Éxito", "Los datos se han exportado a registros_filtrados.xlsx")
```

Implementación en la Interfaz Gráfica

Para integrar esta funcionalidad en la interfaz gráfica, hemos añadido un botón "Exportar a Excel" que llama a la función exportar_a_excel.

VISUALIZACIÓN DE DATOS EN GRÁFICOS

Descripción: La funcionalidad de visualización de datos en gráficos permite a los usuarios generar gráficos a partir de los registros filtrados. Los gráficos disponibles incluyen gráficos de barras, gráficos de pastel y gráficos de dispersión.

1. Gráfico de Barras

Descripción: El gráfico de barras muestra el consumo de bebidas por edad. Este tipo de gráfico es útil para visualizar la distribución del consumo de bebidas en diferentes grupos de edad.

Lógica:

- Se recogen las edades y el consumo de bebidas por semana de los registros filtrados.
- Se genera un gráfico de barras utilizando matplotlib.

```
def visualizar_grafico():
    if not registros_filtrados:
        messagebox.showerror("Error", "No hay datos para mostrar en el gráfico")
        return

    edades = [registro[1] for registro in registros_filtrados]
    bebidas_semana = [registro[3] for registro in registros_filtrados]

    plt.bar(edades, bebidas_semana)
    plt.xlabel('Edad')
    plt.ylabel('Bebidas por Semana')
    plt.title('Consumo de Bebidas por Edad')
    plt.show()
```

2. Gráfico de Pastel

Descripción: El gráfico de pastel muestra la distribución por sexo de los encuestados. Este tipo de gráfico es útil para visualizar la proporción de hombres y mujeres en la muestra de datos.

Lógica:

- Se recogen los datos de sexo de los registros filtrados.
- Se genera un gráfico de pastel utilizando matplotlib.

```
def visualizar_grafico_personalizado():
    if not registros_filtrados:
        messagebox.showerror("Error", "No hay datos para mostrar en el gráfico")
        return

    # Gráfico de barras
    edades = [registro[1] for registro in registros_filtrados]
    bebidas_semana = [registro[3] for registro in registros_filtrados]

    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.bar(edades, bebidas_semana, color='skyblue')
    plt.xlabel('Edad')
    plt.ylabel('Bebidas por Semana')
    plt.title('Consumo de Bebidas por Edad')

    # Gráfico de pastel
    sexo = [registro[2] for registro in registros_filtrados]
    sexo_counts = {s: sexo.count(s) for s in set(sexo)}

    plt.subplot(1, 2, 2)
    plt.pie(sexo_counts.values(), labels=sexo_counts.keys(), autopct='%1.1f%%', startangle=140)
    plt.title('Distribución por Sexo')

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

3. Gráfico de Dispersión

Descripción: El gráfico de dispersión muestra la correlación entre el consumo de alcohol y los problemas digestivos. Este tipo de gráfico es útil para identificar posibles relaciones entre el consumo de alcohol y los problemas de salud.

Lógica:

- Se recogen los datos de consumo de bebidas por semana y la presencia de problemas digestivos de los registros filtrados.
- Se genera un gráfico de dispersión utilizando matplotlib.

```
def correlacion_consumo_problemas():
    if not registros_filtrados:
        messagebox.showerror("Error", "No hay datos para mostrar en el gráfico")
        return

    bebidas_semana = [registro[3] for registro in registros_filtrados]
    problemas_digestivos = [1 if registro[10] == 'Sí' else 0 for registro in registros_filtrados]

    plt.scatter(bebidas_semana, problemas_digestivos, color='red')
    plt.xlabel('Bebidas por Semana')
    plt.ylabel('Problemas Digestivos (1=Sí, 0=No)')
    plt.title('Correlación entre Consumo de Alcohol y Problemas Digestivos')
    plt.show()
```

Implementación en la Interfaz Gráfica

Para integrar estas funcionalidades en la interfaz gráfica, hemos añadido botones que llaman a las funciones correspondientes para generar los gráficos.

```
# Botón para visualizar gráficos
btn_grafico = ttk.Button(frame_filtro, text="Visualizar Gráfico", command=visualizar_grafico)
btn_grafico.grid(row=0, column=5, padx=5)

# Botón para visualizar gráficos personalizados
btn_grafico_personalizado = ttk.Button(frame_filtro, text="Gráfico Personalizado", command=visualizar_gr
btn_grafico_personalizado.grid(row=0, column=6, padx=5)

# Botón para promedio por grupo de edad
btn_promedio_edad = ttk.Button(frame_filtro, text="Promedio por Grupo de Edad", command=promedio_por_gru
btn_promedio_edad.grid(row=1, column=0, columnspan=2, pady=5)

# Botón para correlación entre consumo de alcohol y problemas de salud
btn_correlacion = ttk.Button(frame_filtro, text="Correlación Consumo-Problemas", command=correlacion_con
btn_correlacion.grid(row=1, column=2, columnspan=2, pady=5)
```

BLIBLIOGRAFÍA

- *Matplotlib pyplot*. (s/f). W3schools.com. Recuperado el 18 de noviembre de 2024, de https://www.w3schools.com/python/matplotlib_pyplot.asp

- *Python tkinter*. (2017, junio 17). GeeksforGeeks. https://www.geeksforgeeks.org/python-gui-tkinter/

- Rossum, G. (1999). *Python Tutorial*. https://www.w3schools.com/python/