20-11-2024

Hito2_1T_SGE_AlejandroPawlukiewicz

CAMPUSFP

Contenido

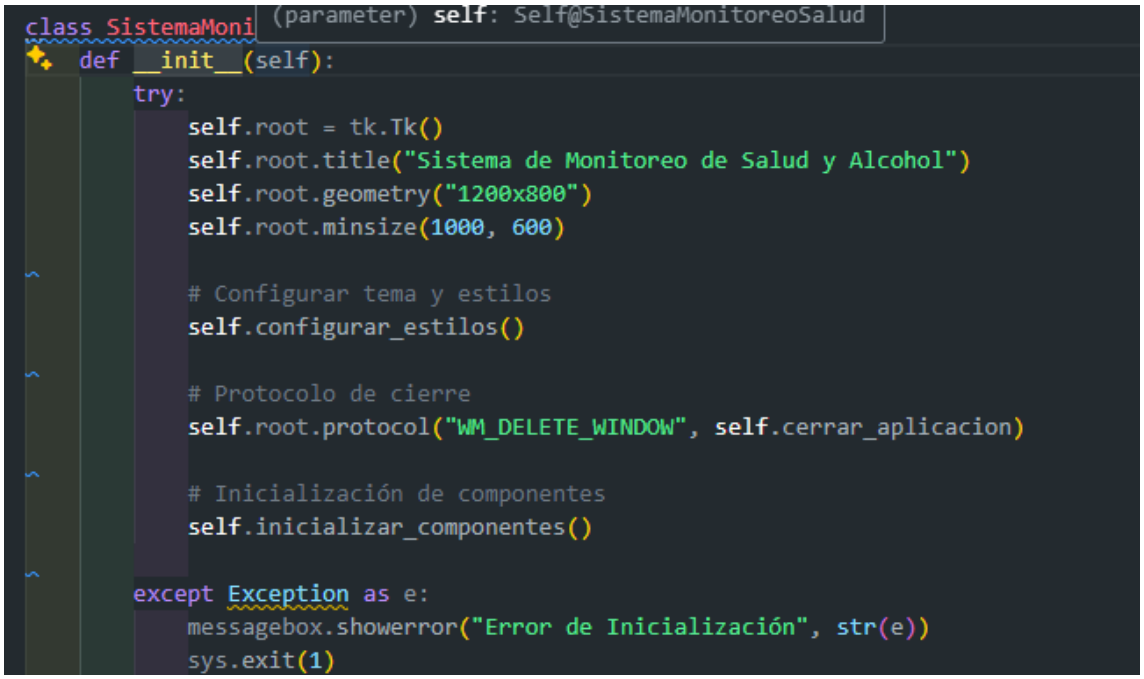
Objetivo del Proyecto

El objetivo principal del proyecto es desarrollar un sistema que permita monitorear y analizar el consumo de alcohol y su relación con problemas de salud específicos, como dolores de cabeza y presión alta, en pacientes. El sistema debe proporcionar funcionalidades para registrar, buscar, actualizar y eliminar datos de pacientes, así como visualizar los resultados de las consultas en gráficos y exportar los datos a un archivo Excel para su análisis.

Descripción de los Archivos y Funcionalidades

main.py

- **Objetivo**: Este archivo es el punto de entrada del sistema. Inicializa la aplicación, configura los estilos, maneja la conexión a la base de datos y coordina las interacciones entre los diferentes componentes del sistema.
- **Clases y Métodos**:
  - SistemaMonitoreoSalud: Clase principal que gestiona la interfaz gráfica y la lógica del sistema.
    - __init__: Inicializa la ventana principal y configura los estilos. Referencia: SistemaMonitoreoSalud.__init__.



  - configurar_estilos: Define los estilos y colores utilizados en la interfaz.
    Referencia: SistemaMonitoreoSalud.configurar_estilos.

main.py 9+ | interfazusuario.py 9+

main.py > SistemaMonitoreoSalud > configurar_estilos

```python
class SistemaMonitoreoSalud:

    def configurar_estilos(self):
        style = ttk.Style()

        # Colores principales
        self.COLOR_PRIMARY = "#2196F3"
        self.COLOR_SECONDARY = "#64B5F6"
        self.COLOR_BACKGROUND = "#F5F5F5"
        self.COLOR_SUCCESS = "#4CAF50"
        self.COLOR_WARNING = "#FFC107"
        self.COLOR_ERROR = "#F44336"
        self.COLOR_TEXT = "#212121"
        self.COLOR_ACCENT = "#1976D2"

        # Configuración general
        self.root.configure(bg=self.COLOR_BACKGROUND)

        # Estilos personalizados
        style.configure(".",
            font=("Segoe UI", 10),
            background=self.COLOR_BACKGROUND,
            foreground=self.COLOR_TEXT
        )

        style.configure("Title.TLabel",
            font=("Segoe UI", 24, "bold"),
            foreground=self.COLOR_PRIMARY,
            padding=(0, 10)
        )

        style.configure("Custom.TLabelframe",
            background=self.COLOR_BACKGROUND,
            padding=15,
            relief="solid"
        )

        style.configure("Primary.TButton",
            font=("Segoe UI", 10, "bold"),
            background=self.COLOR_PRIMARY,
            foreground="black",
            padding=(20, 10)
        )

        style.configure("Secondary.TButton",
            font=("Segoe UI", 10),
            background=self.COLOR_SECONDARY,
            foreground="black",
            padding=(15, 8)
        )

        style.configure("Custom.TEntry",
            padding=8,
            fieldbackground="white"
        )
```

- inicializar_componentes: Inicializa los componentes de la interfaz y establece las conexiones a la base de datos.
  Referencia: SistemaMonitoreoSalud.inicializar_componentes.

```python
def inicializar_componentes(self):
    try:
        print("Inicializando visualizador de gráficas...")
        self.visualizador = VisualizadorGraficas()

        print("Inicializando conexión BD...")
        self.conexion = ConexionBD()
        if not hasattr(self.conexion, 'conexion') or not self.conexion.conexion.is_connected():
            raise DatabaseError("No se pudo establecer la conexión con la base de datos")

        print("Inicializando consultas...")
        self.consultas = ConsultasEncuesta(self.conexion)

        print("Inicializando interfaz...")
        self.interfaz = InterfazSaludAlcohol(self.root)

        # Conectar callbacks
        self.interfaz.registrar_callback(self.registrar_paciente)
        self.interfaz.visualizar_callback(self.mostrar_graficas)
        self.interfaz.estadisticas_callback(self.mostrar_estadisticas)
        self.interfaz.exportar_callback(self.exportar_a_excel)
        self.interfaz.eliminar_callback(self.eliminar_paciente)
        self.interfaz.registrar_buscar_callback(self.buscar_paciente)
        self.interfaz.registrar_actualizar_callback(self.actualizar_paciente)
        self.cargar_datos_iniciales()

        # Cargar listado de pacientes
        pacientes = self.consultas.obtener_listado_pacientes()
        self.interfaz.actualizar_listado_pacientes(pacientes)

    except Exception as e:
        raise Exception(f"Error al inicializar componentes: {str(e)}")
```

- cerrar_aplicacion: Maneja el cierre de la aplicación.
  Referencia: SistemaMonitoreoSalud.cerrar_aplicacion.

```python
def cerrar_aplicacion(self):
    """Cierra la aplicación y la conexión a la base de datos"""
    if messagebox.askokcancel("Salir", "¿Desea salir de la aplicación?"):
        if hasattr(self, 'conexion'):
            self.conexion.cerrar_conexion()
        self.root.destroy()
```

- registrar_paciente: Registra un nuevo paciente en la base de datos.
  Referencia: SistemaMonitoreoSalud.registrar_paciente.

```python
def registrar_paciente(self, datos_paciente):
    """Registra un nuevo paciente en la base de datos"""
    try:
        # Insertar nuevo registro
        self.consultas.insertar_encuesta(datos_paciente)
        messagebox.showinfo("Éxito", "Paciente registrado correctamente")

        # Actualizar estadísticas y gráficas
        self.mostrar_estadisticas()
        self.mostrar_graficas()

        return True

    except Exception as e:
        messagebox.showerror("Error", f"Error al registrar paciente: {str(e)}")
        return False
```

- buscar_paciente: Busca un paciente por su ID.
  Referencia: SistemaMonitoreoSalud.buscar_paciente.

```python
def buscar_paciente(self, id_paciente):
    """Busca un paciente por su ID y muestra sus datos"""
    try:
        datos = self.consultas.obtener_paciente_por_id(id_paciente)
        if datos:
            self.interfaz.mostrar_datos_paciente(datos)
            messagebox.showinfo("Éxito", "Paciente encontrado. Puede proceder a modificar los datos.")
        else:
            messagebox.showwarning("No encontrado", "No se encontró ningún paciente con ese ID")
    except Exception as e:
        messagebox.showerror("Error", f"Error al buscar paciente: {str(e)}")
```

- actualizar_paciente: Actualiza los datos de un paciente existente.
  Referencia: SistemaMonitoreoSalud.actualizar_paciente.

```python
def actualizar_paciente(self, datos):
    """Actualiza los datos de un paciente existente"""
    try:
        self.consultas.actualizar_paciente(datos)
        messagebox.showinfo("Éxito", "Paciente actualizado correctamente")
    except Exception as e:
        messagebox.showerror("Error", f"Error al actualizar paciente: {str(e)}")
```

- eliminar_paciente: Elimina un paciente y actualiza el listado.
  Referencia: SistemaMonitoreoSalud.eliminar_paciente.

```python
def eliminar_paciente(self, id_paciente):
    """
    Elimina un paciente y actualiza el listado
    Args:
        id_paciente: ID del paciente a eliminar
    """
    try:
        self.consultas.eliminar_paciente(id_paciente)
        messagebox.showinfo("Éxito", "Paciente eliminado correctamente")
        # Actualizar el listado
        pacientes = self.consultas.obtener_listado_pacientes()
        self.interfaz.actualizar_listado_pacientes(pacientes)
    except Exception as e:
        messagebox.showerror("Error", f"No se pudo eliminar el paciente: {str(e)}")
```

- mostrar_graficas: Muestra las gráficas basadas en los datos de las consultas.
  Referencia: SistemaMonitoreoSalud.mostrar_graficas.

```python
def mostrar_graficas(self):
    """Muestra las gráficas basadas en el tipo seleccionado"""
    try:
        tipo_grafica = self.interfaz.tipo_grafica.get()
        datos = self._obtener_datos_grafica(tipo_grafica)

        if datos is not None and not datos.empty:
            fig = self._crear_grafica(tipo_grafica, datos)
            if fig:
                self.interfaz.actualizar_grafico(fig)
            else:
                messagebox.showinfo("Info", "No se pudo crear la gráfica")
        else:
            messagebox.showinfo("Info", "No hay datos disponibles para graficar")

    except Exception as e:
        print(f"Error en mostrar_graficas: {str(e)}")
        messagebox.showerror("Error", f"Error al mostrar gráfica: {str(e)}")
```

- mostrar_estadisticas: Muestra las estadísticas de consumo de alcohol y problemas de salud.
  Referencia: SistemaMonitoreoSalud.mostrar_estadisticas.

```python
def mostrar_estadisticas(self):
    """Muestra las estadísticas de consumo y registros recientes"""
    try:
        # Obtener estadísticas básicas
        stats = self.consultas.obtener_estadisticas_consumo()
        if stats is None or stats.empty:
            raise ValueError("No hay datos disponibles")

        # Obtener registros recientes
        registros_recientes = self.consultas.obtener_registros_recientes()

        # Obtener datos de alto consumo
        alto_consumo = self.consultas.filtrar_alto_consumo()

        # Actualizar la interfaz con todos los datos
        self.interfaz.actualizar_estadisticas(stats, alto_consumo)
        if not registros_recientes.empty:
            self.interfaz.actualizar_registros_recientes(registros_recientes)

    except Exception as e:
        messagebox.showerror("Error", f"Error al mostrar estadísticas: {str(e)}")
```

- exportar_a_excel: Exporta los datos a un archivo Excel. Referencia: SistemaMonitoreoSalud.exportar_a_excel.

```python
def exportar_a_excel(self, tipo_exportacion):
    """Exporta los datos a un archivo Excel"""
    try:
        # Obtener datos actuales
        datos = self.consultas.obtener_estadisticas_consumo()
        if datos is None or datos.empty:
            messagebox.showwarning("Advertencia", "No hay datos para exportar")
            return

        # Pedir al usuario la ubicación para guardar el archivo
        from tkinter import filedialog
        filename = filedialog.asksaveasfilename(
            defaultextension=".xlsx",
            filetypes=[("Excel files", "*.xlsx"), ("All files", "*.*")],
            title="Guardar como Excel"
        )

        if filename:
            # Exportar utilizando el método de consultas
            if self.consultas.exportar_a_excel(datos, filename):
                messagebox.showinfo("Éxito", "Datos exportados correctamente")
            else:
                messagebox.showerror("Error", "No se pudo exportar el archivo")

    except Exception as e:
        messagebox.showerror("Error", f"Error al exportar: {str(e)}")
        print(f"Error detallado en exportación: {str(e)}")
```

- cargar_datos_iniciales: Carga los datos iniciales y actualiza la interfaz.
  Referencia: SistemaMonitoreoSalud.cargar_datos_iniciales.

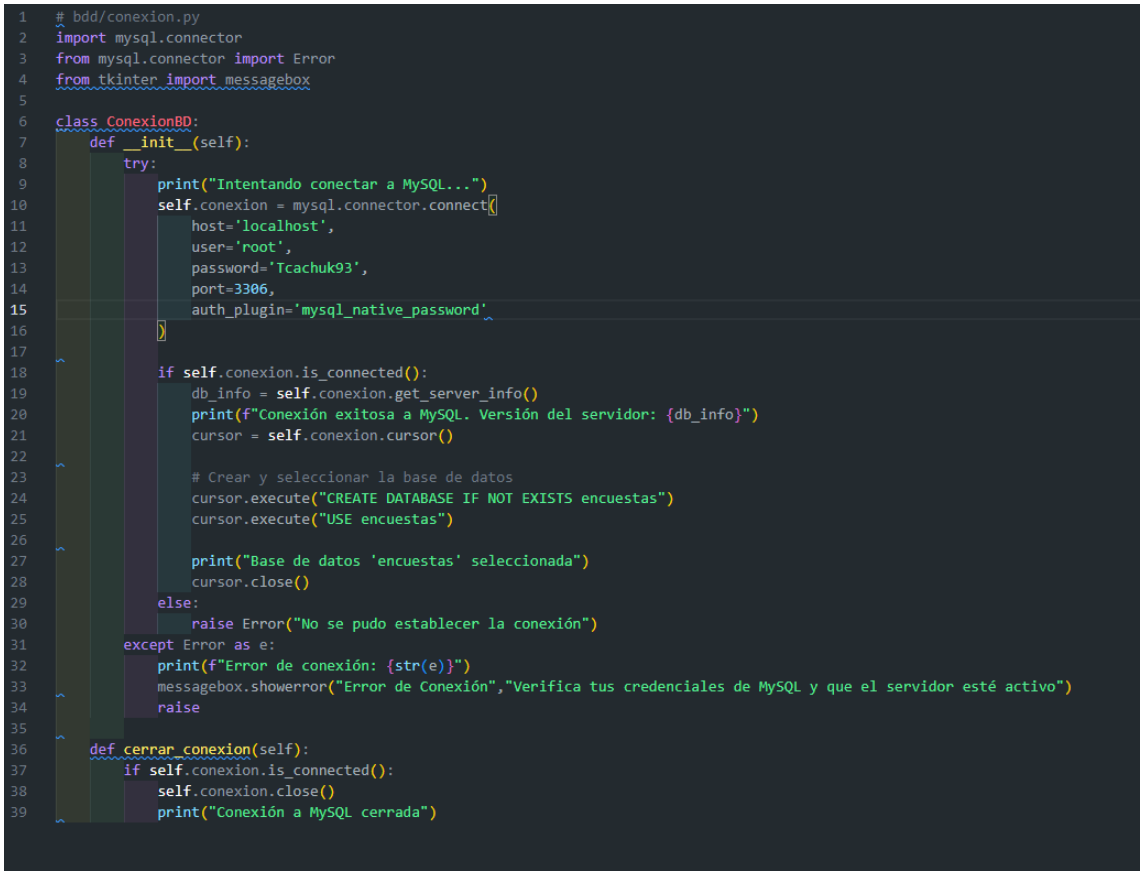
```python
def cargar_datos_iniciales(self):
    """Carga los datos iniciales y actualiza la interfaz"""
    try:
        print("Cargando datos iniciales...")
        stats = self.consultas.obtener_estadisticas_consumo()
        alto_consumo = self.consultas.filtrar_alto_consumo()

        if stats is not None and not stats.empty:
            self.interfaz.actualizar_estadisticas(stats, alto_consumo)
            self.mostrar_graficas()
        else:
            print("No hay datos iniciales disponibles")

    except Exception as e:
        print(f"Error al cargar datos iniciales: {str(e)}")
```

- main: Función principal que inicia la aplicación y maneja las excepciones. Referencia: main.

```python
def main():
    try:
        print("=== Iniciando Sistema de Monitoreo de Salud ===")

        # Verificar conexión BD primero
        print("Verificando conexión a base de datos...")
        conexion = ConexionBD()
        if not conexion.conexion.is_connected():
            raise DatabaseError("No se pudo establecer la conexión con la base de datos")
        conexion.cerrar_conexion()  # Cerrar conexión de prueba
        print("Conexión a base de datos establecida correctamente")

        # Iniciar aplicación
        print("Iniciando aplicación...")
        app = SistemaMonitoreoSalud()
        print("Sistema inicializado correctamente")

        # Ejecutar el bucle principal
        app.root.mainloop()

    except DatabaseError as e:
        messagebox.showerror("Error de Base de Datos", str(e))
        print(f"Error de base de datos: {str(e)}")
        sys.exit(1)
    except Exception as e:
        messagebox.showerror("Error Fatal", str(e))
        print(f"Error fatal: {str(e)}")
        sys.exit(1)
    finally:
        print("=== Finalizando Sistema de Monitoreo de Salud ===")

if __name__ == "__main__":
    main()
```

bdd/conexion.py

- **Objetivo**: Gestiona la conexión a la base de datos.
- **Clases y Métodos**:
  - ConexionBD: Clase que establece y cierra la conexión con la base de datos. Referencia: ConexionBD.



```python
# bdd/conexion.py
import mysql.connector
from mysql.connector import Error
from tkinter import messagebox

class ConexionBD:
    def __init__(self):
        try:
            print("Intentando conectar a MySQL...")
            self.conexion = mysql.connector.connect(
                host='localhost',
                user='root',
                password='Tcachuk93',
                port=3306,
                auth_plugin='mysql_native_password'
            )

            if self.conexion.is_connected():
                db_info = self.conexion.get_server_info()
                print(f"Conexión exitosa a MySQL. Versión del servidor: {db_info}")
                cursor = self.conexion.cursor()

                # Crear y seleccionar la base de datos
                cursor.execute("CREATE DATABASE IF NOT EXISTS encuestas")
                cursor.execute("USE encuestas")

                print("Base de datos 'encuestas' seleccionada")
                cursor.close()
            else:
                raise Error("No se pudo establecer la conexión")
        except Error as e:
            print(f"Error de conexión: {str(e)}")
            messagebox.showerror("Error de Conexión","Verifica tus credenciales de MySQL y que el servidor esté activo")
            raise

    def cerrar_conexion(self):
        if self.conexion.is_connected():
            self.conexion.close()
            print("Conexión a MySQL cerrada")
```

bdd/consultas.py

- **Objetivo**: Realiza las consultas a la base de datos.
- **Clases y Métodos**:
  - ConsultasEncuesta: Clase que contiene métodos para insertar, actualizar, eliminar y consultar datos de la base de datos. Referencia: ConsultasEncuesta.
    - insertar_encuesta: Inserta un nuevo registro en la base de datos. Referencia: ConsultasEncuesta.insertar_encuesta.

```
bdd > consultas.py > ConsultasEncuesta > insertar_encuesta
class ConsultasEncuesta:
    def insertar_encuesta(self, datos):
        query = """
        INSERT INTO encuesta (
            idEncuesta, edad, Sexo,
            BebidasSemana, CervezasSemana, BebidasFinSemana,
            BebidasDestiladasSemana, VinosSemana,
            PerdidasControl, DiversionDependenciaAlcohol,
            ProblemasDigestivos, TensionAlta, DolorCabeza
        ) VALUES (
            %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s
        )
        """
        try:
            cursor = self.conexion.conexion.cursor()

            # Obtener siguiente ID
            cursor.execute("SELECT MAX(idEncuesta) FROM encuesta")
            max_id = cursor.fetchone()[0] or 0
            nuevo_id = max_id + 1

            # Convertir valores Sí/No a 1/0
            def convertir_si_no(valor):
                return 1 if valor.lower() == 'sí' else 0

            valores = (
                nuevo_id,
                int(datos['edad']),
                datos['sexo'],
                float(datos['bebidas_semana']),
                float(datos['cervezas']),
                float(datos['finde']),
                float(datos['destiladas']),
                float(datos['vinos']),
                convertir_si_no(datos['perdidas_control']),
                convertir_si_no(datos['diversion_alcohol']),
                convertir_si_no(datos['problemas_digestivos']),
                convertir_si_no(datos['tension_alta']),
                datos['dolor_cabeza']
            )

            cursor.execute(query, valores)
            self.conexion.conexion.commit()
            return True

        except Error as e:
            self.conexion.conexion.rollback()
            raise Exception(f"Error al insertar datos: {str(e)}")
```

```
finally:
    cursor.close()
```

- obtener_paciente_por_id: Obtiene los datos de un paciente por su ID.
  Referencia: ConsultasEncuesta.obtener_paciente_por_id.

```python
def obtener_paciente_por_id(self, id_paciente):
    """
    Obtiene los datos de un paciente por su ID
    Args:
        id_paciente: ID del paciente a buscar
    Returns:
        dict: Diccionario con los datos del paciente o None si no se encuentra
    """
    query = """
    SELECT
        idEncuesta,
        Sexo,
        edad,
        BebidasSemana,
        BebidasFinSemana,
        CervezasSemana,
        BebidasDestiladasSemana,
        VinosSemana,
        PerdidasControl,
        DiversionDependenciaAlcohol,
        ProblemasDigestivos,
        TensionAlta,
        DolorCabeza
    FROM encuesta
    WHERE idEncuesta = %s
    """
    try:
        with self.conexion.conexion.cursor(dictionary=True) as cursor:
            cursor.execute(query, (id_paciente,))
            return cursor.fetchone()
    except Exception as e:
        raise Exception(f"Error al obtener paciente: {str(e)}")
```

- actualizar_paciente: Actualiza los datos de un paciente existente.
  Referencia: ConsultasEncuesta.actualizar_paciente.

```python
def actualizar_paciente(self, datos):
    """Actualiza los datos de un paciente existente"""
    try:
        self.consultas.actualizar_paciente(datos)
        messagebox.showinfo("Éxito", "Paciente actualizado correctamente")
    except Exception as e:
        messagebox.showerror("Error", f"Error al actualizar paciente: {str(e)}")
```

- eliminar_paciente: Elimina un paciente de la base de datos. Referencia: ConsultasEncuesta.eliminar_paciente.

```python
def eliminar_paciente(self, id_paciente):
    """
    Elimina un paciente y actualiza el listado
    Args:
        id_paciente: ID del paciente a eliminar
    """
    try:
        self.consultas.eliminar_paciente(id_paciente)
        messagebox.showinfo("Éxito", "Paciente eliminado correctamente")
        # Actualizar el listado
        pacientes = self.consultas.obtener_listado_pacientes()
        self.interfaz.actualizar_listado_pacientes(pacientes)
    except Exception as e:
        messagebox.showerror("Error", f"No se pudo eliminar el paciente: {str(e)}")
```

- obtener_listado_pacientes: Obtiene el listado completo de pacientes de la base de datos.
  Referencia: ConsultasEncuesta.obtener_listado_pacientes.

```python
def obtener_listado_pacientes(self):
    """
    Obtiene el listado completo de pacientes de la base de datos
    :return: DataFrame con todos los pacientes
    """
    query = """
    SELECT
        idEncuesta,
        NOW() as Fecha,  # Usamos NOW() como fecha temporal
        Sexo,
        edad,
        BebidasSemana,
        BebidasFinSemana,
        CervezasSemana,
        BebidasDestiladasSemana,
        VinosSemana
    FROM encuesta
    ORDER BY idEncuesta DESC
    """
    try:
        return pd.read_sql(query, self.conexion.conexion)
    except Exception as e:
        print(f"Error al obtener listado de pacientes: {e}")
        raise Exception(f"Error al obtener listado de pacientes: {str(e)}")
```

- obtener_registros_recientes: Obtiene los registros más recientes de la base de datos.

Referencia: ConsultasEncuesta.obtener_registros_recientes.

```python
def obtener_registros_recientes(self, limite=10):
    """
    Obtiene los registros más recientes de la base de datos
    """
    query = """
    SELECT
        idEncuesta,
        Sexo,
        edad,
        BebidasSemana,
        BebidasFinSemana,
        BebidasDestiladasSemana,
        VinosSemana,
        CervezasSemana
    FROM encuesta
    ORDER BY idEncuesta DESC
    LIMIT %s
    """
    try:
        return pd.read_sql(query, self.conexion.conexion, params=(limite,))
    except Exception as e:
        print(f"Error al obtener registros recientes: {e}")
        return pd.DataFrame()
```

- obtener_estadisticas_consumo: Obtiene estadísticas detalladas de consumo.
  Referencia: ConsultasEncuesta.obtener_estadisticas_consumo

```python
def obtener_estadisticas_consumo(self):
    """Obtiene estadísticas detalladas de consumo"""
    query = """
    SELECT
        idEncuesta,
        edad,
        Sexo,
        BebidasSemana,
        CervezasSemana,
        BebidasFinSemana,
        BebidasDestiladasSemana,
        VinosSemana,
        AVG(BebidasSemana) as promedio_semanal,
        AVG(CervezasSemana) as promedio_cerveza,
        AVG(BebidasFinSemana) as promedio_finde,
        AVG(BebidasDestiladasSemana) as promedio_destiladas,
        AVG(VinosSemana) as promedio_vinos
    FROM encuesta
    GROUP BY idEncuesta, edad, Sexo, BebidasSemana, CervezasSemana,
             BebidasFinSemana, BebidasDestiladasSemana, VinosSemana
    """
    try:
        return pd.read_sql(query, self.conexion.conexion)
    except Exception as e:
        print(f"Error en obtener_estadisticas_consumo: {str(e)}")
        return pd.DataFrame()  # Retorna DataFrame vacío en caso de error
```

- filtrar_alto_consumo: Filtra registros con alto consumo de alcohol.
  Referencia: ConsultasEncuesta.filtrar_alto_consumo.

```python
def filtrar_alto_consumo(self, limite=20):
    """Filtra registros con alto consumo de alcohol"""
    query = """
    SELECT
        idEncuesta,
        edad,
        Sexo,
        BebidasSemana,
        CervezasSemana,
        BebidasFinSemana,
        BebidasDestiladasSemana,
        VinosSemana
    FROM encuesta
    WHERE BebidasSemana > %s
       OR BebidasFinSemana > %s
       OR CervezasSemana > %s
       OR BebidasDestiladasSemana > %s
       OR VinosSemana > %s
    ORDER BY BebidasSemana DESC
    LIMIT 100
    """
    params = (limite,) * 5
    return pd.read_sql(query, self.conexion.conexion, params=params)
```

- filtrar_perdidas_control: Filtra pacientes que han perdido el control más veces que el mínimo especificado. Referencia: ConsultasEncuesta.filtrar_perdidas_control.

```python
def filtrar_perdidas_control(self, min_perdidas=3):
    """
    Filtra pacientes que han perdido el control más veces que el mínimo especificado
    Args:
        min_perdidas: Número mínimo de pérdidas de control
    Returns:
        DataFrame con los pacientes filtrados
    """
    query = """
    SELECT
        idEncuesta,
        Sexo,
        edad,
        PerdidasControl,
        BebidasSemana,
        BebidasFinSemana,
        CervezasSemana,
        BebidasDestiladasSemana,
        VinosSemana
    FROM encuesta
    WHERE PerdidasControl >= %s
    ORDER BY PerdidasControl DESC
    """
    try:
        return pd.read_sql(query, self.conexion.conexion, params=(min_perdidas,))
    except Exception as e:
        print(f"Error al filtrar pérdidas de control: {e}")
        return pd.DataFrame()
```

- filtrar_problemas_salud: Analiza problemas de salud relacionados con el consumo.
  Referencia: ConsultasEncuesta.filtrar_problemas_salud.

```python
def filtrar_problemas_salud(self):
    """Analiza problemas de salud relacionados con el consumo"""
    query = """
    SELECT
        Sexo,
        SUM(CASE WHEN ProblemasDigestivos = 'Sí' THEN 1 ELSE 0 END) as problemas_digestivos,
        SUM(CASE WHEN TensionAlta = 'Sí' THEN 1 ELSE 0 END) as tension_alta,
        SUM(CASE WHEN DolorCabeza IN ('A menudo', 'Muy a menudo') THEN 1 ELSE 0 END) as dolor_cabeza_frecuente,
        AVG(BebidasSemana) as promedio_consumo_semanal,
        COUNT(*) as total_casos
    FROM encuesta
    GROUP BY Sexo
    """
    return pd.read_sql(query, self.conexion.conexion)
```

- obtener_correlacion_salud_consumo: Analiza la correlación entre consumo y problemas de salud.
  Referencia: ConsultasEncuesta.obtener_correlacion_salud_consumo.

```python
def obtener_correlacion_salud_consumo(self):
    """Analiza la correlación entre consumo y problemas de salud"""
    query = """
    SELECT
        CASE
            WHEN BebidasSemana = 0 THEN 'No consume'
            WHEN BebidasSemana <= 5 THEN 'Consumo bajo'
            WHEN BebidasSemana <= 15 THEN 'Consumo moderado'
            ELSE 'Consumo alto'
        END as nivel_consumo,
        COUNT(*) as total_personas,
        SUM(CASE WHEN ProblemasDigestivos = 'Sí' THEN 1 ELSE 0 END) as casos_digestivos,
        SUM(CASE WHEN TensionAlta = 'Sí' THEN 1 ELSE 0 END) as casos_tension,
        SUM(CASE WHEN DolorCabeza IN ('A menudo', 'Muy a menudo') THEN 1 ELSE 0 END) as casos_dolor_cabeza
    FROM encuesta
    GROUP BY
        CASE
            WHEN BebidasSemana = 0 THEN 'No consume'
            WHEN BebidasSemana <= 5 THEN 'Consumo bajo'
            WHEN BebidasSemana <= 15 THEN 'Consumo moderado'
            ELSE 'Consumo alto'
        END
    """
    return pd.read_sql(query, self.conexion.conexion)
```

- exportar_a_excel: Exporta un DataFrame a Excel. Referencia: ConsultasEncuesta.exportar_a_excel.

```python
def exportar_a_excel(self, tipo_exportacion):
    """Exporta los datos a un archivo Excel"""
    try:
        # Obtener datos actuales
        datos = self.consultas.obtener_estadisticas_consumo()
        if datos is None or datos.empty:
            messagebox.showwarning("Advertencia", "No hay datos para exportar")
            return

        # Pedir al usuario la ubicación para guardar el archivo
        from tkinter import filedialog
        filename = filedialog.asksaveasfilename(
            defaultextension=".xlsx",
            filetypes=[("Excel files", "*.xlsx"), ("All files", "*.*")],
            title="Guardar como Excel"
        )

        if filename:
            # Exportar utilizando el método de consultas
            if self.consultas.exportar_a_excel(datos, filename):
                messagebox.showinfo("Éxito", "Datos exportados correctamente")
            else:
                messagebox.showerror("Error", "No se pudo exportar el archivo")

    except Exception as e:
        messagebox.showerror("Error", f"Error al exportar: {str(e)}")
        print(f"Error detallado en exportación: {str(e)}")
```

graficas/visualizargraficas.py

- **Objetivo**: Genera y muestra gráficos basados en los datos de las consultas.
- **Clases y Métodos**:
  - VisualizadorGraficas: Clase que contiene métodos para crear diferentes tipos de gráficos (barras, pastel, líneas, etc.). Referencia: VisualizadorGraficas.
    - crear_grafica_consumo_edad: Crea un gráfico de barras para visualizar el consumo por edad. Referencia: VisualizadorGraficas.crear_grafica_consumo_edad

```
def crear_grafica_consumo_edad(self, datos):
    if datos is None or datos.empty:
        return None

    fig, ax = plt.subplots()
    sns.barplot(
        data=datos,
        x='edad',
        y='promedio_semanal',
        hue='Sexo',
        ax=ax
    )
    ax.set_title('Consumo por Edad')
    ax.set_xlabel('Edad')
    ax.set_ylabel('Bebidas por Semana')
    plt.tight_layout()
    return fig
```

    - crear_grafica_tendencia: Crea un gráfico de líneas para visualizar la tendencia de consumo de alcohol. Referencia: VisualizadorGraficas.crear_grafica_tendencia .

```python
def crear_grafica_tendencia(self, datos):
    if datos is None or datos.empty:
        return None

    fig, ax = plt.subplots()
    sns.lineplot(
        data=datos,
        x='idEncuesta',
        y='BebidasSemana',
        ax=ax
    )
    ax.set_title('Tendencia de Consumo')
    ax.set_xlabel('ID Encuesta')
    ax.set_ylabel('Bebidas por Semana')
    plt.tight_layout()
    return fig
```

- crear_grafica_problemas_salud: Crea un gráfico de barras para visualizar problemas de salud por género.
  Referencia: VisualizadorGraficas.crear_grafica_problemas_salud.

```python
def crear_grafica_problemas_salud(self, datos):
    if datos is None or datos.empty:
        return None

    # Transformar los datos para la visualización
    datos_melt = datos.melt(
        id_vars=['Sexo'],
        value_vars=[
            'problemas_digestivos',
            'tension_alta',
            'dolor_cabeza_frecuente'
        ],
        var_name='Problema',
        value_name='Cantidad'
    )

    fig, ax = plt.subplots()
    sns.barplot(
        data=datos_melt,
        x='Problema',
        y='Cantidad',
        hue='Sexo',
        ax=ax
    )

    # Personalizar la gráfica
    ax.set_title('Problemas de Salud por Género')
    ax.set_xlabel('Tipo de Problema')
    ax.set_ylabel('Cantidad de Casos')

    # Mejorar las etiquetas
    etiquetas = {
        'problemas_digestivos': 'Digestivos',
        'tension_alta': 'Tensión Alta',
        'dolor_cabeza_frecuente': 'Dolor Cabeza'
    }
    ax.set_xticklabels([etiquetas[x.get_text()] for x in ax.get_xticklabels()])

    plt.xticks(rotation=45)
    plt.tight_layout()
    return fig
```

interfaz/interfazusuario.py

- **Objetivo**: Define la interfaz gráfica del usuario.
- **Clases y Métodos**:
  - InterfazSaludAlcohol: Clase que crea y gestiona los diferentes paneles y componentes de la interfaz gráfica.
    Referencia: InterfazSaludAlcohol.

- __init__: Inicializa la interfaz y los campos de entrada. Referencia: InterfazSaludAlcohol.__init__.

```python
def __init__(self, root):
    self.root = root
    self.root.title("Sistema de Monitoreo de Salud")

    # Inicializar campos de entrada
    self.entrada_fecha = None
    self.entrada_sexo = None
    self.entrada_edad = None
    self.entrada_alcohol = None
    self.entrada_presion = None
    self.entrada_problemas = None
    self.entrada_bebidas_semana = None
    self.entrada_cervezas = None
    self.entrada_finde = None
    self.entrada_destiladas = None
    self.entrada_vinos = None
    self.entrada_perdidas_control = None
    self.entrada_diversion_alcohol = None
    self.entrada_problemas_digestivos = None
    self.entrada_tension_alta = None
    self.entrada_dolor_cabeza = None

    # Campos para actualización
    self.entrada_id_busqueda = None
    self.entrada_fecha_act = None
    self.entrada_edad_act = None
    self.entrada_sexo_act = None
    self.entrada_bebidas_semana_act = None
    self.entrada_cervezas_act = None
    self.entrada_finde_act = None
    self.entrada_destiladas_act = None
    self.entrada_vinos_act = None
    self.entrada_perdidas_control_act = None
    self.entrada_diversion_alcohol_act = None
    self.entrada_problemas_digestivos_act = None
    self.entrada_tension_alta_act = None
    self.entrada_dolor_cabeza_act = None
```

```
    # Inicializar callbacks
    self._registrar_callback = None
    self._visualizar_callback = None
    self._estadisticas_callback = None
    self._exportar_callback = None
    self._buscar_callback = None
    self._actualizar_callback = None
    self._eliminar_callback = None

    # Crear componentes principales
    self.crear_notebook()
    self.crear_menu()
    self.crear_panel_registro()
    self.crear_panel_visualizacion()
    self.crear_panel_estadisticas()
    self.crear_panel_actualizar()
    self.crear_panel_listado()

    # Inicializar canvas
    self.canvas = None
```

- crear_notebook: Crea las pestañas de la interfaz.
  Referencia: InterfazSaludAlcohol.crear_notebook.

```
def crear_notebook(self):
    self.notebook = ttk.Notebook(self.root)
    self.notebook.pack(fill='both', expand=True)

    self.tab_registro = ttk.Frame(self.notebook)
    self.tab_visualizacion = ttk.Frame(self.notebook)
    self.tab_estadisticas = ttk.Frame(self.notebook)
    self.tab_listado = ttk.Frame(self.notebook)  # Nueva pestaña

    self.notebook.add(self.tab_registro, text='Registro')
    self.notebook.add(self.tab_listado, text='Listado Pacientes')
    self.tab_actualizar = ttk.Frame(self.notebook)
    self.notebook.add(self.tab_actualizar, text='Actualizar Paciente')
    self.notebook.add(self.tab_visualizacion, text='Visualización')
    self.notebook.add(self.tab_estadisticas, text='Estadísticas')
```

- crear_menu: Crea el menú de la aplicación.
  Referencia: InterfazSaludAlcohol.crear_menu.

```python
def crear_menu(self):
    menu_bar = tk.Menu(self.root)
    self.root.config(menu=menu_bar)

    # Menú Archivo
    file_menu = tk.Menu(menu_bar, tearoff=0)
    menu_bar.add_cascade(label="Archivo", menu=file_menu)
    file_menu.add_command(label="Exportar a Excel",
                          command=lambda: self._exportar_callback("excel") if self._exportar_callback else None)
    file_menu.add_separator()
    file_menu.add_command(label="Salir", command=self.root.quit)

    # Menú Consultas
    query_menu = tk.Menu(menu_bar, tearoff=0)
    menu_bar.add_cascade(label="Consultas", menu=query_menu)
    query_menu.add_command(label="Ver Estadísticas",
                          command=lambda: self.notebook.select(self.tab_estadisticas))
    query_menu.add_command(label="Actualizar Paciente",
                          command=lambda: self.notebook.select(self.tab_actualizar))
```

- crear_panel_visualizacion: Crea el panel para visualizar gráficos.
  Referencia: InterfazSaludAlcohol.crear_panel_visualizacion.

```python
def crear_panel_visualizacion(self):
    """Crea el panel de visualización de gráficas"""
    self.frame_grafico = ttk.LabelFrame(self.tab_visualizacion, text="Gráficos")
    self.frame_grafico.pack(padx=10, pady=10, fill='both', expand=True)

    # Panel de controles
    frame_controles = ttk.Frame(self.frame_grafico)
    frame_controles.pack(fill='x', padx=5, pady=5)

    # Selector de tipo de gráfica
    ttk.Label(frame_controles, text="Tipo de Gráfica:").pack(side='left', padx=5)
    self.tipo_grafica = ttk.Combobox(
        frame_controles,
        values=[
            'Consumo por Edad',
            'Problemas de Salud',
            'Tendencia Temporal'
        ],
        state='readonly',
        width=30
    )
    self.tipo_grafica.pack(side='left', padx=5)
    self.tipo_grafica.set('Consumo por Edad')

    # Frame para contener el gráfico
    self.frame_figura = ttk.Frame(self.frame_grafico)
    self.frame_figura.pack(fill='both', expand=True, padx=5, pady=5)

    # Botón actualizar
    ttk.Button(
        frame_controles,
        text="Mostrar Gráfica",
        command=self._on_visualizar
    ).pack(side='right', padx=5)
```

- crear_panel_registro: Crea el panel para registrar nuevos pacientes.
  Referencia: InterfazSaludAlcohol.crear_panel_registro.

```python
def crear_panel_registro(self):
    """Crea un panel de registro mejorado y más intuitivo"""
    frame_principal = ttk.Frame(self.tab_registro, style="Custom.TLabelframe")
    frame_principal.pack(padx=30, pady=20, fill='both', expand=True)

    # Título principal
    titulo = ttk.Label(frame_principal,
                       text="REGISTRO DE PACIENTE",
                       style="Title.TLabel")
    titulo.pack(pady=(0, 20))

    # Sección de datos personales
    frame_datos = ttk.LabelFrame(frame_principal, text="Datos Personales")
    frame_datos.pack(fill='x', padx=20, pady=10)

    frame_grid = ttk.Frame(frame_datos)
    frame_grid.pack(padx=15, pady=15)

    # Datos Personales
    ttk.Label(frame_grid, text="Fecha:").grid(row=0, column=0, padx=10, pady=8, sticky='e')
    self.entrada_fecha = ttk.Entry(frame_grid, width=15)
    self.entrada_fecha.grid(row=0, column=1, padx=10, pady=8, sticky='w')
    ttk.Label(frame_grid, text="(YYYY-MM-DD)", foreground="gray").grid(row=0, column=2, pady=8, sticky='w')

    ttk.Label(frame_grid, text="Edad:").grid(row=1, column=0, padx=10, pady=8, sticky='e')
    self.entrada_edad = ttk.Entry(frame_grid, width=5)
    self.entrada_edad.grid(row=1, column=1, padx=10, pady=8, sticky='w')
    ttk.Label(frame_grid, text="años", foreground="gray").grid(row=1, column=2, pady=8, sticky='w')

    ttk.Label(frame_grid, text="Sexo:").grid(row=2, column=0, padx=10, pady=8, sticky='e')
    self.entrada_sexo = ttk.Combobox(frame_grid, values=["Hombre", "Mujer"], state='readonly')
    self.entrada_sexo.grid(row=2, column=1, padx=10, pady=8, sticky='w')
    self.entrada_sexo.set("Hombre")

    # Sección de Consumo de Alcohol
    frame_consumo = ttk.LabelFrame(frame_principal, text="Consumo de Alcohol")
    frame_consumo.pack(fill='x', padx=20, pady=10)
    frame_grid_consumo = ttk.Frame(frame_consumo)
    frame_grid_consumo.pack(padx=15, pady=15)
```

```python
# Campos de Consumo
campos_consumo = [
    ("Bebidas por Semana:", "bebidas_semana", 5),
    ("Cervezas por Semana:", "cervezas", 5),
    ("Bebidas Fin de Semana:", "finde", 5),
    ("Bebidas Destiladas:", "destiladas", 5),
    ("Vinos por Semana:", "vinos", 5)
]

for i, (label, nombre, width) in enumerate(campos_consumo):
    ttk.Label(frame_grid_consumo, text=label).grid(row=i, column=0, padx=10, pady=8, sticky='e')
    setattr(self, f'entrada_{nombre}', ttk.Entry(frame_grid_consumo, width=width))
    getattr(self, f'entrada_{nombre}').grid(row=i, column=1, padx=10, pady=8, sticky='w')
    ttk.Label(frame_grid_consumo, text="unidades", foreground="gray").grid(row=i, column=2, pady=8, sticky='w')

# Sección de Salud
frame_salud = ttk.LabelFrame(frame_principal, text="Datos de Salud")
frame_salud.pack(fill='x', padx=20, pady=10)
frame_grid_salud = ttk.Frame(frame_salud)
frame_grid_salud.pack(padx=15, pady=15)

# Campos de Salud
campos_salud = [
    ("Pérdidas de Control:", "perdidas_control", ["Sí", "No"]),
    ("Diversión Depende del Alcohol:", "diversion_alcohol", ["Sí", "No"]),
    ("Problemas Digestivos:", "problemas_digestivos", ["Sí", "No"]),
    ("Tensión Alta:", "tension_alta", ["Sí", "No"]),
    ("Frecuencia Dolor de Cabeza:", "dolor_cabeza",
     ["Nunca", "Raramente", "A menudo", "Muy a menudo"])
]

for i, (label, nombre, opciones) in enumerate(campos_salud):
    ttk.Label(frame_grid_salud, text=label).grid(row=i, column=0, padx=10, pady=8, sticky='e')
    setattr(self, f'entrada_{nombre}', ttk.Combobox(frame_grid_salud, values=opciones, state='readonly'))
    getattr(self, f'entrada_{nombre}').grid(row=i, column=1, padx=10, pady=8, sticky='w')
    getattr(self, f'entrada_{nombre}').set(opciones[0])

# Frame para botones
frame_botones = ttk.Frame(frame_principal)
frame_botones.pack(pady=25)
```

```python
# Frame para botones
frame_botones = ttk.Frame(frame_principal)
frame_botones.pack(pady=25)

# Botones mejorados
ttk.Button(frame_botones,
           text="Limpiar Formulario",
           style="Secondary.TButton",
           command=self.limpiar_formulario).pack(side='left', padx=15)
ttk.Button(frame_botones,
           text="Registrar Paciente",
           style="Primary.TButton",
           command=self._on_registrar).pack(side='left', padx=15)

# Configurar validación después de crear todos los widgets
self._configurar_validacion()
```

- crear_panel_listado: Crea el panel para listar pacientes. Referencia: InterfazSaludAlcohol.crear_panel_listado.

```python
def crear_panel_listado(self):
    frame = ttk.LabelFrame(self.tab_listado, text="Listado de Pacientes")
    frame.pack(padx=10, pady=10, fill='both', expand=True)

    # Crear Treeview para el listado
    columns = ('ID', 'Fecha', 'Sexo', 'Edad', 'Bebidas/Sem', 'Fin Sem', 'Cervezas', 'Destiladas', 'Vinos')
    self.tree_pacientes = ttk.Treeview(frame, columns=columns, show='headings', height=15)

    # Configurar columnas
    for col in columns:
        self.tree_pacientes.heading(col, text=col)
        self.tree_pacientes.column(col, width=80)

    # Scrollbars
    scrolly = ttk.Scrollbar(frame, orient='vertical', command=self.tree_pacientes.yview)
    scrollx = ttk.Scrollbar(frame, orient='horizontal', command=self.tree_pacientes.xview)
    self.tree_pacientes.configure(yscrollcommand=scrolly.set, xscrollcommand=scrollx.set)

    # Evento de doble clic para eliminar
    self.tree_pacientes.bind('<Double-1>', self._on_doble_click_paciente)

    # Empaquetar componentes
    self.tree_pacientes.pack(side='left', fill='both', expand=True)
    scrolly.pack(side='right', fill='y')
    scrollx.pack(side='bottom', fill='x')
```

- crear_panel_actualizar: Crea el panel para actualizar datos de pacientes.
  Referencia: InterfazSaludAlcohol.crear_panel_actualizar.

```python
def crear_panel_actualizar(self):
    frame = ttk.LabelFrame(self.tab_actualizar, text="Actualizar Paciente")
    frame.pack(padx=10, pady=10, fill='both', expand=True)

    # Frame búsqueda
    frame_busqueda = ttk.Frame(frame)
    frame_busqueda.pack(fill='x', padx=5, pady=5)

    ttk.Label(frame_busqueda, text="ID Paciente:").pack(side='left', padx=5)
    self.entrada_id_busqueda = ttk.Entry(frame_busqueda, width=10)
    self.entrada_id_busqueda.pack(side='left', padx=5)

    ttk.Button(frame_busqueda,
               text="Buscar",
               command=self._buscar_paciente).pack(side='left', padx=5)

    # Frame principal para los datos
    self.frame_actualizar = ttk.Frame(frame)
    self.frame_actualizar.pack(fill='both', expand=True, padx=5, pady=5)

    # Desactivar frame de actualización hasta que se busque un paciente
    for child in self.frame_actualizar.winfo_children():
        child.configure(state='disabled')

    # Frame principal para los datos
    frame_principal = ttk.Frame(frame)
    frame_principal.pack(fill='both', expand=True, padx=5, pady=5)

    # Sección de datos personales
    frame_datos = ttk.LabelFrame(frame_principal, text="Datos Personales")
    frame_datos.pack(fill='x', padx=20, pady=10)

    frame_grid = ttk.Frame(frame_datos)
    frame_grid.pack(padx=15, pady=15)

    # Datos Personales
    ttk.Label(frame_grid, text="Fecha:").grid(row=0, column=0, padx=10, pady=8, sticky='e')
    self.entrada_fecha_act = ttk.Entry(frame_grid, width=15)
    self.entrada_fecha_act.grid(row=0, column=1, padx=10, pady=8, sticky='w')
    ttk.Label(frame_grid, text="(YYYY-MM-DD)", foreground="gray").grid(row=0, column=2, pady=8, sticky='w')
```

```
    def crear_panel_actualizar(self):
        ttk.Label(frame_grid, text="Edad:").grid(row=1, column=0, padx=10, pady=8, sticky='e')
        self.entrada_edad_act = ttk.Entry(frame_grid, width=5)
        self.entrada_edad_act.grid(row=1, column=1, padx=10, pady=8, sticky='w')
        ttk.Label(frame_grid, text="años", foreground="gray").grid(row=1, column=2, pady=8, sticky='w')

        ttk.Label(frame_grid, text="Sexo:").grid(row=2, column=0, padx=10, pady=8, sticky='e')
        self.entrada_sexo_act = ttk.Combobox(frame_grid, values=["Hombre", "Mujer"], state='readonly')
        self.entrada_sexo_act.grid(row=2, column=1, padx=10, pady=8, sticky='w')

        # Sección de Consumo de Alcohol
        frame_consumo = ttk.LabelFrame(frame_principal, text="Consumo de Alcohol")
        frame_consumo.pack(fill='x', padx=20, pady=10)
        frame_grid_consumo = ttk.Frame(frame_consumo)
        frame_grid_consumo.pack(padx=15, pady=15)

        # Campos de Consumo
        campos_consumo = [
            ("Bebidas por Semana:", "bebidas_semana_act", 5),
            ("Cervezas por Semana:", "cervezas_act", 5),
            ("Bebidas Fin de Semana:", "finde_act", 5),
            ("Bebidas Destiladas:", "destiladas_act", 5),
            ("Vinos por Semana:", "vinos_act", 5)
        ]

        for i, (label, nombre, width) in enumerate(campos_consumo):
            ttk.Label(frame_grid_consumo, text=label).grid(row=i, column=0, padx=10, pady=8, sticky='e')
            setattr(self, f'entrada_{nombre}', ttk.Entry(frame_grid_consumo, width=width))
            getattr(self, f'entrada_{nombre}').grid(row=i, column=1, padx=10, pady=8, sticky='w')
            ttk.Label(frame_grid_consumo, text="unidades", foreground="gray").grid(row=i, column=2, pady=8, sticky='w')

        # Sección de Salud
        frame_salud = ttk.LabelFrame(frame_principal, text="Datos de Salud")
        frame_salud.pack(fill='x', padx=20, pady=10)
        frame_grid_salud = ttk.Frame(frame_salud)
        frame_grid_salud.pack(padx=15, pady=15)

        # Campos de Salud
        campos_salud = [
            ("Pérdidas de Control:", "perdidas_control_act", ["Sí", "No"]),
            ("Diversión Depende del Alcohol:", "diversion_alcohol_act", ["Sí", "No"]),
            ("Problemas Digestivos:", "problemas_digestivos_act", ["Sí", "No"]),
            ("Tensión Alta:", "tension_alta_act", ["Sí", "No"]),
            ("Frecuencia Dolor de Cabeza:", "dolor_cabeza_act",
             ["Nunca", "Raramente", "A menudo", "Muy a menudo"])
        ]
```

```
    for i, (label, nombre, opciones) in enumerate(campos_salud):
        ttk.Label(frame_grid_salud, text=label).grid(row=i, column=0, padx=10, pady=8, sticky='e')
        setattr(self, f'entrada_{nombre}', ttk.Combobox(frame_grid_salud, values=opciones, state='readonly'))
        getattr(self, f'entrada_{nombre}').grid(row=i, column=1, padx=10, pady=8, sticky='w')
        getattr(self, f'entrada_{nombre}').set(opciones[0])

    # Frame para botones
    frame_botones = ttk.Frame(self.frame_actualizar)
    frame_botones.pack(pady=25)

    # Botón para guardar cambios
    self.boton_guardar = ttk.Button(frame_botones,
                                    text="Guardar Cambios",
                                    style="Primary.TButton",
                                    command=self._guardar_actualizacion,
                                    state='disabled')
    self.boton_guardar.pack(side='left', padx=15)
```

- registrar_buscar_callback: Registra el callback para la búsqueda de pacientes.
  Referencia: InterfazSaludAlcohol.registrar_buscar_callback.

```
def registrar_buscar_callback(self, callback):
    """Registra el callback para búsqueda de pacientes"""
    if callable(callback):
        self._buscar_callback = callback
```

- registrar_actualizar_callback: Registra el callback para la actualización de pacientes.
  Referencia: InterfazSaludAlcohol.registrar_actualizar_callback.

```
def registrar_actualizar_callback(self, callback):
    """Registra el callback para actualización de pacientes"""
    if callable(callback):
        self._actualizar_callback = callback
```

- mostrar_datos_paciente: Muestra los datos de un paciente en la interfaz.
  Referencia: InterfazSaludAlcohol.mostrar_datos_paciente.

```python
def mostrar_datos_paciente(self, datos):
    if datos is not None:
        try:
            # Habilitar campos para edición
            for widget in self.frame_actualizar.winfo_children():
                try:
                    widget.configure(state='normal')
                except:
                    pass

            # Rellenar campos con los datos existentes
            self.entrada_edad_act.delete(0, tk.END)
            self.entrada_edad_act.insert(0, str(datos['edad']))

            self.entrada_sexo_act.set(datos['Sexo'])

            self.entrada_bebidas_semana_act.delete(0, tk.END)
            self.entrada_bebidas_semana_act.insert(0, str(datos['BebidasSemana']))

            self.entrada_cervezas_act.delete(0, tk.END)
            self.entrada_cervezas_act.insert(0, str(datos['CervezasSemana']))

            self.entrada_finde_act.delete(0, tk.END)
            self.entrada_finde_act.insert(0, str(datos['BebidasFinSemana']))

            self.entrada_destiladas_act.delete(0, tk.END)
            self.entrada_destiladas_act.insert(0, str(datos['BebidasDestiladasSemana']))

            self.entrada_vinos_act.delete(0, tk.END)
            self.entrada_vinos_act.insert(0, str(datos['VinosSemana']))

            # Habilitar botón de guardar
            self.boton_guardar.configure(state='normal')

        except Exception as e:
            messagebox.showerror("Error", f"Error al mostrar datos del paciente: {str(e)}")
```

- mostrar_estadisticas: Muestra las estadísticas de consumo de alcohol y problemas de salud.
  Referencia: InterfazSaludAlcohol.mostrar_estadisticas.

```python
def mostrar_estadisticas(self):
    """Muestra las estadísticas de consumo y registros recientes"""
    try:
        # Obtener estadísticas básicas
        stats = self.consultas.obtener_estadisticas_consumo()
        if stats is None or stats.empty:
            raise ValueError("No hay datos disponibles")

        # Obtener registros recientes
        registros_recientes = self.consultas.obtener_registros_recientes()

        # Obtener datos de alto consumo
        alto_consumo = self.consultas.filtrar_alto_consumo()

        # Actualizar la interfaz con todos los datos
        self.interfaz.actualizar_estadisticas(stats, alto_consumo)
        if not registros_recientes.empty:
            self.interfaz.actualizar_registros_recientes(registros_recientes)

    except Exception as e:
        messagebox.showerror("Error", f"Error al mostrar estadísticas: {str(e)}")
```

- exportar_a_excel: Exporta los datos a un archivo Excel.
  Referencia: InterfazSaludAlcohol.exportar_a_excel.

```python
def exportar_a_excel(self, tipo_exportacion):
    """Exporta los datos a un archivo Excel"""
    try:
        # Obtener datos actuales
        datos = self.consultas.obtener_estadisticas_consumo()
        if datos is None or datos.empty:
            messagebox.showwarning("Advertencia", "No hay datos para exportar")
            return

        # Pedir al usuario la ubicación para guardar el archivo
        from tkinter import filedialog
        filename = filedialog.asksaveasfilename(
            defaultextension=".xlsx",
            filetypes=[("Excel files", "*.xlsx"), ("All files", "*.*")],
            title="Guardar como Excel"
        )

        if filename:
            # Exportar utilizando el método de consultas
            if self.consultas.exportar_a_excel(datos, filename):
                messagebox.showinfo("Éxito", "Datos exportados correctamente")
            else:
                messagebox.showerror("Error", "No se pudo exportar el archivo")

    except Exception as e:
        messagebox.showerror("Error", f"Error al exportar: {str(e)}")
        print(f"Error detallado en exportación: {str(e)}")
```

- eliminar_paciente: Elimina un paciente utilizando el callback registrado.
  Referencia: InterfazSaludAlcohol.eliminar_paciente.

```python
def eliminar_paciente(self, id_paciente):
    """
    Elimina un paciente y actualiza el listado
    Args:
        id_paciente: ID del paciente a eliminar
    """
    try:
        self.consultas.eliminar_paciente(id_paciente)
        messagebox.showinfo("Éxito", "Paciente eliminado correctamente")
        # Actualizar el listado
        pacientes = self.consultas.obtener_listado_pacientes()
        self.interfaz.actualizar_listado_pacientes(pacientes)
    except Exception as e:
        messagebox.showerror("Error", f"No se pudo eliminar el paciente: {str(e)}")
```

- actualizar_registros_recientes: Actualiza los registros recientes en el Treeview.
  Referencia: InterfazSaludAlcohol.actualizar_registros_recientes.

```python
def actualizar_registros_recientes(self, registros):
    """
    Actualiza los registros recientes en el Treeview
    :param registros: DataFrame con los registros recientes
    """
    try:
        # Limpiar registros existentes
        for item in self.tree_registros_recientes.get_children():
            self.tree_registros_recientes.delete(item)

        # Insertar nuevos registros
        for _, registro in registros.iterrows():
            valores = (
                registro['idEncuesta'],
                registro['Sexo'],
                registro['edad'],
                f"{registro['BebidasSemana']:.1f}",
                f"{registro['BebidasFinSemana']:.1f}",
                f"{registro['BebidasDestiladasSemana']:.1f}",
                f"{registro['VinosSemana']:.1f}",
                f"{registro['CervezasSemana']:.1f}"
            )
            self.tree_registros_recientes.insert('', 'end', values=valores)
    except Exception as e:
        print(f"Error al actualizar registros recientes: {e}")
```

- actualizar_grafico: Actualiza el gráfico en la interfaz.
  Referencia: InterfazSaludAlcohol.actualizar_grafico.

```python
def actualizar_grafico(self, figura):
    """Actualiza el gráfico en la interfaz"""
    try:
        # Limpiar el frame de la figura
        for widget in self.frame_figura.winfo_children():
            widget.destroy()

        if figura is None:
            messagebox.showinfo("Info", "No hay datos para mostrar")
            return

        # Crear canvas para la figura
        canvas = FigureCanvasTkAgg(figura, self.frame_figura)
        canvas.draw()
        widget = canvas.get_tk_widget()
        widget.pack(fill='both', expand=True)

    except Exception as e:
        print(f"Error al actualizar gráfico: {str(e)}")
        messagebox.showerror("Error", f"Error al actualizar gráfico: {str(e)}")
```

- actualizar_listado_pacientes: Actualiza el listado de pacientes en el Treeview.
  Referencia: InterfazSaludAlcohol.actualizar_listado_pacientes.

```python
def actualizar_listado_pacientes(self, pacientes):
    try:
        # Limpiar listado existente
        for item in self.tree_pacientes.get_children():
            self.tree_pacientes.delete(item)

        # Insertar nuevos datos
        for _, paciente in pacientes.iterrows():
            self.tree_pacientes.insert('', 'end', values=(
                paciente['idEncuesta'],
                paciente['Fecha'].strftime('%Y-%m-%d') if 'Fecha' in paciente else '',
                paciente['Sexo'],
                paciente['edad'],
                f"{paciente['BebidasSemana']:.1f}",
                f"{paciente['BebidasFinSemana']:.1f}",
                f"{paciente['CervezasSemana']:.1f}",
                f"{paciente['BebidasDestiladasSemana']:.1f}",
                f"{paciente['VinosSemana']:.1f}"
            ))
    except Exception as e:
        messagebox.showerror("Error", f"Error al actualizar listado: {str(e)}")
```

Funcionalidades Clave

IMPORTANTE!!!!

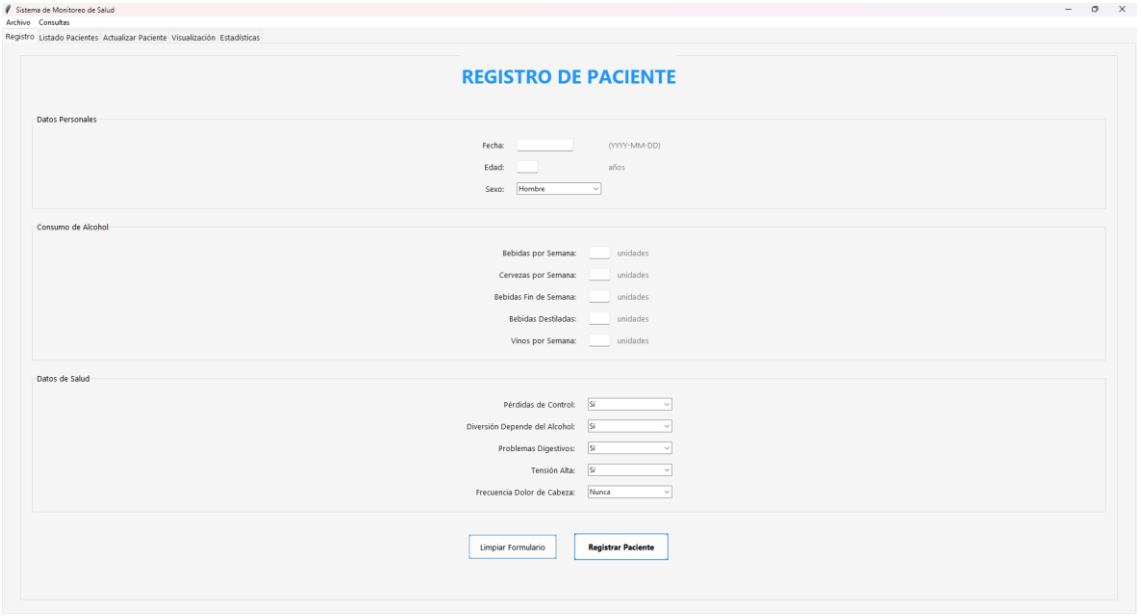
Hay que ejecutar el archivo requirements.txt para instalar todas las dependencias. El código para hacerlo es:

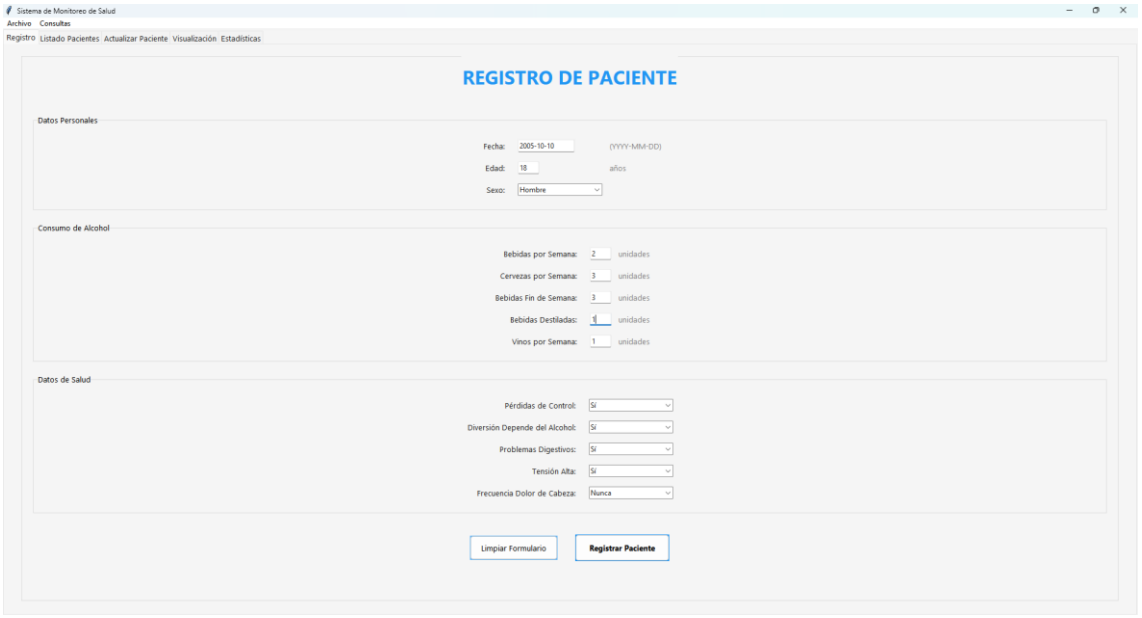
pip install -r requirements.txt

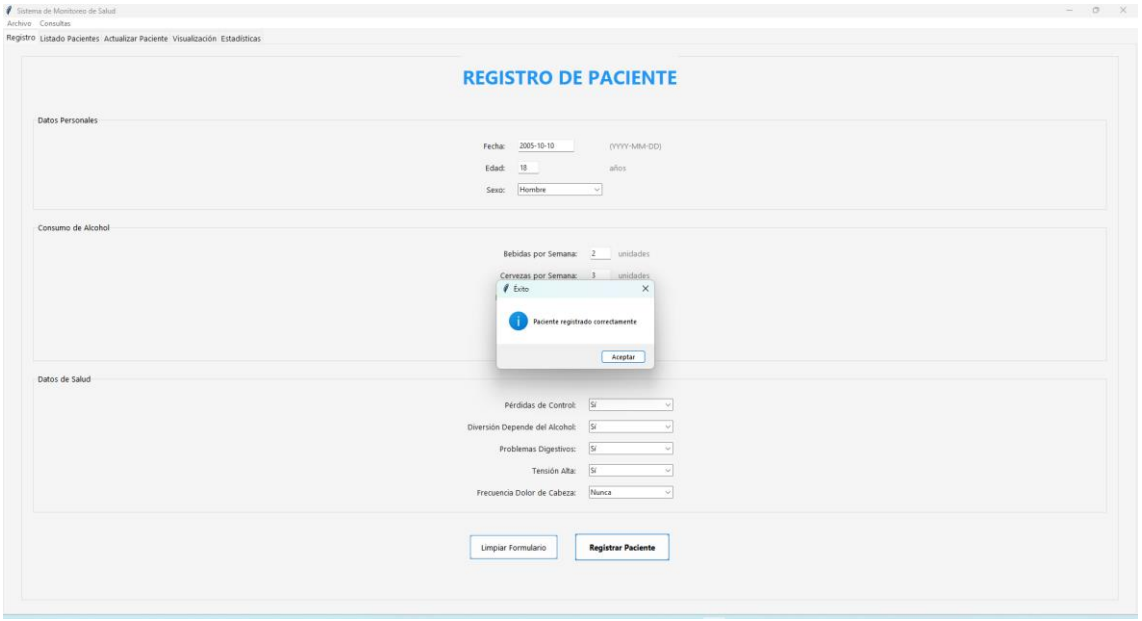
Registro de Pacientes

- Permite registrar nuevos pacientes con datos como fecha, sexo, edad, consumo de alcohol, problemas de salud, etc.

  Resultado:




Sistema de Monitoreo de Salud
Archivo Consultas
Registro Listado Pacientes Actualizar Paciente Visualización Estadísticas
REGISTRO DE PACIENTE
Datos Personales
Fecha: 2005-10-10 (YYYY-MM-DD)
Edad: 18 años
Sexo: Hombre
Consumo de Alcohol
Bebidas por Semana: 2 unidades
Cervezas por Semana: 3 unidades
Bebidas Fin de Semana: 3 unidades
Bebidas Destiladas: 1 unidades
Vinos por Semana: 1 unidades
Datos de Salud
Pérdidas de Control: Sí
Diversión Depende del Alcohol: Sí
Problemas Digestivos: Sí
Tensión Alta: Sí
Frecuencia Dolor de Cabeza: Nunca
Limpiar Formulario
Registrar Paciente


Sistema de Monitoreo de Salud
Archivo Consultas
Registro Listado Pacientes Actualizar Paciente Visualización Estadísticas
REGISTRO DE PACIENTE
Datos Personales
Fecha: 2005-10-10 (YYYY-MM-DD)
Edad: 18 años
Sexo: Hombre
Consumo de Alcohol
Bebidas por Semana: 2 unidades
Cervezas por Semana: 3 unidades
Éxito
Paciente registrado correctamente
Aceptar
Datos de Salud
Pérdidas de Control: Sí
Diversión Depende del Alcohol: Sí
Problemas Digestivos: Sí
Tensión Alta: Sí
Frecuencia Dolor de Cabeza: Nunca
Limpiar Formulario
Registrar Paciente

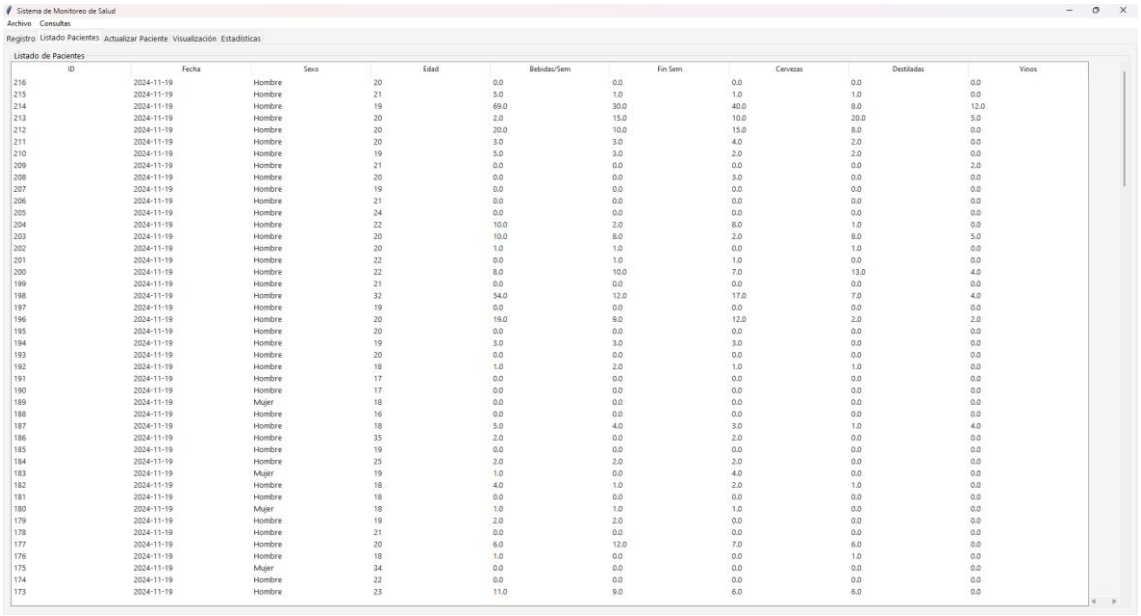
Registros Recientes

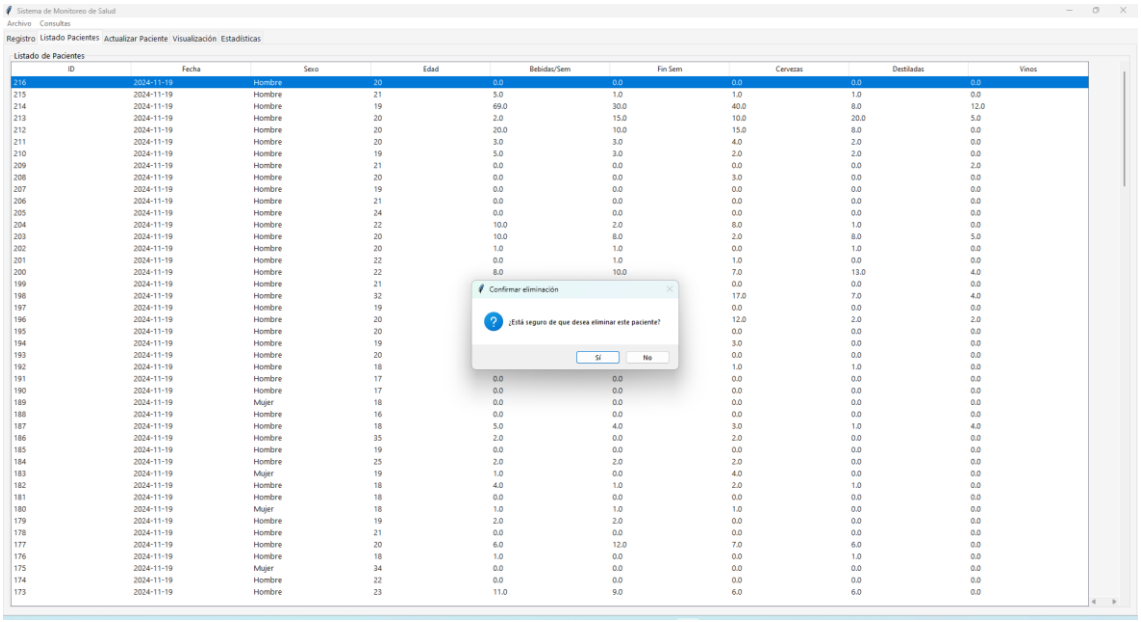
ID	Sexo	Edad	Bebidas	Fin Sem	Dest	Vinos	Cerv
217	Hombre	18	2.0	3.0	1.0	1.0	3.0
216	Hombre	21	3.0	2.0	1.0	3.0	2.0
215	Hombre	21	5.0	1.0	1.0	0.0	1.0
214	Hombre	19	69.0	30.0	8.0	12.0	40.0
213	Hombre	20	2.0	15.0	20.0	5.0	10.0
212	Hombre	20	20.0	10.0	8.0	0.0	15.0
211	Hombre	20	3.0	3.0	2.0	0.0	4.0
210	Hombre	19	5.0	3.0	2.0	0.0	2.0
209	Hombre	21	0.0	0.0	0.0	2.0	0.0
208	Hombre	20	0.0	0.0	0.0	0.0	3.0

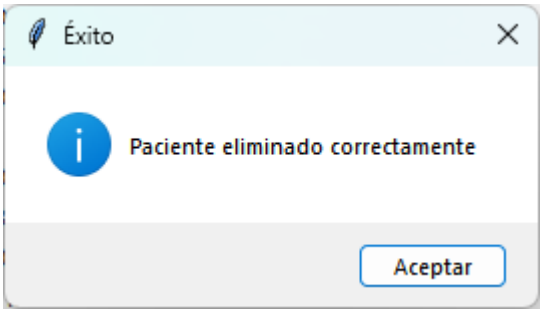
Listado de Pacientes

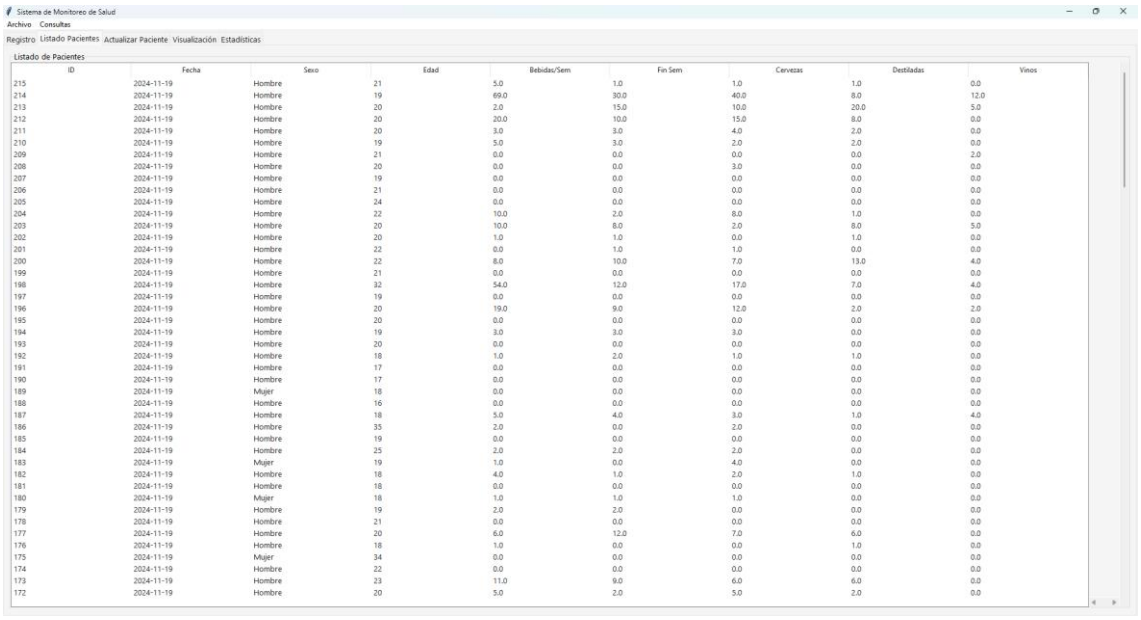
- Permite ver todos los pacientes que hay en la base de datos
- Permite que al darle doble click en un paciente en concreto, salga un mensaje de confirmación para eliminar el paciente.

Resultado:





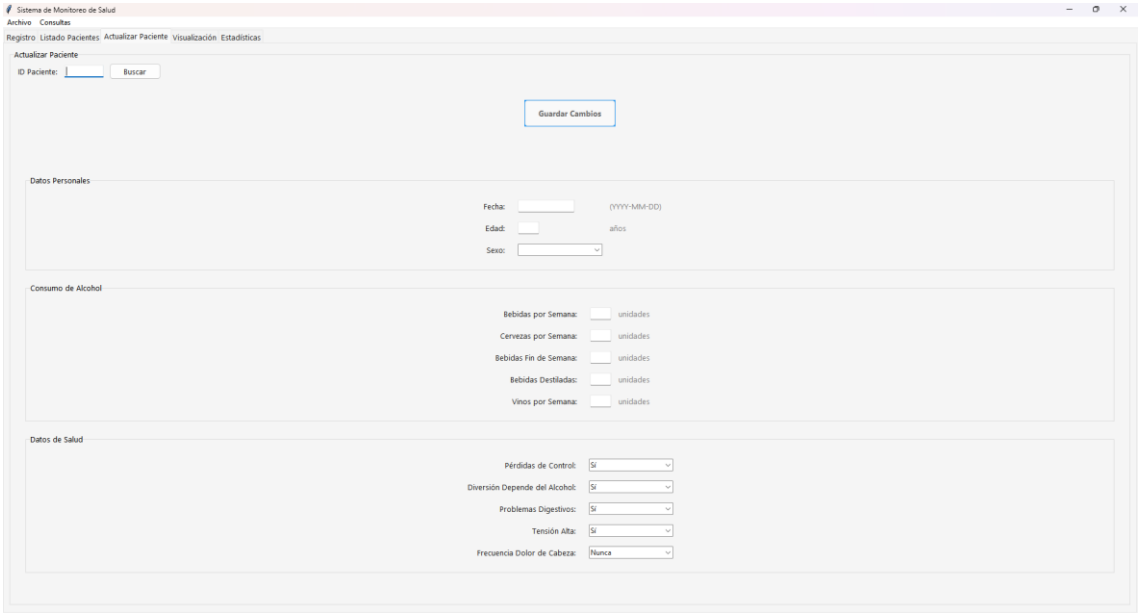


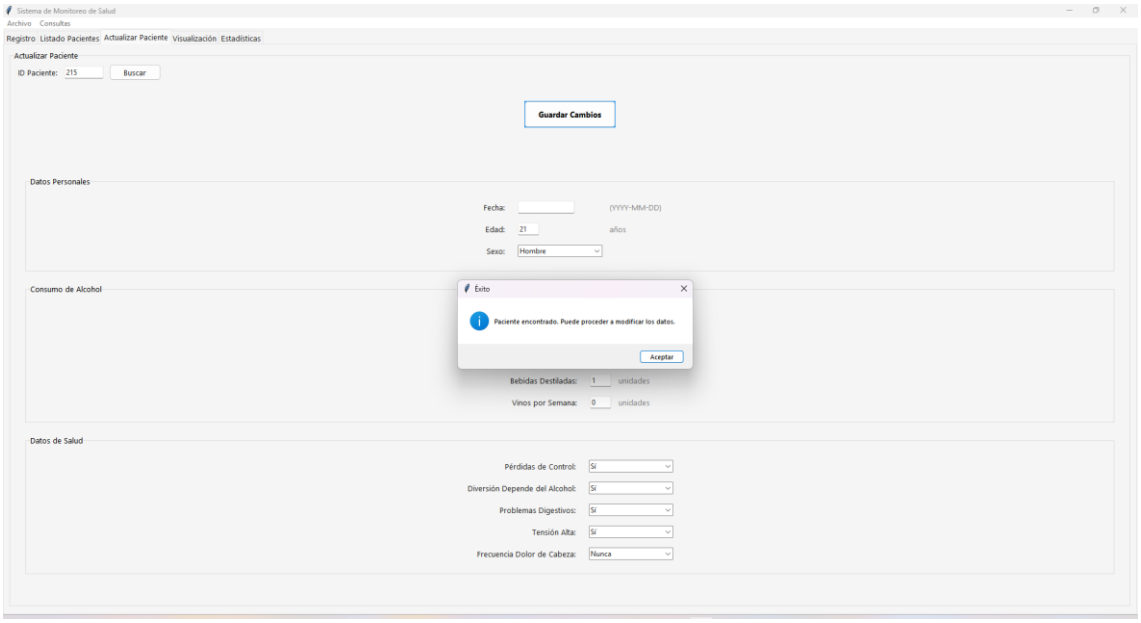


Búsqueda y Actualización de Pacientes

- Permite buscar pacientes por ID y actualizar sus datos.

Resultado:

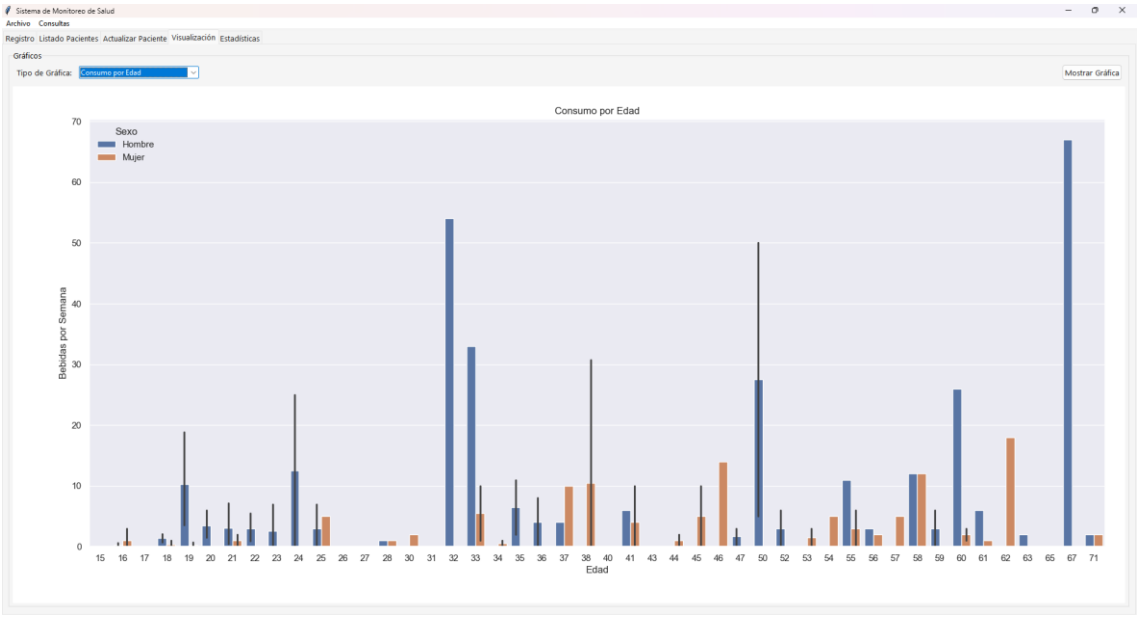


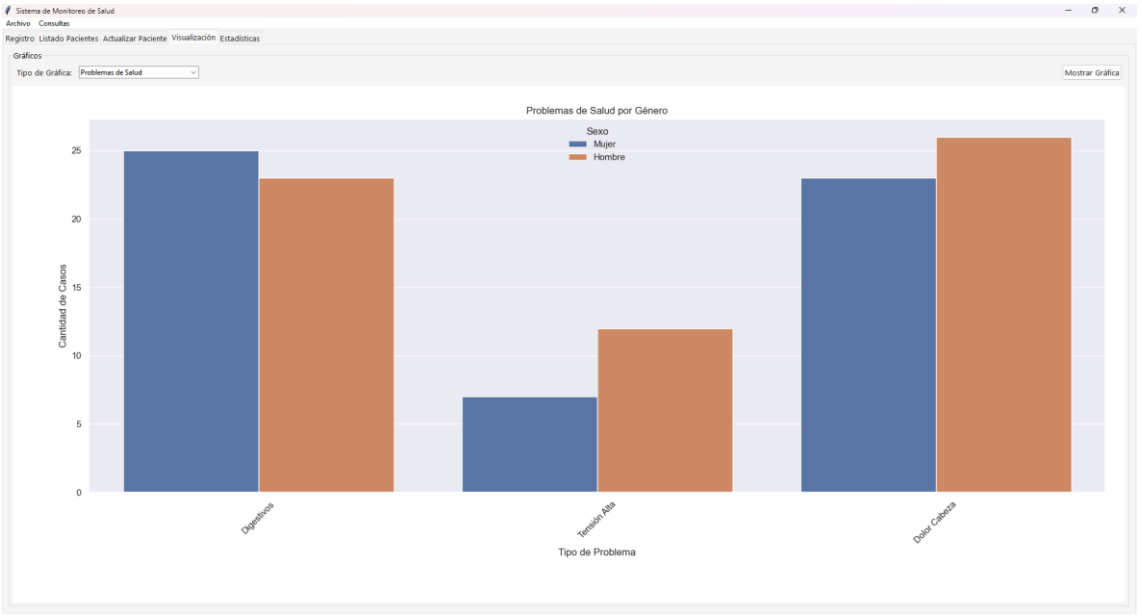


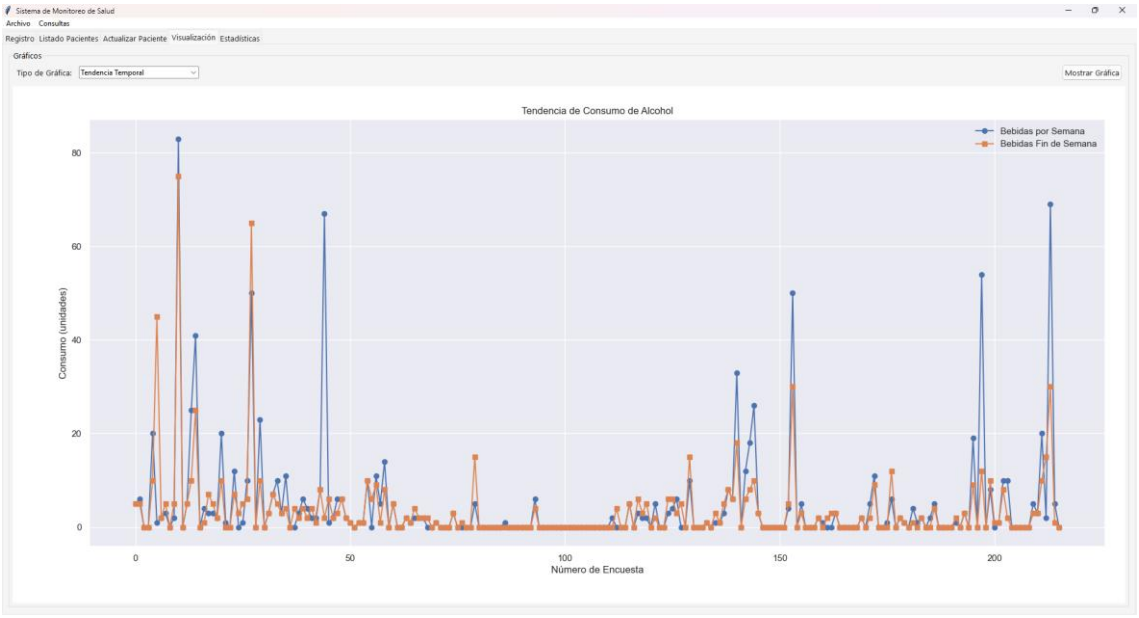
Visualización de Gráficos

- Permite visualizar los resultados de las consultas en diferentes tipos de gráficos (barras, pastel, líneas, etc.).

  Resultado:


Sistema de Monitoreo de Salud
Archivo Consultas
Registro Listado Pacientes Actualizar Paciente Visualización Estadísticas
Gráficos
Tipo de Gráfica: Consumo por Edad
Mostrar Gráfica
Consumo por Edad
Sexo
Hombre
Mujer
Bebidas por Semana
Edad

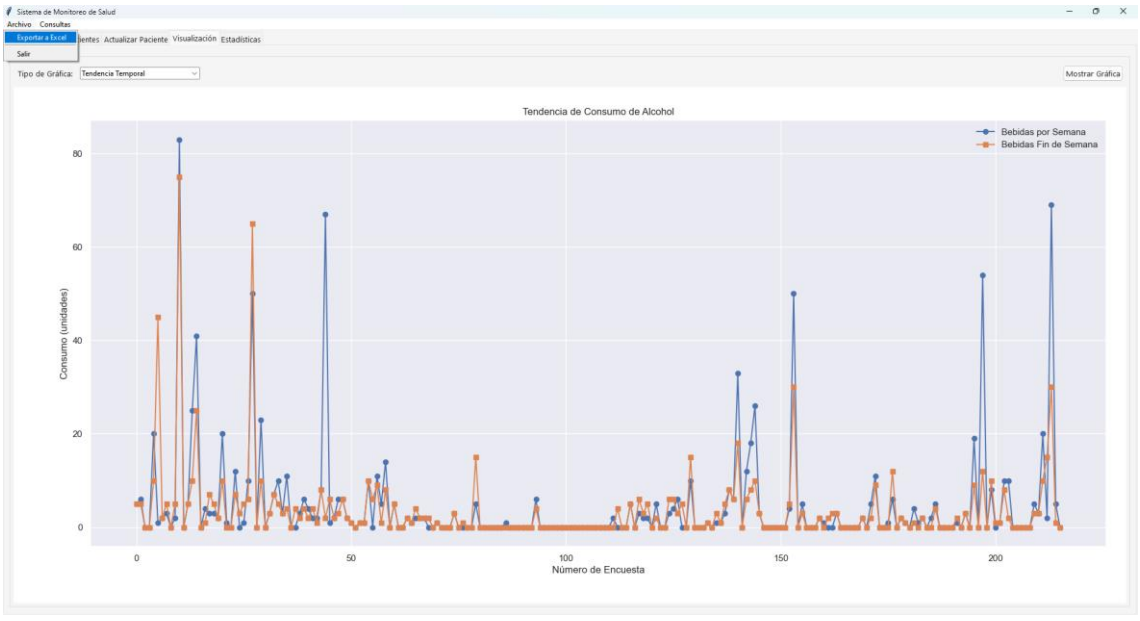

Sistema de Monitoreo de Salud
Archivo Consultas
Registro Listado Pacientes Actualizar Paciente Visualización Estadísticas
Gráficos
Tipo de Gráfica: Problemas de Salud
Mostrar Gráfica
Problemas de Salud por Género
Sexo
Mujer
Hombre
Cantidad de Casos
Digestivos
Tensión Alta
Dolor Cabeza
Tipo de Problema

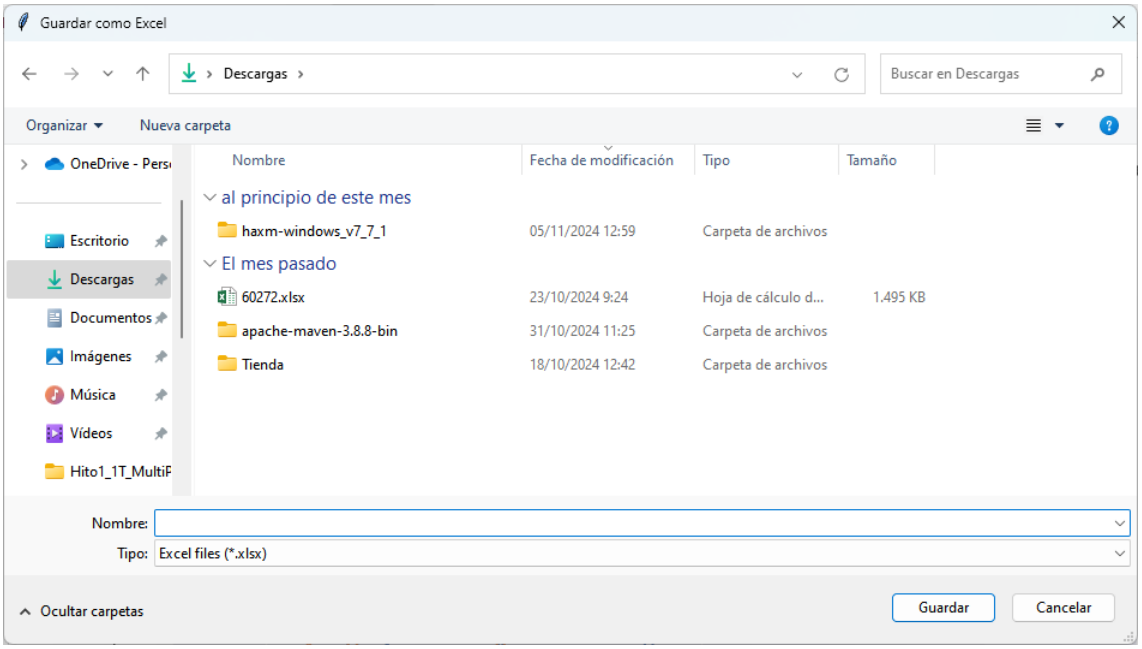


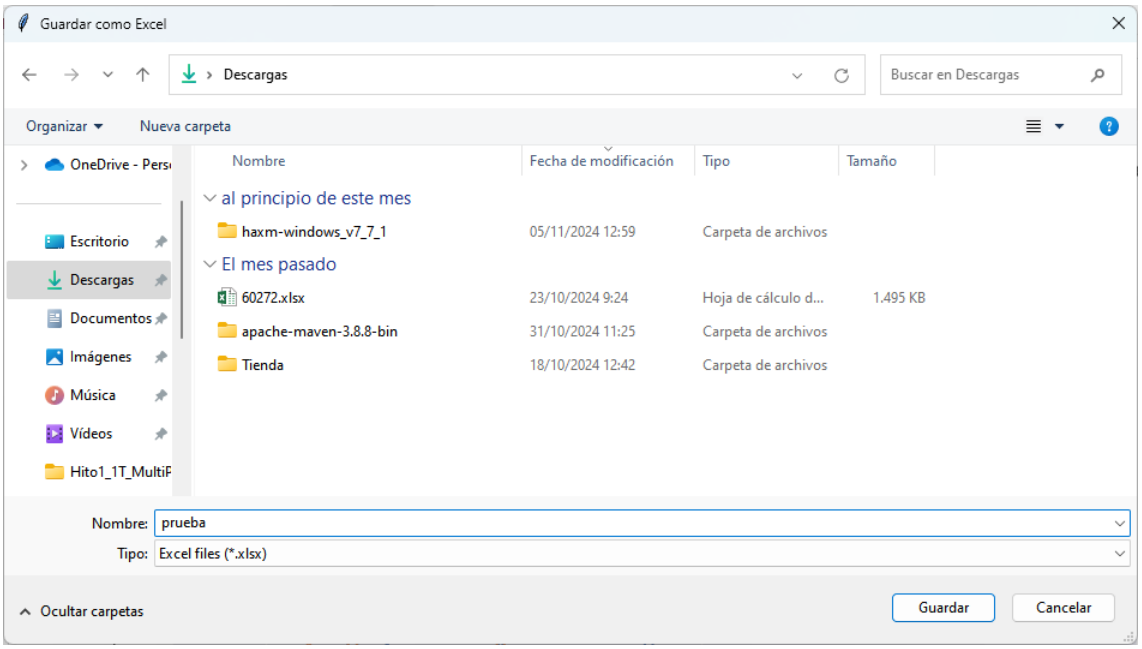
Exportación a Excel

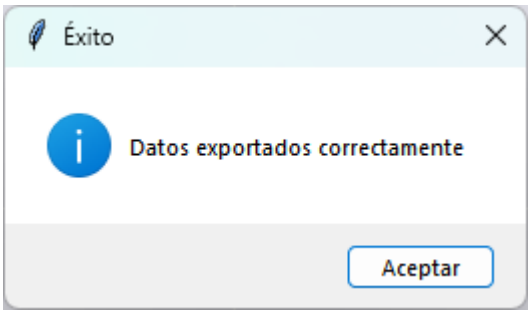
- Permite exportar los datos de las consultas a un archivo Excel para su análisis.

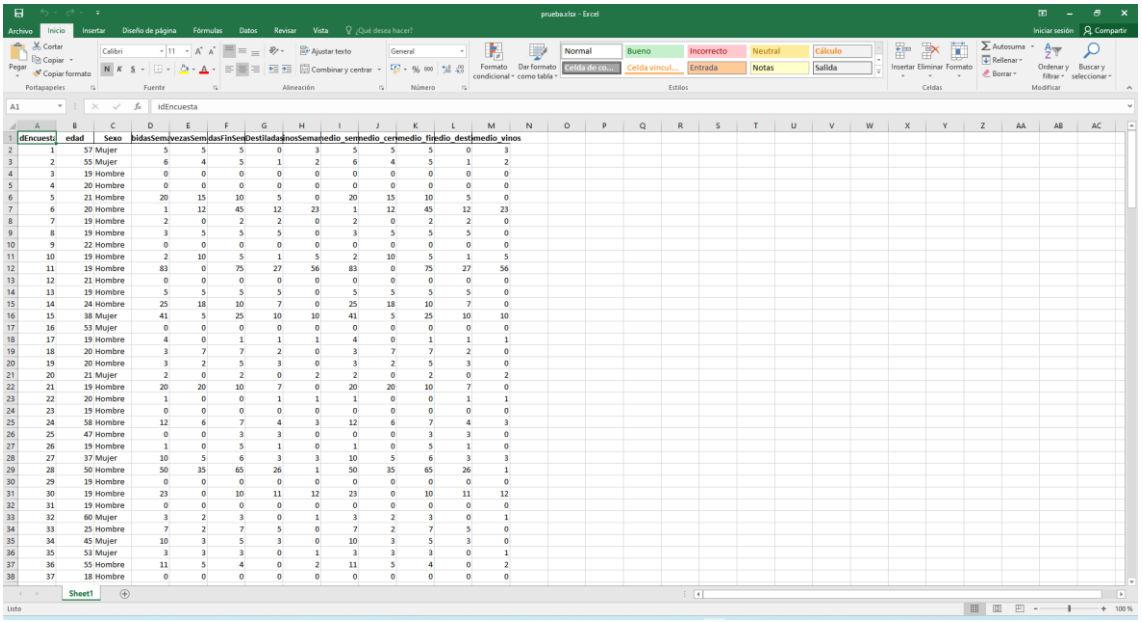
  Resultado:




Guardar como Excel
Descargas
Buscar en Descargas
Organizar
Nueva carpeta
OneDrive - Perso
Escritorio
Descargas
Documentos
Imágenes
Música
Vídeos
Hito1_1T_MultiP
Nombre
Fecha de modificación
Tipo
Tamaño
al principio de este mes
haxm-windows_v7_7_1
05/11/2024 12:59
Carpeta de archivos
El mes pasado
60272.xlsx
23/10/2024 9:24
Hoja de cálculo d...
1.495 KB
apache-maven-3.8.8-bin
31/10/2024 11:25
Carpeta de archivos
Tienda
18/10/2024 12:42
Carpeta de archivos
Nombre:
Tipo:
Excel files (*.xlsx)
Ocultar carpetas
Guardar
Cancelar


Guardar como Excel
Descargas
Buscar en Descargas
Organizar
Nueva carpeta
OneDrive - Perso
Escritorio
Descargas
Documentos
Imágenes
Música
Vídeos
Hito1_1T_MultiP
Nombre
Fecha de modificación
Tipo
Tamaño
al principio de este mes
haxm-windows_v7_7_1
05/11/2024 12:59
Carpeta de archivos
El mes pasado
60272.xlsx
23/10/2024 9:24
Hoja de cálculo d...
1.495 KB
apache-maven-3.8.8-bin
31/10/2024 11:25
Carpeta de archivos
Tienda
18/10/2024 12:42
Carpeta de archivos
Nombre:
prueba
Tipo:
Excel files (*.xlsx)
Ocultar carpetas
Guardar
Cancelar

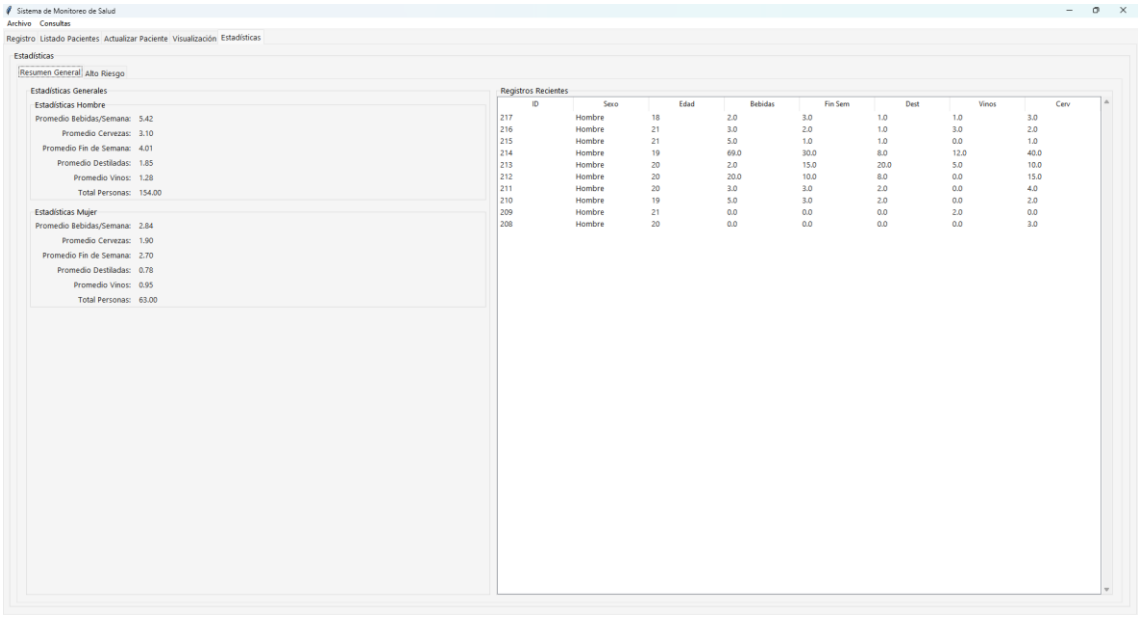

Éxito
Datos exportados correctamente
Aceptar

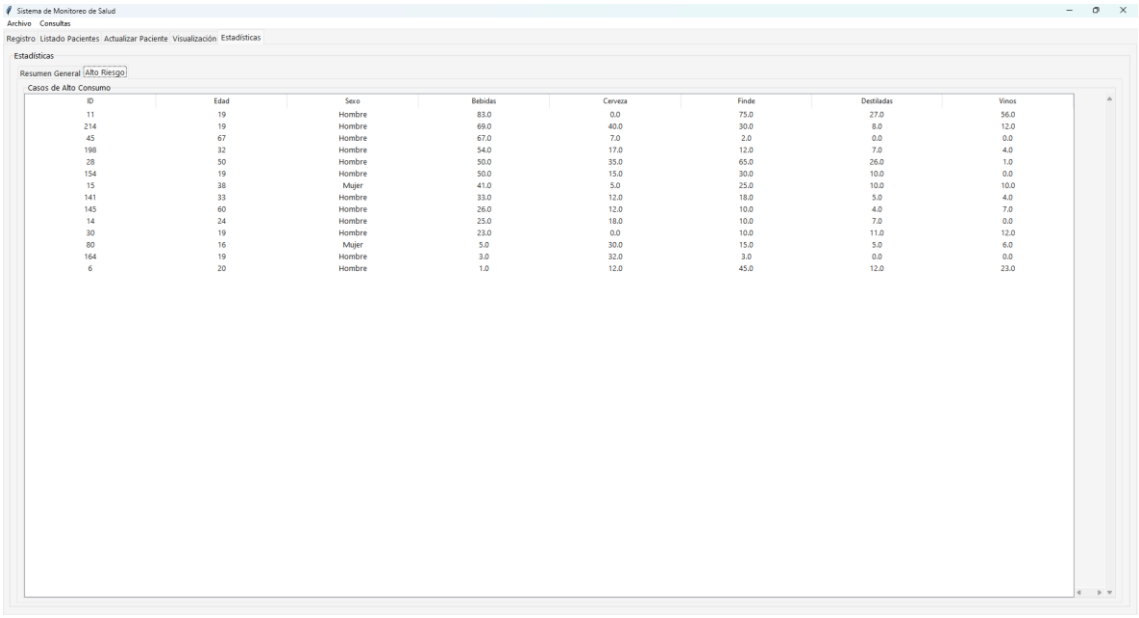


Estadísticas

- Permite visualizar las estadísticas generales de cada sexo, los registros recientes y los casos de alto consumo

  Resultado:





Webgrafía

Python Software Foundation. (2023). *Introducción al API C de Python*. Documentación oficial de Python. Recuperado de https://docs.python.org/es/3.9/c-api/intro.html

ANEXO I

bdd/

__pycache__/

conexion.py

```
# bdd/conexion.py
import mysql.connector
from mysql.connector import Error
from tkinter import messagebox

class ConexionBD:
    def __init__(self):
        try:
            print("Intentando conectar a MySQL...")
            self.conexion = mysql.connector.connect(
                host='localhost',
                user='root',
                password='Tcachuk93',
```

```
                port=3306,
                auth_plugin='mysql_native_password'
            )

            if self.conexion.is_connected():
                db_info = self.conexion.get_server_info()
                print(f"Conexión exitosa a MySQL. Versión del servidor:
{db_info}")
                cursor = self.conexion.cursor()

                # Crear y seleccionar la base de datos
                cursor.execute("CREATE DATABASE IF NOT EXISTS encuestas")
                cursor.execute("USE encuestas")

                print("Base de datos 'encuestas' seleccionada")
                cursor.close()
            else:
                raise Error("No se pudo establecer la conexión")
        except Error as e:
            print(f"Error de conexión: {str(e)}")
            messagebox.showerror("Error de Conexión","Verifica tus
credenciales de MySQL y que el servidor esté activo")
            raise

    def cerrar_conexion(self):
        if self.conexion.is_connected():
            self.conexion.close()
            print("Conexión a MySQL cerrada")
```

consultas.py

```
# bdd/consultas.py
import pandas as pd
from mysql.connector import Error

class ConsultasEncuesta:
    def __init__(self, conexion):
        self.conexion = conexion

    def ordenar_por_campo(self, campo):
        """Obtiene todos los registros ordenados por el campo
especificado"""
        # Mapeo de nombres de columnas
        campo_map = {
            'fecha': 'idEncuesta',  # Usamos idEncuesta en lugar de fecha
            'alcohol': 'BebidasSemana',
```

```
            'presion': None,  # No existe esta columna
            'edad': 'edad',
            'problemas': 'ProblemasDigestivos'
        }

        campo_bd = campo_map.get(campo, campo)
        if not campo_bd:
            raise ValueError(f"Campo {campo} no existe en la base de
datos")

        query = f"""
        SELECT * FROM encuesta
        ORDER BY {campo_bd}
        """
        try:
            return pd.read_sql(query, self.conexion.conexion)
        except Error as e:
            raise Exception(f"Error al ordenar datos: {str(e)}")

    def insertar_encuesta(self, datos):
        query = """
        INSERT INTO encuesta (
            idEncuesta, edad, Sexo,
            BebidasSemana, CervezasSemana, BebidasFinSemana,
            BebidasDestiladasSemana, VinosSemana,
            PerdidasControl, DiversionDependenciaAlcohol,
            ProblemasDigestivos, TensionAlta, DolorCabeza
        ) VALUES (
            %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s
        )
        """
        try:
            cursor = self.conexion.conexion.cursor()

            # Obtener siguiente ID
            cursor.execute("SELECT MAX(idEncuesta) FROM encuesta")
            max_id = cursor.fetchone()[0] or 0
            nuevo_id = max_id + 1

            # Convertir valores Sí/No a 1/0
            def convertir_si_no(valor):
                return 1 if valor.lower() == 'sí' else 0

            valores = (
                nuevo_id,
                int(datos['edad']),
                datos['sexo'],
                float(datos['bebidas_semana']),
                float(datos['cervezas']),
```

```
                float(datos['finde']),
                float(datos['destiladas']),
                float(datos['vinos']),
                convertir_si_no(datos['perdidas_control']),
                convertir_si_no(datos['diversion_alcohol']),
                convertir_si_no(datos['problemas_digestivos']),
                convertir_si_no(datos['tension_alta']),
                datos['dolor_cabeza']
            )

            cursor.execute(query, valores)
            self.conexion.conexion.commit()
            return True

        except Error as e:
            self.conexion.conexion.rollback()
            raise Exception(f"Error al insertar datos: {str(e)}")
        finally:
            cursor.close()

    def obtener_registros_recientes(self, limite=10):
        """
        Obtiene los registros más recientes de la base de datos
        """
        query = """
        SELECT
            idEncuesta,
            Sexo,
            edad,
            BebidasSemana,
            BebidasFinSemana,
            BebidasDestiladasSemana,
            VinosSemana,
            CervezasSemana
        FROM encuesta
        ORDER BY idEncuesta DESC
        LIMIT %s
        """
        try:
            return pd.read_sql(query, self.conexion.conexion,
params=(limite,))
        except Exception as e:
            print(f"Error al obtener registros recientes: {e}")
            return pd.DataFrame()

    def obtener_tendencia_temporal(self):
        """Obtiene la tendencia del consumo por edad"""
        query = """
        SELECT
```

```
            edad,
            AVG(BebidasSemana) as consumo_semanal,
            AVG(BebidasFinSemana) as consumo_finde
        FROM encuesta
        GROUP BY edad
        ORDER BY edad
        """
        return pd.read_sql(query, self.conexion.conexion)

    def obtener_estadisticas_consumo(self):
        """Obtiene estadísticas detalladas de consumo"""
        query = """
        SELECT
            idEncuesta,
            edad,
            Sexo,
            BebidasSemana,
            CervezasSemana,
            BebidasFinSemana,
            BebidasDestiladasSemana,
            VinosSemana,
            AVG(BebidasSemana) as promedio_semanal,
            AVG(CervezasSemana) as promedio_cerveza,
            AVG(BebidasFinSemana) as promedio_finde,
            AVG(BebidasDestiladasSemana) as promedio_destiladas,
            AVG(VinosSemana) as promedio_vinos
        FROM encuesta
        GROUP BY idEncuesta, edad, Sexo, BebidasSemana, CervezasSemana,
                BebidasFinSemana, BebidasDestiladasSemana, VinosSemana
        """
        try:
            return pd.read_sql(query, self.conexion.conexion)
        except Exception as e:
            print(f"Error en obtener_estadisticas_consumo: {str(e)}")
            return pd.DataFrame()  # Retorna DataFrame vacío en caso de
error

    def filtrar_alto_consumo(self, limite=20):
        """Filtra registros con alto consumo de alcohol"""
        query = """
        SELECT
            idEncuesta,
            edad,
            Sexo,
            BebidasSemana,
            CervezasSemana,
            BebidasFinSemana,
            BebidasDestiladasSemana,
            VinosSemana
```

```
        FROM encuesta
        WHERE BebidasSemana > %s
           OR BebidasFinSemana > %s
           OR CervezasSemana > %s
           OR BebidasDestiladasSemana > %s
           OR VinosSemana > %s
        ORDER BY BebidasSemana DESC
        LIMIT 100
        """
        params = (limite,) * 5
        return pd.read_sql(query, self.conexion.conexion, params=params)

    def filtrar_perdidas_control(self, min_perdidas=3):
        """
        Filtra pacientes que han perdido el control más veces que el
mínimo especificado
        Args:
            min_perdidas: Número mínimo de pérdidas de control
        Returns:
            DataFrame con los pacientes filtrados
        """
        query = """
        SELECT
            idEncuesta,
            Sexo,
            edad,
            PerdidasControl,
            BebidasSemana,
            BebidasFinSemana,
            CervezasSemana,
            BebidasDestiladasSemana,
            VinosSemana
        FROM encuesta
        WHERE PerdidasControl >= %s
        ORDER BY PerdidasControl DESC
        """
        try:
            return pd.read_sql(query, self.conexion.conexion,
params=(min_perdidas,))
        except Exception as e:
            print(f"Error al filtrar pérdidas de control: {e}")
            return pd.DataFrame()

    def filtrar_problemas_salud(self):
        """Analiza problemas de salud relacionados con el consumo"""
        query = """
        SELECT
            Sexo,
```

```
            SUM(CASE WHEN ProblemasDigestivos = 'Sí' THEN 1 ELSE 0 END)
as problemas_digestivos,
            SUM(CASE WHEN TensionAlta = 'Sí' THEN 1 ELSE 0 END) as
tension_alta,
            SUM(CASE WHEN DolorCabeza IN ('A menudo', 'Muy a menudo')
THEN 1 ELSE 0 END) as dolor_cabeza_frecuente,
            AVG(BebidasSemana) as promedio_consumo_semanal,
            COUNT(*) as total_casos
        FROM encuesta
        GROUP BY Sexo
        """
        return pd.read_sql(query, self.conexion.conexion)

    def obtener_correlacion_salud_consumo(self):
        """Analiza la correlación entre consumo y problemas de salud"""
        query = """
        SELECT
            CASE
                WHEN BebidasSemana = 0 THEN 'No consume'
                WHEN BebidasSemana <= 5 THEN 'Consumo bajo'
                WHEN BebidasSemana <= 15 THEN 'Consumo moderado'
                ELSE 'Consumo alto'
            END as nivel_consumo,
            COUNT(*) as total_personas,
            SUM(CASE WHEN ProblemasDigestivos = 'Sí' THEN 1 ELSE 0 END)
as casos_digestivos,
            SUM(CASE WHEN TensionAlta = 'Sí' THEN 1 ELSE 0 END) as
casos_tension,
            SUM(CASE WHEN DolorCabeza IN ('A menudo', 'Muy a menudo')
THEN 1 ELSE 0 END) as casos_dolor_cabeza
        FROM encuesta
        GROUP BY
            CASE
                WHEN BebidasSemana = 0 THEN 'No consume'
                WHEN BebidasSemana <= 5 THEN 'Consumo bajo'
                WHEN BebidasSemana <= 15 THEN 'Consumo moderado'
                ELSE 'Consumo alto'
            END
        """
        return pd.read_sql(query, self.conexion.conexion)

    def actualizar_paciente(self, datos):
        """
        Actualiza los datos de un paciente existente
        Args:
            datos: Diccionario con los datos del paciente
        """
        query = """
        UPDATE encuesta
```

```
        SET Sexo = %s,
            edad = %s,
            BebidasSemana = %s,
            CervezasSemana = %s,
            BebidasFinSemana = %s,
            BebidasDestiladasSemana = %s,
            VinosSemana = %s,
            PerdidasControl = %s,
            DiversionDependenciaAlcohol = %s,
            ProblemasDigestivos = %s,
            TensionAlta = %s,
            DolorCabeza = %s
        WHERE idEncuesta = %s
        """
        try:
            def convertir_si_no(valor):
                return 1 if valor else 0

            cursor = self.conexion.conexion.cursor()
            values = (
                datos['sexo'],
                int(datos['edad']),
                float(datos['bebidas_semana']),
                float(datos['cervezas']),
                float(datos['finde']),
                float(datos['destiladas']),
                float(datos['vinos']),
                convertir_si_no(datos['perdidas_control']),
                convertir_si_no(datos['diversion_alcohol']),
                convertir_si_no(datos['problemas_digestivos']),
                convertir_si_no(datos['tension_alta']),
                datos['dolor_cabeza'],
                int(datos['id'])
            )

            cursor.execute(query, values)
            self.conexion.conexion.commit()
            return True
        except Exception as e:
            self.conexion.conexion.rollback()
            raise Exception(f"Error al actualizar paciente: {str(e)}")
        finally:
            cursor.close()


    def obtener_listado_pacientes(self):
        """
        Obtiene el listado completo de pacientes de la base de datos
        :return: DataFrame con todos los pacientes
```

```
        """
        query = """
        SELECT
            idEncuesta,
            NOW() as Fecha,  # Usamos NOW() como fecha temporal
            Sexo,
            edad,
            BebidasSemana,
            BebidasFinSemana,
            CervezasSemana,
            BebidasDestiladasSemana,
            VinosSemana
        FROM encuesta
        ORDER BY idEncuesta DESC
        """
        try:
            return pd.read_sql(query, self.conexion.conexion)
        except Exception as e:
            print(f"Error al obtener listado de pacientes: {e}")
            raise Exception(f"Error al obtener listado de pacientes:
{str(e)}")

    def obtener_paciente_por_id(self, id_paciente):
        """
        Obtiene los datos de un paciente por su ID
        Args:
            id_paciente: ID del paciente a buscar
        Returns:
            dict: Diccionario con los datos del paciente o None si no se
encuentra
        """
        query = """
        SELECT
            idEncuesta,
            Sexo,
            edad,
            BebidasSemana,
            BebidasFinSemana,
            CervezasSemana,
            BebidasDestiladasSemana,
            VinosSemana,
            PerdidasControl,
            DiversionDependenciaAlcohol,
            ProblemasDigestivos,
            TensionAlta,
            DolorCabeza
        FROM encuesta
        WHERE idEncuesta = %s
        """
```

```
        try:
            with self.conexion.conexion.cursor(dictionary=True) as
cursor:
                cursor.execute(query, (id_paciente,))
                return cursor.fetchone()
        except Exception as e:
            raise Exception(f"Error al obtener paciente: {str(e)}")

    def eliminar_paciente(self, id_paciente):
        """
        Elimina un paciente de la base de datos
        Args:
            id_paciente: ID del paciente a eliminar
        """
        try:
            cursor = self.conexion.conexion.cursor()
            query = "DELETE FROM encuesta WHERE idEncuesta = %s"
            cursor.execute(query, (id_paciente,))
            self.conexion.conexion.commit()
            cursor.close()
            return True
        except Exception as e:
            self.conexion.conexion.rollback()
            raise Exception(f"No se pudo eliminar el paciente: {str(e)}")

    def exportar_a_excel(self, df, nombre_archivo):
        """Exporta un DataFrame a Excel"""
        try:
            df.to_excel(nombre_archivo, index=False)
            return True
        except Exception as e:
            print(f"Error al exportar: {e}")
            return False
```

graficas/

__pycache__/

visualizargraficas.py

```
# graficas/visualizargraficas.py
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from tkinter import messagebox
```

```
class VisualizadorGraficas:
    def __init__(self):
        plt.style.use('seaborn-v0_8-darkgrid')
        sns.set_theme()
        plt.rcParams['figure.figsize'] = [10, 6]
        plt.rcParams['figure.dpi'] = 100

    def crear_grafica_consumo_edad(self, datos):
        if datos is None or datos.empty:
            return None

        fig, ax = plt.subplots()
        sns.barplot(
            data=datos,
            x='edad',
            y='promedio_semanal',
            hue='Sexo',
            ax=ax
        )
        ax.set_title('Consumo por Edad')
        ax.set_xlabel('Edad')
        ax.set_ylabel('Bebidas por Semana')
        plt.tight_layout()
        return fig

    def crear_grafica_tendencia_temporal(self, datos):
        if datos is None or datos.empty:
            return None

        try:
            # Usar las columnas correctas que vienen de la base de datos
            fig = plt.figure(figsize=(10, 6))
            plt.plot(datos.index, datos['BebidasSemana'], marker='o',
label='Bebidas por Semana')
            plt.plot(datos.index, datos['BebidasFinSemana'], marker='s',
label='Bebidas Fin de Semana')

            plt.title('Tendencia de Consumo de Alcohol')
            plt.xlabel('Número de Encuesta')
            plt.ylabel('Consumo (unidades)')
            plt.legend()
            plt.grid(True)
            plt.tight_layout()
            return fig
        except Exception as e:
            print(f"Error al crear gráfica de tendencia: {str(e)}")
            return None
```

```
    def crear_grafica_problemas_salud(self, datos):
        if datos is None or datos.empty:
            return None

        # Transformar los datos para la visualización
        datos_melt = datos.melt(
            id_vars=['Sexo'],
            value_vars=[
                'problemas_digestivos',
                'tension_alta',
                'dolor_cabeza_frecuente'
            ],
            var_name='Problema',
            value_name='Cantidad'
        )

        fig, ax = plt.subplots()
        sns.barplot(
            data=datos_melt,
            x='Problema',
            y='Cantidad',
            hue='Sexo',
            ax=ax
        )

        # Personalizar la gráfica
        ax.set_title('Problemas de Salud por Género')
        ax.set_xlabel('Tipo de Problema')
        ax.set_ylabel('Cantidad de Casos')

        # Mejorar las etiquetas
        etiquetas = {
            'problemas_digestivos': 'Digestivos',
            'tension_alta': 'Tensión Alta',
            'dolor_cabeza_frecuente': 'Dolor Cabeza'
        }
        ax.set_xticklabels([etiquetas[x.get_text()] for x in
ax.get_xticklabels()])

        plt.xticks(rotation=45)
        plt.tight_layout()
        return fig

    def crear_grafica_tendencia(self, datos):
        if datos is None or datos.empty:
            return None

        fig, ax = plt.subplots()
        sns.lineplot(
```

```
            data=datos,
            x='idEncuesta',
            y='BebidasSemana',
            ax=ax
        )
        ax.set_title('Tendencia de Consumo')
        ax.set_xlabel('ID Encuesta')
        ax.set_ylabel('Bebidas por Semana')
        plt.tight_layout()
        return fig
```

interfaz/

__pycache__/

interfazusuario.py

```
# interfazusuario.py
import tkinter as tk
from tkinter import ttk, messagebox
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from datetime import datetime

class InterfazSaludAlcohol:
    def __init__(self, root):
        self.root = root
        self.root.title("Sistema de Monitoreo de Salud")

        # Inicializar campos de entrada
        self.entrada_fecha = None
        self.entrada_sexo = None
        self.entrada_edad = None
        self.entrada_alcohol = None
        self.entrada_presion = None
        self.entrada_problemas = None
        self.entrada_bebidas_semana = None
        self.entrada_cervezas = None
        self.entrada_finde = None
        self.entrada_destiladas = None
        self.entrada_vinos = None
        self.entrada_perdidas_control = None
        self.entrada_diversion_alcohol = None
        self.entrada_problemas_digestivos = None
```

```
        self.entrada_tension_alta = None
        self.entrada_dolor_cabeza = None

        # Campos para actualización
        self.entrada_id_busqueda = None
        self.entrada_fecha_act = None
        self.entrada_edad_act = None
        self.entrada_sexo_act = None
        self.entrada_bebidas_semana_act = None
        self.entrada_cervezas_act = None
        self.entrada_finde_act = None
        self.entrada_destiladas_act = None
        self.entrada_vinos_act = None
        self.entrada_perdidas_control_act = None
        self.entrada_diversion_alcohol_act = None
        self.entrada_problemas_digestivos_act = None
        self.entrada_tension_alta_act = None
        self.entrada_dolor_cabeza_act = None

        # Inicializar callbacks
        self._registrar_callback = None
        self._visualizar_callback = None
        self._estadisticas_callback = None
        self._exportar_callback = None
        self._buscar_callback = None
        self._actualizar_callback = None
        self._eliminar_callback = None

        # Crear componentes principales
        self.crear_notebook()
        self.crear_menu()
        self.crear_panel_registro()
        self.crear_panel_visualizacion()
        self.crear_panel_estadisticas()
        self.crear_panel_actualizar()
        self.crear_panel_listado()

        # Inicializar canvas
        self.canvas = None

    def crear_notebook(self):
        self.notebook = ttk.Notebook(self.root)
        self.notebook.pack(fill='both', expand=True)

        self.tab_registro = ttk.Frame(self.notebook)
        self.tab_visualizacion = ttk.Frame(self.notebook)
        self.tab_estadisticas = ttk.Frame(self.notebook)
        self.tab_listado = ttk.Frame(self.notebook)  # Nueva pestaña
```

```
        self.notebook.add(self.tab_registro, text='Registro')
        self.notebook.add(self.tab_listado, text='Listado Pacientes')
        self.tab_actualizar = ttk.Frame(self.notebook)
        self.notebook.add(self.tab_actualizar, text='Actualizar
Paciente')
        self.notebook.add(self.tab_visualizacion, text='Visualización')
        self.notebook.add(self.tab_estadisticas, text='Estadísticas')

    def crear_menu(self):
        menu_bar = tk.Menu(self.root)
        self.root.config(menu=menu_bar)

        # Menú Archivo
        file_menu = tk.Menu(menu_bar, tearoff=0)
        menu_bar.add_cascade(label="Archivo", menu=file_menu)
        file_menu.add_command(label="Exportar a Excel",
                              command=lambda:
self._exportar_callback("excel") if self._exportar_callback else None)
        file_menu.add_separator()
        file_menu.add_command(label="Salir", command=self.root.quit)

        # Menú Consultas
        query_menu = tk.Menu(menu_bar, tearoff=0)
        menu_bar.add_cascade(label="Consultas", menu=query_menu)
        query_menu.add_command(label="Ver Estadísticas",
                              command=lambda:
self.notebook.select(self.tab_estadisticas))
        query_menu.add_command(label="Actualizar Paciente",
                              command=lambda:
self.notebook.select(self.tab_actualizar))

    def _crear_seccion_campos(self, titulo, campos, frame_principal):
        """
        Crea una sección de campos en el formulario
        Args:
            titulo (str): Título de la sección
            campos (list): Lista de tuplas con la información de los
campos
            frame_principal (ttk.Frame): Frame contenedor
        """
        frame = ttk.LabelFrame(frame_principal, text=titulo,
style="Custom.TLabelframe")
        frame.pack(fill='x', padx=20, pady=10)

        frame_grid = ttk.Frame(frame)
        frame_grid.pack(padx=15, pady=15)

        for i, campo in enumerate(campos):
```

```
            if len(campo) == 4:  # Campo con etiqueta, nombre, ancho y
hint
                label, attr, width, hint = campo
                ttk.Label(frame_grid, text=label).grid(row=i, column=0,
padx=10, pady=8, sticky='e')
                setattr(self, attr, ttk.Entry(frame_grid, width=width))
                getattr(self, attr).grid(row=i, column=1, padx=10,
pady=8, sticky='w')
                if hint:
                    ttk.Label(frame_grid, text=hint,
foreground="gray").grid(row=i, column=2, padx=5, pady=8, sticky='w')

            elif len(campo) == 3:  # Campo con etiqueta, nombre y
opciones
                label, attr, opciones = campo
                ttk.Label(frame_grid, text=label).grid(row=i, column=0,
padx=10, pady=8, sticky='e')
                setattr(self, attr, ttk.Combobox(frame_grid,
values=opciones, state='readonly'))
                getattr(self, attr).grid(row=i, column=1, padx=10,
pady=8, sticky='w')
                getattr(self, attr).set(opciones[0])

    def crear_panel_registro(self):
        """Crea un panel de registro mejorado y más intuitivo"""
        frame_principal = ttk.Frame(self.tab_registro,
style="Custom.TLabelframe")
        frame_principal.pack(padx=30, pady=20, fill='both', expand=True)

        # Título principal
        titulo = ttk.Label(frame_principal,
                           text="REGISTRO DE PACIENTE",
                           style="Title.TLabel")
        titulo.pack(pady=(0, 20))

        # Sección de datos personales
        frame_datos = ttk.LabelFrame(frame_principal, text="Datos
Personales")
        frame_datos.pack(fill='x', padx=20, pady=10)

        frame_grid = ttk.Frame(frame_datos)
        frame_grid.pack(padx=15, pady=15)

        # Datos Personales
        ttk.Label(frame_grid, text="Fecha:").grid(row=0, column=0,
padx=10, pady=8, sticky='e')
        self.entrada_fecha = ttk.Entry(frame_grid, width=15)
        self.entrada_fecha.grid(row=0, column=1, padx=10, pady=8,
sticky='w')
```

```
        ttk.Label(frame_grid, text="(YYYY-MM-DD)",
foreground="gray").grid(row=0, column=2, pady=8, sticky='w')

        ttk.Label(frame_grid, text="Edad:").grid(row=1, column=0,
padx=10, pady=8, sticky='e')
        self.entrada_edad = ttk.Entry(frame_grid, width=5)
        self.entrada_edad.grid(row=1, column=1, padx=10, pady=8,
sticky='w')
        ttk.Label(frame_grid, text="años", foreground="gray").grid(row=1,
column=2, pady=8, sticky='w')

        ttk.Label(frame_grid, text="Sexo:").grid(row=2, column=0,
padx=10, pady=8, sticky='e')
        self.entrada_sexo = ttk.Combobox(frame_grid, values=["Hombre",
"Mujer"], state='readonly')
        self.entrada_sexo.grid(row=2, column=1, padx=10, pady=8,
sticky='w')
        self.entrada_sexo.set("Hombre")

        # Sección de Consumo de Alcohol
        frame_consumo = ttk.LabelFrame(frame_principal, text="Consumo de
Alcohol")
        frame_consumo.pack(fill='x', padx=20, pady=10)
        frame_grid_consumo = ttk.Frame(frame_consumo)
        frame_grid_consumo.pack(padx=15, pady=15)

        # Campos de Consumo
        campos_consumo = [
            ("Bebidas por Semana:", "bebidas_semana", 5),
            ("Cervezas por Semana:", "cervezas", 5),
            ("Bebidas Fin de Semana:", "finde", 5),
            ("Bebidas Destiladas:", "destiladas", 5),
            ("Vinos por Semana:", "vinos", 5)
        ]

        for i, (label, nombre, width) in enumerate(campos_consumo):
            ttk.Label(frame_grid_consumo, text=label).grid(row=i,
column=0, padx=10, pady=8, sticky='e')
            setattr(self, f'entrada_{nombre}',
ttk.Entry(frame_grid_consumo, width=width))
            getattr(self, f'entrada_{nombre}').grid(row=i, column=1,
padx=10, pady=8, sticky='w')
            ttk.Label(frame_grid_consumo, text="unidades",
foreground="gray").grid(row=i, column=2, pady=8, sticky='w')

        # Sección de Salud
        frame_salud = ttk.LabelFrame(frame_principal, text="Datos de
Salud")
        frame_salud.pack(fill='x', padx=20, pady=10)
```

```
        frame_grid_salud = ttk.Frame(frame_salud)
        frame_grid_salud.pack(padx=15, pady=15)

        # Campos de Salud
        campos_salud = [
            ("Pérdidas de Control:", "perdidas_control", ["Sí", "No"]),
            ("Diversión Depende del Alcohol:", "diversion_alcohol",
["Sí", "No"]),
            ("Problemas Digestivos:", "problemas_digestivos", ["Sí",
"No"]),
            ("Tensión Alta:", "tension_alta", ["Sí", "No"]),
            ("Frecuencia Dolor de Cabeza:", "dolor_cabeza",
             ["Nunca", "Raramente", "A menudo", "Muy a menudo"])
        ]

        for i, (label, nombre, opciones) in enumerate(campos_salud):
            ttk.Label(frame_grid_salud, text=label).grid(row=i, column=0,
padx=10, pady=8, sticky='e')
            setattr(self, f'entrada_{nombre}',
ttk.Combobox(frame_grid_salud, values=opciones, state='readonly'))
            getattr(self, f'entrada_{nombre}').grid(row=i, column=1,
padx=10, pady=8, sticky='w')
            getattr(self, f'entrada_{nombre}').set(opciones[0])

        # Frame para botones
        frame_botones = ttk.Frame(frame_principal)
        frame_botones.pack(pady=25)

        # Botones mejorados
        ttk.Button(frame_botones,
                text="Limpiar Formulario",
                style="Secondary.TButton",
                command=self.limpiar_formulario).pack(side='left',
padx=15)
        ttk.Button(frame_botones,
                text="Registrar Paciente",
                style="Primary.TButton",
                command=self._on_registrar).pack(side='left', padx=15)

        # Configurar validación después de crear todos los widgets
        self._configurar_validacion()

    def crear_panel_listado(self):
        frame = ttk.LabelFrame(self.tab_listado, text="Listado de
Pacientes")
        frame.pack(padx=10, pady=10, fill='both', expand=True)

        # Crear Treeview para el listado
```

```
        columns = ('ID', 'Fecha', 'Sexo', 'Edad', 'Bebidas/Sem', 'Fin
Sem', 'Cervezas', 'Destiladas', 'Vinos')
        self.tree_pacientes = ttk.Treeview(frame, columns=columns,
show='headings', height=15)

        # Configurar columnas
        for col in columns:
            self.tree_pacientes.heading(col, text=col)
            self.tree_pacientes.column(col, width=80)

        # Scrollbars
        scrolly = ttk.Scrollbar(frame, orient='vertical',
command=self.tree_pacientes.yview)
        scrollx = ttk.Scrollbar(frame, orient='horizontal',
command=self.tree_pacientes.xview)
        self.tree_pacientes.configure(yscrollcommand=scrolly.set,
xscrollcommand=scrollx.set)

        # Evento de doble clic para eliminar
        self.tree_pacientes.bind('<Double-1>',
self._on_doble_click_paciente)

        # Empaquetar componentes
        self.tree_pacientes.pack(side='left', fill='both', expand=True)
        scrolly.pack(side='right', fill='y')
        scrollx.pack(side='bottom', fill='x')

    def crear_panel_actualizar(self):
        frame = ttk.LabelFrame(self.tab_actualizar, text="Actualizar
Paciente")
        frame.pack(padx=10, pady=10, fill='both', expand=True)

        # Frame búsqueda
        frame_busqueda = ttk.Frame(frame)
        frame_busqueda.pack(fill='x', padx=5, pady=5)

        ttk.Label(frame_busqueda, text="ID Paciente:").pack(side='left',
padx=5)
        self.entrada_id_busqueda = ttk.Entry(frame_busqueda, width=10)
        self.entrada_id_busqueda.pack(side='left', padx=5)

        ttk.Button(frame_busqueda,
                text="Buscar",
                command=self._buscar_paciente).pack(side='left', padx=5)

        # Frame principal para los datos
        self.frame_actualizar = ttk.Frame(frame)
        self.frame_actualizar.pack(fill='both', expand=True, padx=5,
pady=5)
```

```
        # Desactivar frame de actualización hasta que se busque un
paciente
        for child in self.frame_actualizar.winfo_children():
            child.configure(state='disabled')

        # Frame principal para los datos
        frame_principal = ttk.Frame(frame)
        frame_principal.pack(fill='both', expand=True, padx=5, pady=5)

        # Sección de datos personales
        frame_datos = ttk.LabelFrame(frame_principal, text="Datos
Personales")
        frame_datos.pack(fill='x', padx=20, pady=10)

        frame_grid = ttk.Frame(frame_datos)
        frame_grid.pack(padx=15, pady=15)

        # Datos Personales
        ttk.Label(frame_grid, text="Fecha:").grid(row=0, column=0,
padx=10, pady=8, sticky='e')
        self.entrada_fecha_act = ttk.Entry(frame_grid, width=15)
        self.entrada_fecha_act.grid(row=0, column=1, padx=10, pady=8,
sticky='w')
        ttk.Label(frame_grid, text="(YYYY-MM-DD)",
foreground="gray").grid(row=0, column=2, pady=8, sticky='w')

        ttk.Label(frame_grid, text="Edad:").grid(row=1, column=0,
padx=10, pady=8, sticky='e')
        self.entrada_edad_act = ttk.Entry(frame_grid, width=5)
        self.entrada_edad_act.grid(row=1, column=1, padx=10, pady=8,
sticky='w')
        ttk.Label(frame_grid, text="años", foreground="gray").grid(row=1,
column=2, pady=8, sticky='w')

        ttk.Label(frame_grid, text="Sexo:").grid(row=2, column=0,
padx=10, pady=8, sticky='e')
        self.entrada_sexo_act = ttk.Combobox(frame_grid,
values=["Hombre", "Mujer"], state='readonly')
        self.entrada_sexo_act.grid(row=2, column=1, padx=10, pady=8,
sticky='w')

        # Sección de Consumo de Alcohol
        frame_consumo = ttk.LabelFrame(frame_principal, text="Consumo de
Alcohol")
        frame_consumo.pack(fill='x', padx=20, pady=10)
        frame_grid_consumo = ttk.Frame(frame_consumo)
        frame_grid_consumo.pack(padx=15, pady=15)
```

```
        # Campos de Consumo
        campos_consumo = [
            ("Bebidas por Semana:", "bebidas_semana_act", 5),
            ("Cervezas por Semana:", "cervezas_act", 5),
            ("Bebidas Fin de Semana:", "finde_act", 5),
            ("Bebidas Destiladas:", "destiladas_act", 5),
            ("Vinos por Semana:", "vinos_act", 5)
        ]

        for i, (label, nombre, width) in enumerate(campos_consumo):
            ttk.Label(frame_grid_consumo, text=label).grid(row=i,
column=0, padx=10, pady=8, sticky='e')
            setattr(self, f'entrada_{nombre}',
ttk.Entry(frame_grid_consumo, width=width))
            getattr(self, f'entrada_{nombre}').grid(row=i, column=1,
padx=10, pady=8, sticky='w')
            ttk.Label(frame_grid_consumo, text="unidades",
foreground="gray").grid(row=i, column=2, pady=8, sticky='w')

        # Sección de Salud
        frame_salud = ttk.LabelFrame(frame_principal, text="Datos de
Salud")
        frame_salud.pack(fill='x', padx=20, pady=10)
        frame_grid_salud = ttk.Frame(frame_salud)
        frame_grid_salud.pack(padx=15, pady=15)

        # Campos de Salud
        campos_salud = [
            ("Pérdidas de Control:", "perdidas_control_act", ["Sí",
"No"]),
            ("Diversión Depende del Alcohol:", "diversion_alcohol_act",
["Sí", "No"]),
            ("Problemas Digestivos:", "problemas_digestivos_act", ["Sí",
"No"]),
            ("Tensión Alta:", "tension_alta_act", ["Sí", "No"]),
            ("Frecuencia Dolor de Cabeza:", "dolor_cabeza_act",
             ["Nunca", "Raramente", "A menudo", "Muy a menudo"])
        ]

        for i, (label, nombre, opciones) in enumerate(campos_salud):
            ttk.Label(frame_grid_salud, text=label).grid(row=i, column=0,
padx=10, pady=8, sticky='e')
            setattr(self, f'entrada_{nombre}',
ttk.Combobox(frame_grid_salud, values=opciones, state='readonly'))
            getattr(self, f'entrada_{nombre}').grid(row=i, column=1,
padx=10, pady=8, sticky='w')
            getattr(self, f'entrada_{nombre}').set(opciones[0])

        # Frame para botones
```

```
        frame_botones = ttk.Frame(self.frame_actualizar)
        frame_botones.pack(pady=25)

        # Botón para guardar cambios
        self.boton_guardar = ttk.Button(frame_botones,
                                        text="Guardar Cambios",
                                        style="Primary.TButton",
                                        command=self._guardar_actualizacion,
                                        state='disabled')
        self.boton_guardar.pack(side='left', padx=15)

    def registrar_buscar_callback(self, callback):
        """Registra el callback para búsqueda de pacientes"""
        if callable(callback):
            self._buscar_callback = callback

    def registrar_actualizar_callback(self, callback):
        """Registra el callback para actualización de pacientes"""
        if callable(callback):
            self._actualizar_callback = callback

    def _buscar_paciente(self):
        if hasattr(self, '_buscar_callback') and
callable(self._buscar_callback):
            id_paciente = self.entrada_id_busqueda.get().strip()
            if id_paciente:
                self._buscar_callback(id_paciente)

    def _guardar_actualizacion(self):
        if hasattr(self, '_actualizar_callback') and
callable(self._actualizar_callback):
            datos = self.obtener_datos_formulario_actualizacion()
            if datos:
                self._actualizar_callback(datos)

    def mostrar_datos_paciente(self, datos):
        if datos is not None:
            try:
                # Habilitar campos para edición
                for widget in self.frame_actualizar.winfo_children():
                    try:
                        widget.configure(state='normal')
                    except:
                        pass

                # Rellenar campos con los datos existentes
                self.entrada_edad_act.delete(0, tk.END)
                self.entrada_edad_act.insert(0, str(datos['edad']))
```

```
                self.entrada_sexo_act.set(datos['Sexo'])

                self.entrada_bebidas_semana_act.delete(0, tk.END)
                self.entrada_bebidas_semana_act.insert(0,
str(datos['BebidasSemana']))

                self.entrada_cervezas_act.delete(0, tk.END)
                self.entrada_cervezas_act.insert(0,
str(datos['CervezasSemana']))

                self.entrada_finde_act.delete(0, tk.END)
                self.entrada_finde_act.insert(0,
str(datos['BebidasFinSemana']))

                self.entrada_destiladas_act.delete(0, tk.END)
                self.entrada_destiladas_act.insert(0,
str(datos['BebidasDestiladasSemana']))

                self.entrada_vinos_act.delete(0, tk.END)
                self.entrada_vinos_act.insert(0,
str(datos['VinosSemana']))

                # Habilitar botón de guardar
                self.boton_guardar.configure(state='normal')

            except Exception as e:
                messagebox.showerror("Error", f"Error al mostrar datos
del paciente: {str(e)}")

    def obtener_datos_formulario_actualizacion(self):
        """
        Obtiene los datos del formulario de actualización
        Returns:
            dict: Diccionario con los datos o None si hay error
        """
        try:
            datos = {
                'id': self.entrada_id_busqueda.get().strip(),
                'fecha': self.entrada_fecha_act.get().strip(),
                'edad': self.entrada_edad_act.get().strip(),
                'sexo': self.entrada_sexo_act.get(),
                'bebidas_semana':
self.entrada_bebidas_semana_act.get().strip(),
                'cervezas': self.entrada_cervezas_act.get().strip(),
                'finde': self.entrada_finde_act.get().strip(),
                'destiladas': self.entrada_destiladas_act.get().strip(),
                'vinos': self.entrada_vinos_act.get().strip(),
                'perdidas_control':
self.entrada_perdidas_control_act.get() == 'Sí',
```

```
                'diversion_alcohol':
self.entrada_diversion_alcohol_act.get() == 'Sí',
                'problemas_digestivos':
self.entrada_problemas_digestivos_act.get() == 'Sí',
                'tension_alta': self.entrada_tension_alta_act.get() ==
'Sí',
                'dolor_cabeza': self.entrada_dolor_cabeza_act.get()
            }

            # Validar datos
            if not all(str(v).strip() for v in datos.values() if not
isinstance(v, bool)):
                raise ValueError("Todos los campos son obligatorios")

            return datos

        except Exception as e:
            messagebox.showerror("Error", f"Error al obtener datos:
{str(e)}")
            return None

    def _guardar_actualizacion(self):
        if messagebox.askyesno("Confirmar actualización",
                               "¿Está seguro de que desea actualizar los
datos del paciente?"):
            if hasattr(self, '_actualizar_callback'):
                datos = self.obtener_datos_formulario_actualizacion()
                if datos:
                    self._actualizar_callback(datos)
                    # Deshabilitar campos después de actualizar
                    for child in self.frame_actualizar.winfo_children():
                        child.configure(state='disabled')
                    self.boton_guardar.configure(state='disabled')

    def _configurar_validacion(self):
        """Configura la validación de campos con feedback mejorado"""
        try:
            # Configurar estilos de validación
            style = ttk.Style()
            style.configure('Error.TEntry',
                            fieldbackground='#ffd1d1',
                            bordercolor='#ff0000')
            style.configure('Warning.TEntry',
                            fieldbackground='#fff3cd',
                            bordercolor='#ffeeba')
            style.configure('Valid.TEntry',
                            fieldbackground='#d4edda',
                            bordercolor='#c3e6cb')
```

```
            # Función genérica para validar campos numéricos
            def validar_numerico(widget, min_val=0, max_val=None,
mensaje=""):
                def _validar(event):
                    try:
                        valor = widget.get().strip()
                        if not valor:
                            widget.configure(style='Warning.TEntry')
                            self._mostrar_tooltip(widget, "Campo
requerido")
                            return False

                        valor = float(valor)
                        if valor < min_val or (max_val and valor >
max_val):
                            raise ValueError(f"Valor fuera de rango
({min_val}-{max_val if max_val else 'inf'})")

                        widget.configure(style='Valid.TEntry')
                        self._ocultar_tooltip(widget)
                        return True

                    except ValueError:
                        widget.configure(style='Error.TEntry')
                        self._mostrar_tooltip(widget, mensaje)
                        return False
                return _validar

            # Función para validar fecha
            def validar_fecha(event):
                try:
                    fecha = self.entrada_fecha.get().strip()
                    if not fecha:
                        self.entrada_fecha.configure(style='Warning.TEntr
y')
                        self._mostrar_tooltip(self.entrada_fecha, "La
fecha es obligatoria")
                        return

                    datetime.strptime(fecha, '%Y-%m-%d')
                    self.entrada_fecha.configure(style='Valid.TEntry')
                    self._ocultar_tooltip(self.entrada_fecha)

                except ValueError:
                    self.entrada_fecha.configure(style='Error.TEntry')
                    self._mostrar_tooltip(self.entrada_fecha, "Formato
inválido (YYYY-MM-DD)")

            # Configurar validaciones solo si los widgets existen
```

```
            widgets_validacion = {
                'entrada_fecha': (validar_fecha, None),
                'entrada_edad': (validar_numerico, {'min_val': 0,
'max_val': 120, 'mensaje': "Edad inválida (0-120)"}),
                'entrada_alcohol': (validar_numerico, {'min_val': 0,
'mensaje': "Cantidad inválida"})
            }

            # Campos de consumo
            campos_consumo = [
                'bebidas_semana', 'cervezas', 'finde', 'destiladas',
'vinos'
            ]

            # Agregar campos de consumo a la validación
            for campo in campos_consumo:
                widgets_validacion[f'entrada_{campo}'] = (
                    validar_numerico,
                    {'min_val': 0, 'mensaje': "Cantidad inválida"}
                )

            # Aplicar validaciones
            for widget_name, (validator, params) in
widgets_validacion.items():
                if hasattr(self, widget_name) and getattr(self,
widget_name) is not None:
                    widget = getattr(self, widget_name)
                    if params is None:
                        widget.bind('<FocusOut>', validator)
                    else:
                        widget.bind('<FocusOut>', validator(widget,
**params))

        except Exception as e:
            print(f"Error al configurar validación: {str(e)}")

    def _mostrar_tooltip(self, widget, mensaje):
        """Muestra un tooltip con el mensaje de error"""
        try:
            x, y, _, _ = widget.bbox("insert")
            x += widget.winfo_rootx() + 25
            y += widget.winfo_rooty() + 20

            self.tooltip = tk.Toplevel(widget)
            self.tooltip.wm_overrideredirect(True)
            self.tooltip.wm_geometry(f"+{x}+{y}")

            label = ttk.Label(self.tooltip, text=mensaje,
                              justify='left',
```

```
                            background="#ffeeee",
                            relief='solid',
                            borderwidth=1,
                            font=("Segoe UI", 9))
            label.pack()

        except Exception as e:
            print(f"Error al mostrar tooltip: {e}")

    def _ocultar_tooltip(self, widget):
        """Oculta el tooltip si existe"""
        if hasattr(self, 'tooltip'):
            self.tooltip.destroy()


    def crear_panel_visualizacion(self):
        """Crea el panel de visualización de gráficas"""
        self.frame_grafico = ttk.LabelFrame(self.tab_visualizacion,
text="Gráficos")
        self.frame_grafico.pack(padx=10, pady=10, fill='both',
expand=True)

        # Panel de controles
        frame_controles = ttk.Frame(self.frame_grafico)
        frame_controles.pack(fill='x', padx=5, pady=5)

        # Selector de tipo de gráfica
        ttk.Label(frame_controles, text="Tipo de
Gráfica:").pack(side='left', padx=5)
        self.tipo_grafica = ttk.Combobox(
            frame_controles,
            values=[
                'Consumo por Edad',
                'Problemas de Salud',
                'Tendencia Temporal'
            ],
            state='readonly',
            width=30
        )
        self.tipo_grafica.pack(side='left', padx=5)
        self.tipo_grafica.set('Consumo por Edad')

        # Frame para contener el gráfico
        self.frame_figura = ttk.Frame(self.frame_grafico)
        self.frame_figura.pack(fill='both', expand=True, padx=5, pady=5)

        # Botón actualizar
        ttk.Button(
            frame_controles,
```

```
            text="Mostrar Gráfica",
            command=self._on_visualizar
        ).pack(side='right', padx=5)

    def crear_panel_estadisticas(self):
        self.frame_estadisticas = ttk.LabelFrame(self.tab_estadisticas,
text="Estadísticas")
        self.frame_estadisticas.pack(padx=10, pady=10, fill='both',
expand=True)

        ttk.Button(self.frame_estadisticas, text="Actualizar
Estadísticas",
                   command=self._on_estadisticas).pack(pady=5)


    def registrar_callback(self, callback):
        self._registrar_callback = callback

    def visualizar_callback(self, callback):
        self._visualizar_callback = callback

    def estadisticas_callback(self, callback):
        self._estadisticas_callback = callback

    def exportar_callback(self, callback):
        self._exportar_callback = callback

    def eliminar_callback(self, callback):
        """
        Registra el callback para eliminar pacientes
        Args:
            callback: Función a llamar cuando se quiera eliminar un
paciente
        """
        self._eliminar_callback = callback

    def _on_registrar(self):
        """Maneja el evento de registro de paciente"""
        if self._registrar_callback:
            datos = self.obtener_datos_formulario()
            if datos:  # Verifica que se obtuvieron datos válidos
                if self._registrar_callback(datos):  # Llama al callback
con los datos
                    # Si el registro fue exitoso, limpiar el formulario
                    self.limpiar_formulario()


    def _on_visualizar(self):
        if self._visualizar_callback:
```

```
            self._visualizar_callback()

    def _on_estadisticas(self):
        if self._estadisticas_callback:
            self._estadisticas_callback()

    def obtener_datos_formulario(self):
        """
        Obtiene y valida los datos del formulario.
        Returns:
            dict: Diccionario con los datos validados o None si hay error
        """
        try:
            # Verificar que los widgets existan
            widgets_requeridos = [
                'entrada_fecha',
                'entrada_edad',
                'entrada_sexo',
                'entrada_bebidas_semana',
                'entrada_cervezas',
                'entrada_finde',
                'entrada_destiladas',
                'entrada_vinos',
                'entrada_perdidas_control',
                'entrada_diversion_alcohol',
                'entrada_problemas_digestivos',
                'entrada_tension_alta',
                'entrada_dolor_cabeza'
            ]

            # Verificar existencia de widgets
            for widget in widgets_requeridos:
                if not hasattr(self, widget) or getattr(self, widget) is
None:
                    raise ValueError(f"Campo {widget} no inicializado")

            # Obtener y validar datos
            datos = {}

            # Campos de texto con strip()
            campos_texto = {
                'fecha': self.entrada_fecha,
                'edad': self.entrada_edad,
                'sexo': self.entrada_sexo,
                'bebidas_semana': self.entrada_bebidas_semana,
                'cervezas': self.entrada_cervezas,
                'finde': self.entrada_finde,
                'destiladas': self.entrada_destiladas,
                'vinos': self.entrada_vinos
```

```
            }

            # Campos booleanos/selección
            campos_seleccion = {
                'perdidas_control': self.entrada_perdidas_control,
                'diversion_alcohol': self.entrada_diversion_alcohol,
                'problemas_digestivos':
self.entrada_problemas_digestivos,
                'tension_alta': self.entrada_tension_alta,
                'dolor_cabeza': self.entrada_dolor_cabeza
            }

            # Procesar campos de texto
            for campo, widget in campos_texto.items():
                valor = widget.get().strip()
                if not valor:
                    self._mostrar_error_campo(campo)
                    raise ValueError(f"El campo {campo} es obligatorio")
                datos[campo] = valor

            # Procesar campos de selección
            for campo, widget in campos_seleccion.items():
                valor = widget.get()
                if not valor:
                    self._mostrar_error_campo(campo)
                    raise ValueError(f"El campo {campo} es obligatorio")
                datos[campo] = valor

            # Validaciones específicas
            self._validar_fecha(datos['fecha'])
            self._validar_edad(datos['edad'])
            self._validar_campos_numericos(datos)

            return datos

        except ValueError as e:
            messagebox.showerror("Error de Validación", str(e))
            return None
        except Exception as e:
            messagebox.showerror("Error", f"Error al procesar el
formulario: {str(e)}")
            return None

    def _validar_fecha(self, fecha):
        """Valida el formato de la fecha"""
        try:
            datetime.strptime(fecha, '%Y-%m-%d')
        except ValueError:
            self._mostrar_error_campo('fecha')
```

```
            raise ValueError("Formato de fecha inválido (YYYY-MM-DD)")

    def _validar_edad(self, edad):
        """Valida que la edad esté en el rango correcto"""
        try:
            edad = int(edad)
            if not (0 <= edad <= 120):
                raise ValueError
        except ValueError:
            self._mostrar_error_campo('edad')
            raise ValueError("Edad inválida (0-120 años)")

    def _validar_campos_numericos(self, datos):
        """Valida los campos numéricos del formulario"""
        campos_numericos = ['bebidas_semana', 'cervezas', 'finde',
'destiladas', 'vinos']
        for campo in campos_numericos:
            try:
                valor = float(datos[campo])
                if valor < 0:
                    raise ValueError
            except ValueError:
                self._mostrar_error_campo(campo)
                raise ValueError(f"Valor inválido en {campo}")

    def _mostrar_error_campo(self, campo):
        """Resalta visualmente el campo con error"""
        if hasattr(self, f'entrada_{campo}'):
            widget = getattr(self, f'entrada_{campo}')
            widget.configure(style='Error.TEntry')
            widget.focus()

    def validar_datos(self, datos):
        if not all(datos.values()):
            messagebox.showwarning("Error", "Todos los campos son
requeridos")
            return False
        try:
            datetime.strptime(datos['fecha'], '%Y-%m-%d')
            float(datos['alcohol'])
            int(datos['edad'])
            if not '/' in datos['presion']:
                raise ValueError("Formato de presión inválido")
            return True
        except ValueError as e:
            messagebox.showerror("Error", f"Error de validación:
{str(e)}")
            return False
```

```
    def limpiar_formulario(self):
        """Limpia todos los campos del formulario"""
        campos_a_limpiar = [
            'entrada_fecha',
            'entrada_edad',
            'entrada_bebidas_semana',
            'entrada_cervezas',
            'entrada_finde',
            'entrada_destiladas',
            'entrada_vinos'
        ]

        # Limpiar campos de entrada de texto
        for campo in campos_a_limpiar:
            if hasattr(self, campo):
                widget = getattr(self, campo)
                if widget:
                    widget.delete(0, tk.END)

        # Resetear comboboxes
        campos_combo = [
            'entrada_sexo',
            'entrada_perdidas_control',
            'entrada_diversion_alcohol',
            'entrada_problemas_digestivos',
            'entrada_tension_alta',
            'entrada_dolor_cabeza'
        ]

        for campo in campos_combo:
            if hasattr(self, campo):
                widget = getattr(self, campo)
                if widget:
                    widget.set(widget.cget('values')[0])

    def actualizar_grafico(self, figura):
        """Actualiza el gráfico en la interfaz"""
        try:
            # Limpiar el frame de la figura
            for widget in self.frame_figura.winfo_children():
                widget.destroy()

            if figura is None:
                messagebox.showinfo("Info", "No hay datos para mostrar")
                return

            # Crear canvas para la figura
            canvas = FigureCanvasTkAgg(figura, self.frame_figura)
            canvas.draw()
```

```
            widget = canvas.get_tk_widget()
            widget.pack(fill='both', expand=True)

        except Exception as e:
            print(f"Error al actualizar gráfico: {str(e)}")
            messagebox.showerror("Error", f"Error al actualizar gráfico:
{str(e)}")

    def actualizar_registros_recientes(self, registros):
        """
        Actualiza los registros recientes en el Treeview
        :param registros: DataFrame con los registros recientes
        """
        try:
            # Limpiar registros existentes
            for item in self.tree_registros_recientes.get_children():
                self.tree_registros_recientes.delete(item)

            # Insertar nuevos registros
            for _, registro in registros.iterrows():
                valores = (
                    registro['idEncuesta'],
                    registro['Sexo'],
                    registro['edad'],
                    f"{registro['BebidasSemana']:.1f}",
                    f"{registro['BebidasFinSemana']:.1f}",
                    f"{registro['BebidasDestiladasSemana']:.1f}",
                    f"{registro['VinosSemana']:.1f}",
                    f"{registro['CervezasSemana']:.1f}"
                )
                self.tree_registros_recientes.insert('', 'end',
values=valores)
        except Exception as e:
            print(f"Error al actualizar registros recientes: {e}")

    def actualizar_estadisticas(self, stats, alto_consumo):
        try:
            # Limpiar widgets anteriores
            for widget in self.frame_estadisticas.winfo_children():
                widget.destroy()

            # Panel principal con pestañas
            notebook = ttk.Notebook(self.frame_estadisticas)
            notebook.pack(fill='both', expand=True, padx=10, pady=5)

            # Solo mantenemos las dos pestañas principales
            tab_resumen = ttk.Frame(notebook)
            tab_alto_riesgo = ttk.Frame(notebook)
```

```
            notebook.add(tab_resumen, text='Resumen General')
            notebook.add(tab_alto_riesgo, text='Alto Riesgo')

            # Crear contenido de las pestañas
            self._crear_resumen_general(tab_resumen, stats)
            self._crear_panel_alto_riesgo(tab_alto_riesgo, alto_consumo)

        except Exception as e:
            messagebox.showerror("Error", f"Error al actualizar
estadísticas: {str(e)}")

    def eliminar_paciente(self, id_paciente):
        """
        Elimina un paciente utilizando el callback registrado
        Args:
            id_paciente: ID del paciente a eliminar
        """
        try:
            if self._eliminar_callback:
                if self._eliminar_callback(id_paciente):
                    messagebox.showinfo("Éxito", "Paciente eliminado
correctamente")
                else:
                    messagebox.showerror("Error", "No se pudo eliminar el
paciente")
        except Exception as e:
            messagebox.showerror("Error", f"Error al eliminar paciente:
{str(e)}")

    def actualizar_listado_pacientes(self, pacientes):
        try:
            # Limpiar listado existente
            for item in self.tree_pacientes.get_children():
                self.tree_pacientes.delete(item)

            # Insertar nuevos datos
            for _, paciente in pacientes.iterrows():
                self.tree_pacientes.insert('', 'end', values=(
                    paciente['idEncuesta'],
                    paciente['Fecha'].strftime('%Y-%m-%d') if 'Fecha' in
paciente else '',
                    paciente['Sexo'],
                    paciente['edad'],
                    f"{paciente['BebidasSemana']:.1f}",
                    f"{paciente['BebidasFinSemana']:.1f}",
                    f"{paciente['CervezasSemana']:.1f}",
                    f"{paciente['BebidasDestiladasSemana']:.1f}",
                    f"{paciente['VinosSemana']:.1f}"
                ))
```

```
        except Exception as e:
            messagebox.showerror("Error", f"Error al actualizar listado:
{str(e)}")

    def _crear_resumen_general(self, tab, stats):
        # Frame principal con grid layout
        main_frame = ttk.Frame(tab)
        main_frame.pack(fill='both', expand=True, padx=10, pady=5)

        # Frame izquierdo para estadísticas existentes
        frame_stats = ttk.LabelFrame(main_frame, text="Estadísticas
Generales")
        frame_stats.pack(side='left', padx=5, pady=5, fill='both',
expand=True)

        # Frame derecho para registros recientes
        frame_recientes = ttk.LabelFrame(main_frame, text="Registros
Recientes")
        frame_recientes.pack(side='right', padx=5, pady=5, fill='both',
expand=True)

        # Estadísticas por género
        for sexo in ['Hombre', 'Mujer']:
            datos_sexo = stats[stats['Sexo'] == sexo]
            frame_genero = ttk.LabelFrame(frame_stats,
text=f"Estadísticas {sexo}")
            frame_genero.pack(padx=5, pady=5, fill='x')

            estadisticas = {
                'Promedio Bebidas/Semana':
datos_sexo['BebidasSemana'].mean(),
                'Promedio Cervezas': datos_sexo['CervezasSemana'].mean(),
                'Promedio Fin de Semana':
datos_sexo['BebidasFinSemana'].mean(),
                'Promedio Destiladas':
datos_sexo['BebidasDestiladasSemana'].mean(),
                'Promedio Vinos': datos_sexo['VinosSemana'].mean(),
                'Total Personas': len(datos_sexo)
            }

            for i, (label, value) in enumerate(estadisticas.items()):
                ttk.Label(frame_genero, text=f"{label}:",
style="Bold.TLabel").grid(
                    row=i, column=0, padx=5, pady=2, sticky='e')
                ttk.Label(frame_genero, text=f"{value:.2f}").grid(
                    row=i, column=1, padx=5, pady=2, sticky='w')

        # Crear Treeview para registros recientes
```

```
        columns = ('ID', 'Sexo', 'Edad', 'Bebidas', 'Fin Sem', 'Dest',
'Vinos', 'Cerv')
        self.tree_registros_recientes = ttk.Treeview(frame_recientes,
columns=columns, show='headings', height=8)

        # Configurar columnas
        for col in columns:
            self.tree_registros_recientes.heading(col, text=col)
            self.tree_registros_recientes.column(col, width=60)

        # Añadir scrollbar
        scrollbar = ttk.Scrollbar(frame_recientes, orient='vertical',
command=self.tree_registros_recientes.yview)
        self.tree_registros_recientes.configure(yscrollcommand=scrollbar.
set)

        # Empaquetar Treeview y scrollbar
        self.tree_registros_recientes.pack(side='left', fill='both',
expand=True)
        scrollbar.pack(side='right', fill='y')

    def _crear_panel_alto_riesgo(self, tab, alto_consumo):
        frame = ttk.LabelFrame(tab, text="Casos de Alto Consumo")
        frame.pack(padx=10, pady=5, fill='both', expand=True)

        # Crear Treeview con más detalles
        columns = ('ID', 'Edad', 'Sexo', 'Bebidas', 'Cerveza', 'Finde',
'Destiladas', 'Vinos')
        tree = ttk.Treeview(frame, columns=columns, show='headings',
height=10)

        # Configurar columnas
        for col in columns:
            tree.heading(col, text=col)
            tree.column(col, width=80, anchor='center')

        # Insertar datos
        for _, row in alto_consumo.iterrows():
            tree.insert('', 'end', values=(
                row['idEncuesta'],
                row['edad'],
                row['Sexo'],
                f"{row['BebidasSemana']:.1f}",
                f"{row['CervezasSemana']:.1f}",
                f"{row['BebidasFinSemana']:.1f}",
                f"{row['BebidasDestiladasSemana']:.1f}",
                f"{row['VinosSemana']:.1f}"
            ))
```

```
        # Agregar scrollbars
        scrolly = ttk.Scrollbar(frame, orient='vertical',
command=tree.yview)
        scrollx = ttk.Scrollbar(frame, orient='horizontal',
command=tree.xview)
        tree.configure(yscrollcommand=scrolly.set,
xscrollcommand=scrollx.set)

        # Eventos del árbol
        tree.bind('<<TreeviewSelect>>', lambda e:
self._on_select_paciente(e, tree))

        # Empaquetar componentes
        tree.pack(side='left', fill='both', expand=True)
        scrolly.pack(side='right', fill='y')
        scrollx.pack(side='bottom', fill='x')

    def _on_select_paciente(self, event, tree):
        """Maneja la selección de un paciente en el TreeView"""
        try:
            if tree.selection():
                item = tree.selection()[0]
                valores = tree.item(item)['values']
                print(f"Paciente seleccionado: ID={valores[0]}")
        except Exception as e:
            print(f"Error al seleccionar paciente: {e}")


    def _on_doble_click_paciente(self, event):
        """
        Maneja el evento de doble clic para eliminar un paciente
        """
        try:
            if self.tree_pacientes.selection():
                item = self.tree_pacientes.selection()[0]
                paciente_id = self.tree_pacientes.item(item)['values'][0]

                if messagebox.askyesno("Confirmar eliminación",
                                       "¿Está seguro de que desea eliminar
este paciente?"):
                    if self._eliminar_callback:
                        self._eliminar_callback(paciente_id)
        except Exception as e:
            messagebox.showerror("Error", f"Error al intentar eliminar:
{str(e)}")

    def mostrar_mensaje(self, titulo, mensaje):
        messagebox.showinfo(titulo, mensaje)
```

main.py

```
import tkinter as tk
from tkinter import ttk, messagebox
from bdd.conexion import ConexionBD
from bdd.consultas import ConsultasEncuesta
from graficas.visualizargraficas import VisualizadorGraficas
from interfaz.interfazusuario import InterfazSaludAlcohol
from mysql.connector import Error as DatabaseError
import pandas as pd
import sys
from datetime import datetime

class SistemaMonitoreoSalud:
    def __init__(self):
        try:
            self.root = tk.Tk()
            self.root.title("Sistema de Monitoreo de Salud y Alcohol")
            self.root.geometry("1200x800")
            self.root.minsize(1000, 600)

            # Configurar tema y estilos
            self.configurar_estilos()

            # Protocolo de cierre
            self.root.protocol("WM_DELETE_WINDOW",
self.cerrar_aplicacion)

            # Inicialización de componentes
            self.inicializar_componentes()

        except Exception as e:
            messagebox.showerror("Error de Inicialización", str(e))
            sys.exit(1)

    def configurar_estilos(self):
        style = ttk.Style()

        # Colores principales
        self.COLOR_PRIMARY = "#2196F3"
        self.COLOR_SECONDARY = "#64B5F6"
        self.COLOR_BACKGROUND = "#F5F5F5"
        self.COLOR_SUCCESS = "#4CAF50"
        self.COLOR_WARNING = "#FFC107"
        self.COLOR_ERROR = "#F44336"
        self.COLOR_TEXT = "#212121"
```

```
        self.COLOR_ACCENT = "#1976D2"

        # Configuración general
        self.root.configure(bg=self.COLOR_BACKGROUND)

        # Estilos personalizados
        style.configure(".",
            font=("Segoe UI", 10),
            background=self.COLOR_BACKGROUND,
            foreground=self.COLOR_TEXT
        )

        style.configure("Title.TLabel",
            font=("Segoe UI", 24, "bold"),
            foreground=self.COLOR_PRIMARY,
            padding=(0, 10)
        )

        style.configure("Custom.TLabelframe",
            background=self.COLOR_BACKGROUND,
            padding=15,
            relief="solid"
        )

        style.configure("Primary.TButton",
            font=("Segoe UI", 10, "bold"),
            background=self.COLOR_PRIMARY,
            foreground="black",
            padding=(20, 10)
        )

        style.configure("Secondary.TButton",
            font=("Segoe UI", 10),
            background=self.COLOR_SECONDARY,
            foreground="black",
            padding=(15, 8)
        )

        style.configure("Custom.TEntry",
            padding=8,
            fieldbackground="white"
        )

    def inicializar_componentes(self):
        try:
            print("Inicializando visualizador de gráficas...")
            self.visualizador = VisualizadorGraficas()

            print("Inicializando conexión BD...")
```

```
            self.conexion = ConexionBD()
            if not hasattr(self.conexion, 'conexion') or not
self.conexion.conexion.is_connected():
                raise DatabaseError("No se pudo establecer la conexión
con la base de datos")

            print("Inicializando consultas...")
            self.consultas = ConsultasEncuesta(self.conexion)

            print("Inicializando interfaz...")
            self.interfaz = InterfazSaludAlcohol(self.root)

            # Conectar callbacks
            self.interfaz.registrar_callback(self.registrar_paciente)
            self.interfaz.visualizar_callback(self.mostrar_graficas)
            self.interfaz.estadisticas_callback(self.mostrar_estadisticas
)
            self.interfaz.exportar_callback(self.exportar_a_excel)
            self.interfaz.eliminar_callback(self.eliminar_paciente)
            self.interfaz.registrar_buscar_callback(self.buscar_paciente)
            self.interfaz.registrar_actualizar_callback(self.actualizar_p
aciente)
            self.cargar_datos_iniciales()

            # Cargar listado de pacientes
            pacientes = self.consultas.obtener_listado_pacientes()
            self.interfaz.actualizar_listado_pacientes(pacientes)

        except Exception as e:
            raise Exception(f"Error al inicializar componentes:
{str(e)}")

    def cargar_datos_iniciales(self):
        """Carga los datos iniciales y actualiza la interfaz"""
        try:
            print("Cargando datos iniciales...")
            stats = self.consultas.obtener_estadisticas_consumo()
            alto_consumo = self.consultas.filtrar_alto_consumo()

            if stats is not None and not stats.empty:
                self.interfaz.actualizar_estadisticas(stats,
alto_consumo)
                self.mostrar_graficas()
            else:
                print("No hay datos iniciales disponibles")

        except Exception as e:
            print(f"Error al cargar datos iniciales: {str(e)}")
```

```
    def registrar_paciente(self, datos_paciente):
        """Registra un nuevo paciente en la base de datos"""
        try:
            # Insertar nuevo registro
            self.consultas.insertar_encuesta(datos_paciente)
            messagebox.showinfo("Éxito", "Paciente registrado
correctamente")

            # Actualizar estadísticas y gráficas
            self.mostrar_estadisticas()
            self.mostrar_graficas()

            return True

        except Exception as e:
            messagebox.showerror("Error", f"Error al registrar paciente:
{str(e)}")
            return False

    def buscar_paciente(self, id_paciente):
        """Busca un paciente por su ID y muestra sus datos"""
        try:
            datos = self.consultas.obtener_paciente_por_id(id_paciente)
            if datos:
                self.interfaz.mostrar_datos_paciente(datos)
                messagebox.showinfo("Éxito", "Paciente encontrado. Puede
proceder a modificar los datos.")
            else:
                messagebox.showwarning("No encontrado", "No se encontró
ningún paciente con ese ID")
        except Exception as e:
            messagebox.showerror("Error", f"Error al buscar paciente:
{str(e)}")

    def actualizar_paciente(self, datos):
        """Actualiza los datos de un paciente existente"""
        try:
            self.consultas.actualizar_paciente(datos)
            messagebox.showinfo("Éxito", "Paciente actualizado
correctamente")
        except Exception as e:
            messagebox.showerror("Error", f"Error al actualizar paciente:
{str(e)}")

    def eliminar_paciente(self, id_paciente):
        """
        Elimina un paciente y actualiza el listado
        Args:
            id_paciente: ID del paciente a eliminar
```

```
        """
        try:
            self.consultas.eliminar_paciente(id_paciente)
            messagebox.showinfo("Éxito", "Paciente eliminado
correctamente")
            # Actualizar el listado
            pacientes = self.consultas.obtener_listado_pacientes()
            self.interfaz.actualizar_listado_pacientes(pacientes)
        except Exception as e:
            messagebox.showerror("Error", f"No se pudo eliminar el
paciente: {str(e)}")

    def mostrar_graficas(self):
        """Muestra las gráficas basadas en el tipo seleccionado"""
        try:
            tipo_grafica = self.interfaz.tipo_grafica.get()
            datos = self._obtener_datos_grafica(tipo_grafica)

            if datos is not None and not datos.empty:
                fig = self._crear_grafica(tipo_grafica, datos)
                if fig:
                    self.interfaz.actualizar_grafico(fig)
                else:
                    messagebox.showinfo("Info", "No se pudo crear la
gráfica")
            else:
                messagebox.showinfo("Info", "No hay datos disponibles
para graficar")

        except Exception as e:
            print(f"Error en mostrar_graficas: {str(e)}")
            messagebox.showerror("Error", f"Error al mostrar gráfica:
{str(e)}")

    def mostrar_estadisticas(self):
        """Muestra las estadísticas de consumo y registros recientes"""
        try:
            # Obtener estadísticas básicas
            stats = self.consultas.obtener_estadisticas_consumo()
            if stats is None or stats.empty:
                raise ValueError("No hay datos disponibles")

            # Obtener registros recientes
            registros_recientes =
self.consultas.obtener_registros_recientes()

            # Obtener datos de alto consumo
            alto_consumo = self.consultas.filtrar_alto_consumo()
```

```
            # Actualizar la interfaz con todos los datos
            self.interfaz.actualizar_estadisticas(stats, alto_consumo)
            if not registros_recientes.empty:
                self.interfaz.actualizar_registros_recientes(registros_re
cientes)

        except Exception as e:
            messagebox.showerror("Error", f"Error al mostrar
estadísticas: {str(e)}")

    def exportar_a_excel(self, tipo_exportacion):
        """Exporta los datos a un archivo Excel"""
        try:
            # Obtener datos actuales
            datos = self.consultas.obtener_estadisticas_consumo()
            if datos is None or datos.empty:
                messagebox.showwarning("Advertencia", "No hay datos para
exportar")
                return

            # Pedir al usuario la ubicación para guardar el archivo
            from tkinter import filedialog
            filename = filedialog.asksaveasfilename(
                defaultextension=".xlsx",
                filetypes=[("Excel files", "*.xlsx"), ("All files",
"*.*")],
                title="Guardar como Excel"
            )

            if filename:
                # Exportar utilizando el método de consultas
                if self.consultas.exportar_a_excel(datos, filename):
                    messagebox.showinfo("Éxito", "Datos exportados
correctamente")
                else:
                    messagebox.showerror("Error", "No se pudo exportar el
archivo")

        except Exception as e:
            messagebox.showerror("Error", f"Error al exportar: {str(e)}")
            print(f"Error detallado en exportación: {str(e)}")

    def _obtener_datos_grafica(self, tipo_grafica):
        """Obtiene los datos necesarios para la gráfica seleccionada"""
        if tipo_grafica == 'Consumo por Edad':
            return self.consultas.obtener_estadisticas_consumo()
        elif tipo_grafica == 'Problemas de Salud':
            return self.consultas.filtrar_problemas_salud()
        elif tipo_grafica == 'Tendencia Temporal':
```

```
            return self.consultas.ordenar_por_campo('idEncuesta')
        return None

    def _crear_grafica(self, tipo_grafica, datos):
        """Crea la gráfica basada en el tipo y los datos
proporcionados"""
        if tipo_grafica == 'Consumo por Edad':
            return self.visualizador.crear_grafica_consumo_edad(datos)
        elif tipo_grafica == 'Problemas de Salud':
            return self.visualizador.crear_grafica_problemas_salud(datos)
        elif tipo_grafica == 'Tendencia Temporal':
            return
self.visualizador.crear_grafica_tendencia_temporal(datos)
        return None

    def cerrar_aplicacion(self):
        """Cierra la aplicación y la conexión a la base de datos"""
        if messagebox.askokcancel("Salir", "¿Desea salir de la
aplicación?"):
            if hasattr(self, 'conexion'):
                self.conexion.cerrar_conexion()
            self.root.destroy()

def main():
    try:
        print("=== Iniciando Sistema de Monitoreo de Salud ===")

        # Verificar conexión BD primero
        print("Verificando conexión a base de datos...")
        conexion = ConexionBD()
        if not conexion.conexion.is_connected():
            raise DatabaseError("No se pudo establecer la conexión con la
base de datos")
        conexion.cerrar_conexion()  # Cerrar conexión de prueba
        print("Conexión a base de datos establecida correctamente")

        # Iniciar aplicación
        print("Iniciando aplicación...")
        app = SistemaMonitoreoSalud()
        print("Sistema inicializado correctamente")

        # Ejecutar el bucle principal
        app.root.mainloop()

    except DatabaseError as e:
        messagebox.showerror("Error de Base de Datos", str(e))
        print(f"Error de base de datos: {str(e)}")
        sys.exit(1)
    except Exception as e:
```

```
        messagebox.showerror("Error Fatal", str(e))
        print(f"Error fatal: {str(e)}")
        sys.exit(1)
    finally:
        print("=== Finalizando Sistema de Monitoreo de Salud ===")

if __name__ == "__main__":
    main()
```

requirements.txt

```
# Base de Python
python>=3.8

# Interfaz gráfica
tkinter>=8.6

# Base de datos
mysql-connector-python>=8.0.26

# Análisis de datos y visualización
pandas>=1.3.0
matplotlib>=3.4.3
seaborn>=0.11.2

# Exportación a Excel
openpyxl>=3.0.7

# Formateo de fechas
python-dateutil>=2.8.2
```