**“UNIVERSIDAD NACIONAL DEL ALTIPLANO – PUNO”**

**FACULTAD DE MECANICA ELECTRICA, ELECTRONICA Y SISTEMAS**

**ESCUELA PROFESIONAL DE INGENIERIA DE SISTEMAS**

|  |
| --- |
|  |
|  |  |

|  |
| --- |
|  |
|  |  |

**CURSO:**

PROGRAMACION ORIENTADA A OBJETOS II

**TEMA:**

ANALISIS DE DATOS CON PYTHON

**DOCENTE:**

ING. ZANABRIA GALVEZ ALDO HERNAN

**INTEGRANTES:**

EDUARDO PAYE BRANDON

CRISTOFER DEX QUISPE PEREZ

PERCY ALVARO QUISPE PAREDES

**PUNO – PERU**

**2024**

**ANALSIS DE DATOS DE MATRICULAS 2026-1\_2024-I**

**Descripción General**

El proyecto MatriculasApp es una aplicación de interfaz gráfica desarrollada en Python utilizando la biblioteca Tkinter. La aplicación está diseñada para gestionar y visualizar datos de matrículas correspondientes al período 2016-2024. Permite a los usuarios cargar datos desde un archivo CSV, realizar filtros específicos, generar gráficos y exportar la información procesada.

**OBJETIVOS**

El objetivo general de este proyecto se centra en la creación de una aplicación que facilite el manejo de grandes volúmenes de datos relacionados con matrículas educativas, mejorando la accesibilidad y comprensión de la información para una amplia gama de usuarios. Esta meta se desglosa en varios aspectos clave que incluyen la interactividad, la facilidad de uso, la capacidad de análisis y la visualización efectiva de datos.

Así mismo desarrollar una aplicación interactiva y fácil de usar que permita la manipulación, análisis y visualización de datos de matrículas educativas, proporcionando a los usuarios herramientas avanzadas para explorar y comprender mejor la información.

**Bibliotecas Utilizadas(**Desarrollo: código fuente)

tkinter: Utilizada para la creación de la interfaz gráfica. Tkinter es una biblioteca estándar de Python para la creación de interfaces de usuario, ofreciendo una forma sencilla y eficaz de crear GUIs.

pandas: Empleada para la manipulación y análisis de datos. Pandas es una biblioteca poderosa que proporciona estructuras de datos flexibles y eficientes para manipular datos etiquetados/relacionados.

matplotlib: Usada para la generación de gráficos. Matplotlib es una biblioteca de trazado 2D en Python que produce figuras de calidad de publicación en una variedad de formatos y entornos interactivos.

seaborn: Utilizada para la creación de mapas de calor y gráficos avanzados. Seaborn es una biblioteca de visualización de datos basada en Matplotlib que proporciona una interfaz de alto nivel para dibujar gráficos estadísticos atractivos e informativos.

matplotlib.backends.backend\_pdf: Utilizada para guardar los gráficos en formato PDF. Esta extensión de Matplotlib permite la creación de archivos PDF que pueden contener múltiples páginas de gráficos.

**Estructura de la Aplicación**

Clase Principal: MatriculasApp

Método \_\_init\_\_

Este método inicializa la ventana principal y configura todos los elementos de la interfaz, incluidos los botones, menús desplegables y el Treeview para mostrar datos.

**Métodos para Cargar y Mostrar Datos**

file\_open(): Permite cargar datos desde un archivo CSV y almacenarlos en un DataFrame de pandas. Utiliza un cuadro de diálogo para seleccionar el archivo y maneja errores si el archivo no puede ser leído.

display\_data(): Muestra los datos cargados en un widget Treeview, que es una tabla con filas y columnas que permite a los usuarios ver y explorar los datos fácilmente.

**Métodos para Generar Gráficos**

graficar\_barras(): Genera gráficos de barras basados en la columna seleccionada por el usuario. Este método obtiene la columna seleccionada del menú desplegable y utiliza Matplotlib para crear el gráfico.

graficar\_heatmap(): Crea un mapa de calor de las correlaciones entre las columnas numéricas. Utiliza Seaborn para generar un mapa de calor que visualiza las correlaciones, facilitando la identificación de relaciones entre variables.

**Métodos para Filtrar Datos**

filtrar\_por\_facultad(): Filtra los datos según la facultad especificada por el usuario.

filtrar\_por\_especialidad(): Filtra los datos según la especialidad especificada por el usuario.

filtrar\_por\_nacimiento\_depa(): Filtra los datos según el departamento de nacimiento especificado por el usuario.

filtrar\_por\_colegio\_dist(): Filtra los datos según el colegio distrital especificado por el usuario.

filtrar\_por\_anio(): Filtra los datos según el año especificado por el usuario.

filtrar\_por\_anio\_nacimiento(): Filtra los datos según el año de nacimiento especificado por el usuario.

Estos métodos permiten a los usuarios explorar los datos de manera interactiva y focalizada, aplicando múltiples filtros según sus necesidades.

**Método para Buscar Datos**

buscar\_por\_idhash(): Busca y muestra datos basados en el IDHASH proporcionado por el usuario. Este método es útil para encontrar rápidamente registros específicos en el conjunto de datos.

**Método para Mostrar Estadísticas**

matricula\_stats(): Muestra estadísticas detalladas sobre las matrículas, desglosadas por año, facultad, especialidad y ciclo. Utiliza ventanas adicionales de Tkinter para mostrar estas estadísticas de manera organizada.

**Métodos para Exportar y Guardar Datos**

guardar\_como\_pdf(): Guarda los gráficos generados en un archivo PDF. Este método utiliza matplotlib.backends.backend\_pdf.PdfPages para crear un archivo PDF que puede contener múltiples gráficos.

exportar\_datos(): Exporta los datos filtrados a un archivo CSV. Utiliza un cuadro de diálogo para guardar archivos, permitiendo a los usuarios guardar los datos en su formato preferido.

**Métodos Adicionales**

restablecer\_base\_datos(): Restablece los datos originales cargados desde el archivo CSV. Esto permite a los usuarios volver al estado inicial de los datos después de aplicar múltiples filtros.

salir(): Cierra la aplicación de manera segura.

**Funcionalidades Principales**

Cargar y Mostrar Datos

Los usuarios pueden cargar datos de matrículas desde un archivo CSV y visualizarlos en un formato tabular dentro de la aplicación. Esto facilita la exploración inicial y la comprensión de los datos.

**Generación de Gráficos**

La aplicación permite generar gráficos de barras y mapas de calor, proporcionando una forma visual y fácil de entender la distribución y las correlaciones en los datos.

**Filtros Personalizados**

Los usuarios pueden aplicar filtros específicos para visualizar datos según facultad, especialidad, departamento de nacimiento, colegio distrital, año y año de nacimiento. Estos filtros ayudan a enfocar el análisis en subconjuntos específicos de datos.

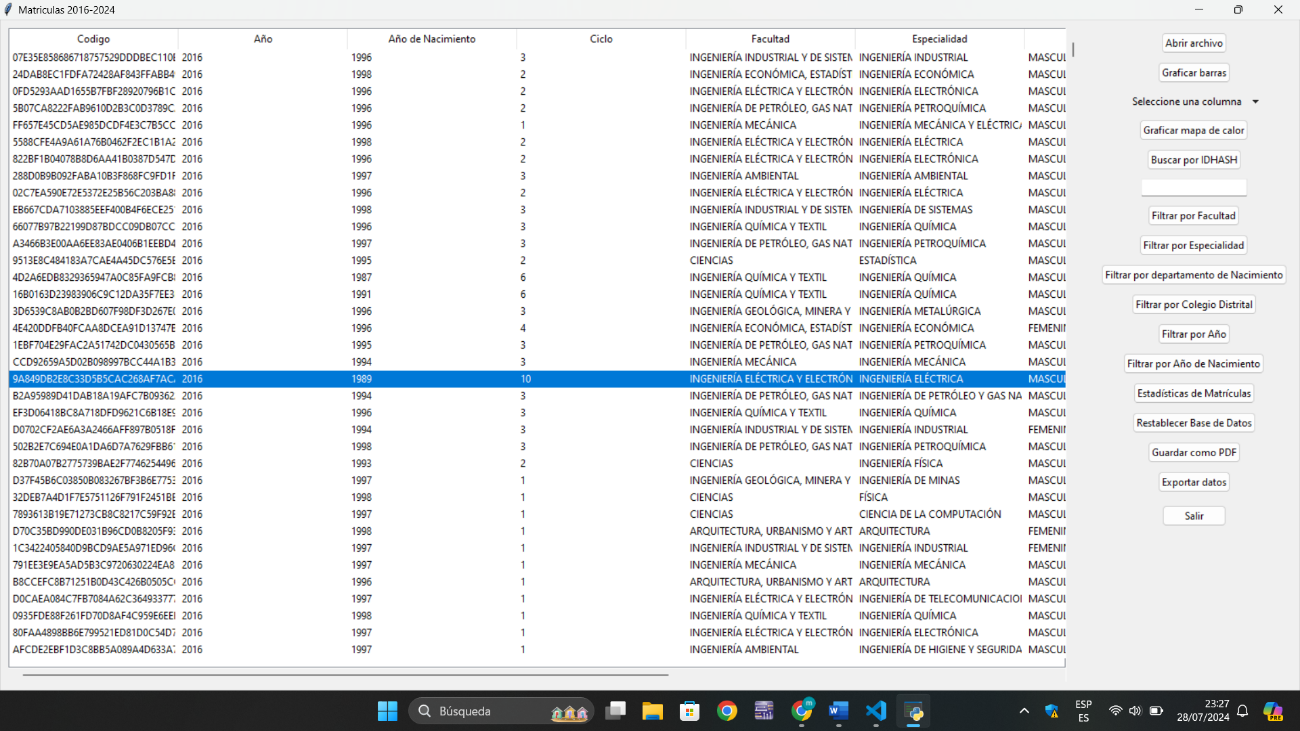
**Estadísticas Detalladas**

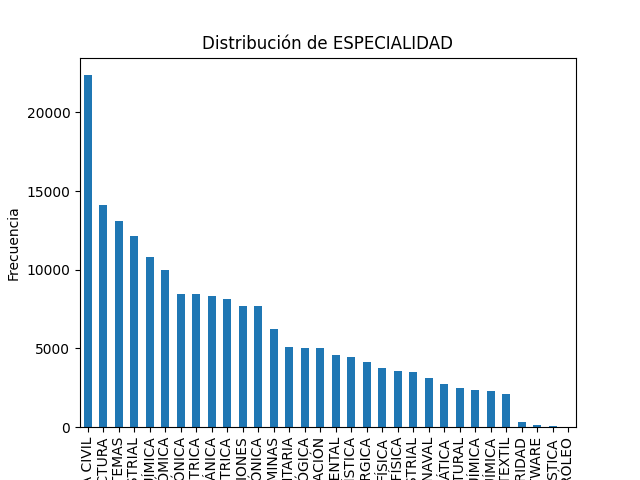
La aplicación ofrece la capacidad de obtener estadísticas detalladas sobre las matrículas, mostrando la distribución de estudiantes por año, facultad, especialidad y ciclo. Esto es útil para análisis más profundos y presentaciones.

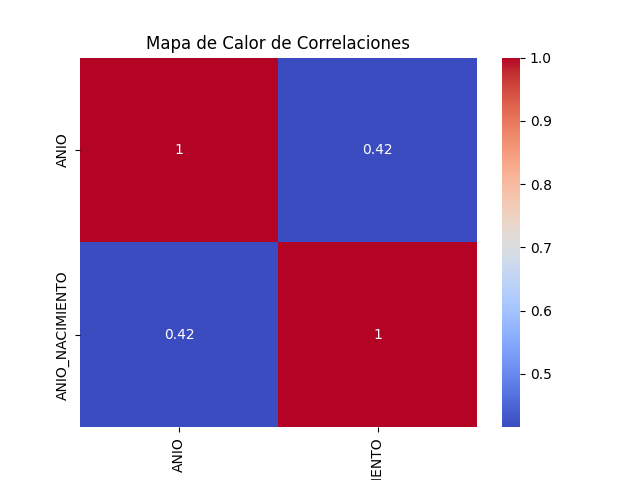
**Exportación y Guardado**

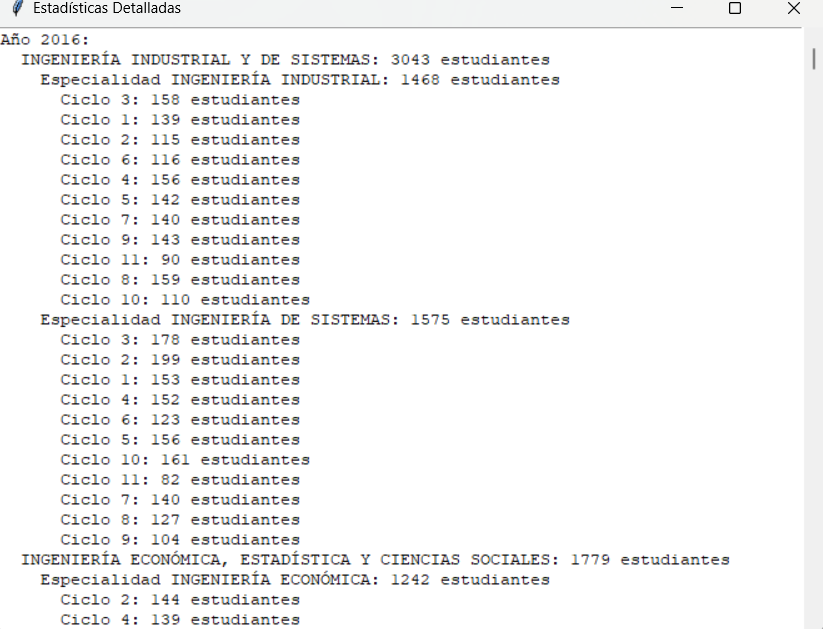
La aplicación permite exportar los datos filtrados a un archivo CSV y guardar los gráficos generados en un archivo PDF. Esto facilita el almacenamiento y la compartición de los resultados del análisis.

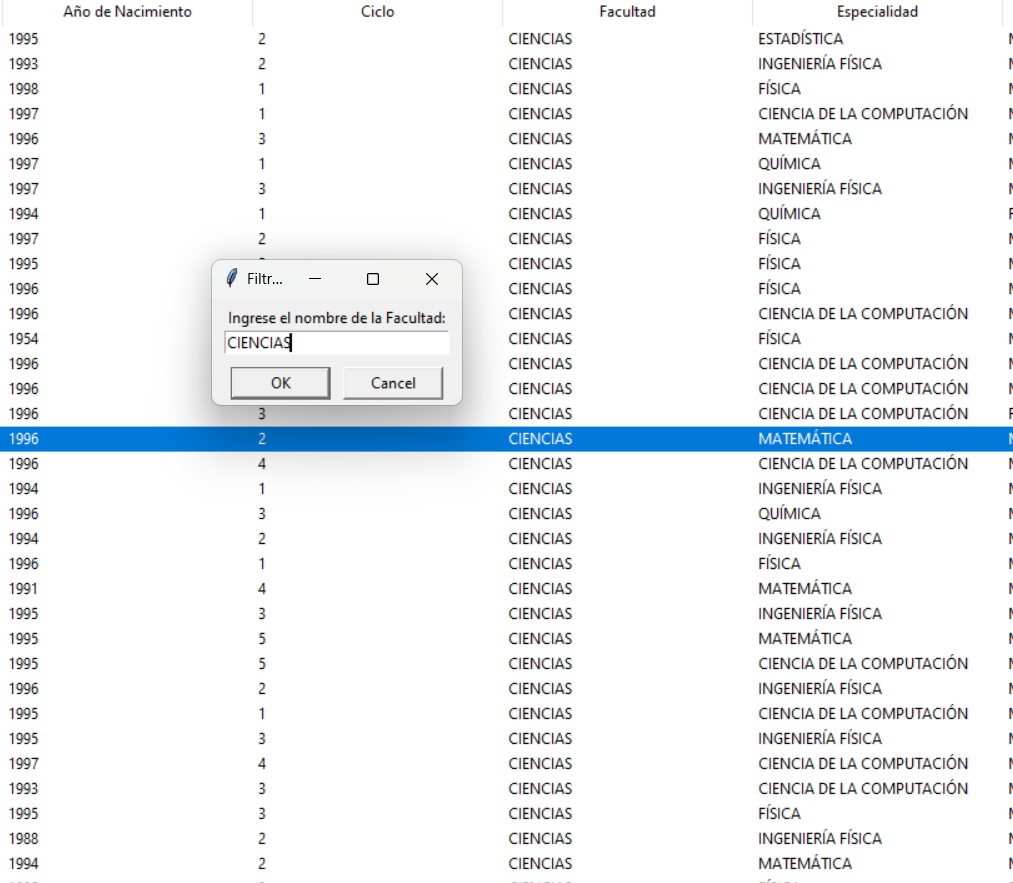
ALGUNO DE LOS RESULTADOS;



****

****

****

****

**BIBLIOGRAFIA:**

<https://www.datosabiertos.gob.pe/dataset/alumnos-matriculados-en-la-universidad-nacional-de-ingenier%C3%ADa-uni>

Oliphant, T. E. (2006). A Guide to NumPy. USA: Trelgol Publishing.

Wes McKinney. (2012). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly Media.

<https://matplotlib.org/stable/api/backend_pdf_api.html>

<https://docs.python.org/3/library/tkinter.html>

**CONCLUSIÓN**

La aplicación MatriculasApp proporciona una herramienta poderosa y flexible para la manipulación, análisis y visualización de datos de matrículas. Con una interfaz intuitiva y diversas funcionalidades, permite a los usuarios explorar datos complejos de manera efectiva y obtener insights valiosos a través de gráficos y estadísticas detalladas.

ANEXOSOS

import tkinter as tk

from tkinter import ttk, messagebox, filedialog, simpledialog

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from matplotlib.backends.backend\_pdf import PdfPages

class MatriculasApp:

    def \_\_init\_\_(self, root):

        self.root = root

        self.root.title("Matriculas 2016-2024")

        self.root.geometry("1000x600")

        # DataFrame para almacenar los datos

        self.df = None

        # Restablecer

        self.original\_df = None

        # Crear Frame principal

        self.main\_frame = ttk.Frame(self.root)

        self.main\_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

        # Crear Frame para los botones

        self.button\_frame = ttk.Frame(self.main\_frame)

        self.button\_frame.pack(side=tk.RIGHT, fill=tk.Y, padx=10, pady=10)

        # Crear botones

        self.open\_button = ttk.Button(self.button\_frame, text="Abrir archivo", command=self.file\_open)

        self.open\_button.pack(side=tk.TOP, padx=5, pady=5)

        self.plot\_bar\_button = ttk.Button(self.button\_frame, text="Graficar barras", command=self.graficar\_barras)

        self.plot\_bar\_button.pack(side=tk.TOP, padx=5, pady=5)

        # Menú desplegable para seleccionar la columna para el gráfico de barras

        self.column\_selection = tk.StringVar()

        self.column\_selection.set("Seleccione una columna")

        self.column\_menu = ttk.OptionMenu(self.button\_frame, self.column\_selection, "Seleccione una columna",

                                          "COLEGIO\_DEPA", "COLEGIO0\_PROV", "COLEGIO\_DIST", "ANIO",

                                          "PERIODO", "TIPO\_MATRICULA", "DOMICILIO\_DEPA", "DOMICILIO\_PROV",

                                          "DOMICILIO\_DIST", "ANIO\_NACIMIENTO", "NACIMIENTO\_PAIS",

                                          "NACIMIENTO\_DEPA", "NACIMIENTO\_PROV", "NACIMIENTO\_DIST",

                                          "SEXO", "MODALIDAD", "METODOLOGIA", "FACULTAD", "ESPECIALIDAD",

                                          "CICLO\_RELATIVO")

        self.column\_menu.pack(side=tk.TOP, padx=5, pady=5)

        # Seguir con los botones

        self.plot\_heatmap\_button = ttk.Button(self.button\_frame, text="Graficar mapa de calor", command=self.graficar\_heatmap)

        self.plot\_heatmap\_button.pack(side=tk.TOP, padx=5, pady=5)

        self.search\_button = ttk.Button(self.button\_frame, text="Buscar por IDHASH", command=self.buscar\_por\_idhash)

        self.search\_button.pack(side=tk.TOP, padx=5, pady=5)

        # Campo de entrada para búsqueda por IDHASH

        self.idhash\_entry = ttk.Entry(self.button\_frame)

        self.idhash\_entry.pack(side=tk.TOP, padx=5, pady=5)

        # Botones

        self.filter\_faculty\_button = ttk.Button(self.button\_frame, text="Filtrar por Facultad", command=self.filtrar\_por\_facultad)

        self.filter\_faculty\_button.pack(side=tk.TOP, padx=5, pady=5)

        self.filter\_specialty\_button = ttk.Button(self.button\_frame, text="Filtrar por Especialidad", command=self.filtrar\_por\_especialidad)

        self.filter\_specialty\_button.pack(side=tk.TOP, padx=5, pady=5)

        self.filter\_nacimiento\_depa\_button = ttk.Button(self.button\_frame, text="Filtrar por departamento de Nacimiento", command=self.filtrar\_por\_nacimiento\_depa)

        self.filter\_nacimiento\_depa\_button.pack(side=tk.TOP, padx=5, pady=5)

        self.filter\_colegio\_dist\_button = ttk.Button(self.button\_frame, text="Filtrar por Colegio Distrital", command=self.filtrar\_por\_colegio\_dist)

        self.filter\_colegio\_dist\_button.pack(side=tk.TOP, padx=5, pady=5)

        self.filter\_by\_year\_button = ttk.Button(self.button\_frame, text="Filtrar por Año", command=self.filtrar\_por\_anio)

        self.filter\_by\_year\_button.pack(side=tk.TOP, padx=5, pady=5)

        self.filter\_by\_birth\_year\_button = ttk.Button(self.button\_frame, text="Filtrar por Año de Nacimiento", command=self.filtrar\_por\_anio\_nacimiento)

        self.filter\_by\_birth\_year\_button.pack(side=tk.TOP, padx=5, pady=5)

        self.matricula\_stats\_button = ttk.Button(self.button\_frame, text="Estadísticas de Matrículas", command=self.matricula\_stats)

        self.matricula\_stats\_button.pack(side=tk.TOP, padx=5, pady=5)

        self.reset\_button = ttk.Button(self.button\_frame, text="Restablecer Base de Datos", command=self.restablecer\_base\_datos)

        self.reset\_button.pack(side=tk.TOP, padx=5, pady=5)

        self.save\_pdf\_button = ttk.Button(self.button\_frame, text="Guardar como PDF", command=self.guardar\_como\_pdf)

        self.save\_pdf\_button.pack(side=tk.TOP, padx=5, pady=5)

        self.export\_button = ttk.Button(self.button\_frame, text="Exportar datos", command=self.exportar\_datos)

        self.export\_button.pack(side=tk.TOP, padx=5, pady=5)

        self.exit\_button = ttk.Button(self.button\_frame, text="Salir", command=self.salir)

        self.exit\_button.pack(side=tk.TOP, pady=10)

        # Crear Frame para datos con barras de desplazamiento

        self.data\_frame = ttk.Frame(self.main\_frame)

        self.data\_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=10, pady=10)

        # Crear barras de desplazamiento

        self.scroll\_y = ttk.Scrollbar(self.data\_frame, orient="vertical")

        self.scroll\_y.pack(side=tk.RIGHT, fill=tk.Y)

        self.scroll\_x = ttk.Scrollbar(self.data\_frame, orient="horizontal")

        self.scroll\_x.pack(side=tk.BOTTOM, fill=tk.X)

        # Crear Treeview para mostrar los datos

        self.tree = ttk.Treeview(self.data\_frame, columns=("IDHASH", "anio", "anio\_nacimiento", "ciclo", "facultad", "especialidad", "sexo", "modalidad", "domicilio\_depa", "colegio\_dist"), show='headings', yscrollcommand=self.scroll\_y.set, xscrollcommand=self.scroll\_x.set)

        self.tree.heading("IDHASH", text="Codigo")

        self.tree.heading("anio", text="Año")

        self.tree.heading("anio\_nacimiento", text="Año de Nacimiento")

        self.tree.heading("ciclo", text="Ciclo")

        self.tree.heading("facultad", text="Facultad")

        self.tree.heading("especialidad", text="Especialidad")

        self.tree.heading("sexo", text="Sexo")

        self.tree.heading("modalidad", text="Modalidad")

        self.tree.heading("domicilio\_depa", text="Departamento")

        self.tree.heading("colegio\_dist", text="Colegio Distrital")

        self.tree.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

        # Configurar el comportamiento de las barras de desplazamiento

        self.scroll\_y.config(command=self.tree.yview)

        self.scroll\_x.config(command=self.tree.xview)

    def file\_open(self):

        file\_path = filedialog.askopenfilename(filetypes=[("CSV files", "\*.csv"), ("All files", "\*.\*")])

        if file\_path:

            try:

                self.df = pd.read\_csv(file\_path)

                self.original\_df = self.df.copy()

                self.display\_data()

            except Exception as e:

                messagebox.showerror("Error", str(e))

    def display\_data(self):

        for row in self.tree.get\_children():

            self.tree.delete(row)

        for index, row in self.df.iterrows():

            self.tree.insert("", "end", text=index, values=(row["IDHASH"], row["ANIO"], row["ANIO\_NACIMIENTO"], row["CICLO\_RELATIVO"], row["FACULTAD"], row["ESPECIALIDAD"], row["SEXO"], row["MODALIDAD"], row["DOMICILIO\_DEPA"], row["COLEGIO\_DIST"]))

    def graficar\_barras(self):

        if self.df is not None:

            column = self.column\_selection.get()

            if column != "Seleccione una columna":

                try:

                    self.df[column].value\_counts().plot(kind='bar')

                    plt.title(f"Distribución de {column}")

                    plt.xlabel(column)

                    plt.ylabel("Frecuencia")

                    plt.show()

                except KeyError:

                    messagebox.showerror("Error", f"La columna '{column}' no se encuentra en el DataFrame.")

            else:

                messagebox.showinfo("Información", "Por favor seleccione una columna para graficar.")

    def graficar\_heatmap(self):

        if self.df is not None:

            numeric\_columns = ['ANIO', 'ANIO\_NACIMIENTO']  # Añadir otras columnas numéricas según sea necesario

            correlation\_matrix = self.df[numeric\_columns].corr()

            sns.heatmap(correlation\_matrix, annot=True, cmap="coolwarm")

            plt.title("Mapa de Calor de Correlaciones")

            plt.show()

    def filtrar\_por\_facultad(self):

        if self.df is not None:

            facultad = simpledialog.askstring("Filtrar por Facultad", "Ingrese el nombre de la Facultad:")

            if facultad:

                filtered\_df = self.df[self.df["FACULTAD"] == facultad]

                self.df = filtered\_df

                self.display\_data()

    def filtrar\_por\_especialidad(self):

        if self.df is not None:

            especialidad = simpledialog.askstring("Filtrar por Especialidad", "Ingrese el nombre de la Especialidad:")

            if especialidad:

                filtered\_df = self.df[self.df["ESPECIALIDAD"] == especialidad]

                self.df = filtered\_df

                self.display\_data()

    def filtrar\_por\_nacimiento\_depa(self):

        if self.df is not None:

            nacimiento\_depa = simpledialog.askstring("Filtrar por Departamento de Nacimiento", "Ingrese el nombre del Departamento de Nacimiento:")

            if nacimiento\_depa:

                filtered\_df = self.df[self.df["NACIMIENTO\_DEPA"] == nacimiento\_depa]

                self.df = filtered\_df

                self.display\_data()

    def filtrar\_por\_colegio\_dist(self):

        if self.df is not None:

            colegio\_dist = simpledialog.askstring("Filtrar por Colegio Distrital", "Ingrese el nombre del Colegio Distrital:")

            if colegio\_dist:

                filtered\_df = self.df[self.df["COLEGIO\_DIST"] == colegio\_dist]

                self.df = filtered\_df

                self.display\_data()

    def filtrar\_por\_anio(self):

        if self.df is not None:

            anio = simpledialog.askinteger("Filtrar por Año", "Ingrese el Año:")

            if anio:

                filtered\_df = self.df[self.df["ANIO"] == anio]

                self.df = filtered\_df

                self.display\_data()

    def filtrar\_por\_anio\_nacimiento(self):

        if self.df is not None:

            anio\_nacimiento = simpledialog.askinteger("Filtrar por Año de Nacimiento", "Ingrese el Año de Nacimiento:")

            if anio\_nacimiento:

                filtered\_df = self.df[self.df["ANIO\_NACIMIENTO"] == anio\_nacimiento]

                self.df = filtered\_df

                self.display\_data()

    def buscar\_por\_idhash(self):

        if self.df is not None:

            idhash = self.idhash\_entry.get()

            if idhash:

                filtered\_df = self.df[self.df["IDHASH"] == idhash]

                if not filtered\_df.empty:

                    self.df = filtered\_df

                    self.display\_data()

                else:

                    messagebox.showinfo("Información", "IDHASH no encontrado.")

    def matricula\_stats(self):

        if self.df is not None:

            # Crear la primera ventana con las estadísticas de tipo de matrícula

            tipo\_matricula\_window = tk.Toplevel(self.root)

            tipo\_matricula\_window.title("Estadísticas de Tipo de Matrícula")

            stats\_tipo\_matricula = self.df['TIPO\_MATRICULA'].value\_counts().to\_string()

            tk.Label(tipo\_matricula\_window, text=stats\_tipo\_matricula).pack()

            # Crear la segunda ventana con las estadísticas por año, facultad, ciclo y especialidad

            stats\_window = tk.Toplevel(self.root)

            stats\_window.title("Estadísticas Detalladas")

            # Crear un Frame dentro de la ventana de estadísticas

            stats\_frame = tk.Frame(stats\_window)

            stats\_frame.pack(fill=tk.BOTH, expand=True)

            # Crear la barra de desplazamiento

            scrollbar = tk.Scrollbar(stats\_frame, orient="vertical")

            scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

            # Crear el widget Text para mostrar las estadísticas

            stats\_text\_widget = tk.Text(stats\_frame, wrap=tk.WORD, height=30, width=80, yscrollcommand=scrollbar.set)

            stats\_text\_widget.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

            scrollbar.config(command=stats\_text\_widget.yview)

            # Crear el contenido de la ventana

            stats\_text = ""

            for year in sorted(self.df["ANIO"].unique()):

                df\_year = self.df[self.df["ANIO"] == year]

                stats\_text += f"Año {year}:\n"

                for facultad in df\_year["FACULTAD"].unique():

                    df\_facultad = df\_year[df\_year["FACULTAD"] == facultad]

                    total = len(df\_facultad)

                    stats\_text += f"  {facultad}: {total} estudiantes\n"

                    for especialidad in df\_facultad["ESPECIALIDAD"].unique():

                        df\_especialidad = df\_facultad[df\_facultad["ESPECIALIDAD"] == especialidad]

                        stats\_text += f"    Especialidad {especialidad}: {len(df\_especialidad)} estudiantes\n"

                        for ciclo in df\_especialidad["CICLO\_RELATIVO"].unique():

                            df\_ciclo = df\_especialidad[df\_especialidad["CICLO\_RELATIVO"] == ciclo]

                            stats\_text += f"      Ciclo {ciclo}: {len(df\_ciclo)} estudiantes\n"

            # Insertar el contenido en el widget Text

            stats\_text\_widget.insert(tk.END, stats\_text)

            stats\_text\_widget.config(state=tk.DISABLED)

    def guardar\_como\_pdf(self):

        if self.df is not None:

            try:

                with PdfPages('estadisticas\_matriculas.pdf') as pdf:

                    # Gráfico de barras por año

                    plt.figure(figsize=(10, 6))

                    self.df['ANIO'].value\_counts().plot(kind='bar')

                    plt.title('Número de Matrículas por Año')

                    plt.xlabel('Año')

                    plt.ylabel('Número de Matrículas')

                    pdf.savefig()

                    plt.close()

                    # Mapa de calor de correlación

                    plt.figure(figsize=(10, 8))

                    numeric\_columns = ['ANIO', 'ANIO\_NACIMIENTO']  # Añadir otras columnas numéricas según sea necesario

                    correlation\_matrix = self.df[numeric\_columns].corr()

                    sns.heatmap(correlation\_matrix, annot=True, cmap="coolwarm")

                    plt.title('Mapa de Calor de Correlaciones')

                    pdf.savefig()

                    plt.close()

                messagebox.showinfo("Éxito", "Datos guardados en PDF con éxito.")

            except Exception as e:

                messagebox.showerror("Error", f"Error al guardar el PDF: {e}")

    def exportar\_datos(self):

        if self.df is not None:

            file\_path = filedialog.asksaveasfilename(defaultextension=".csv", filetypes=[("CSV files", "\*.csv")])

            if file\_path:

                try:

                    self.df.to\_csv(file\_path, index=False)

                    messagebox.showinfo("Éxito", "Datos exportados con éxito.")

                except Exception as e:

                    messagebox.showerror("Error", f"Error al exportar los datos: {e}")

    def restablecer\_base\_datos(self):

        self.df = self.original\_df.copy() if self.original\_df is not None else None

        self.display\_data()

    def salir(self):

        self.root.quit()

if \_\_name\_\_ == "\_\_main\_\_":

    root = tk.Tk()

    app = MatriculasApp(root)

    root.mainloop()