## Vicerrectorado Uagrm - ULLOA: EL ESCUDO DE LA UAGRM FUE REGISTRADO EN EL SENAPI COMO MARCA INSTITUCIONAL El Escudo de la Universidad Autónoma Gabriel René Moreno (UAGRM) fue registrado en el Servicio Sistema Experto de Recomendación de Películas

**Docente:**

**Vargas Peña Leonardo**

**Estudiantes:**

* **Mandepora Perez Yerling**
* **Chilimani Cuellar Edson Neil**
* **Ayelen Estevez Segovia**

**Facultad Integral Del Chaco, Universidad Autónoma Gabriel Rene Moreno**

**Ingeniería en Informática**

**Tecnología Web**

**Camiri-Santa Cruz-Bolivia**

**2023**

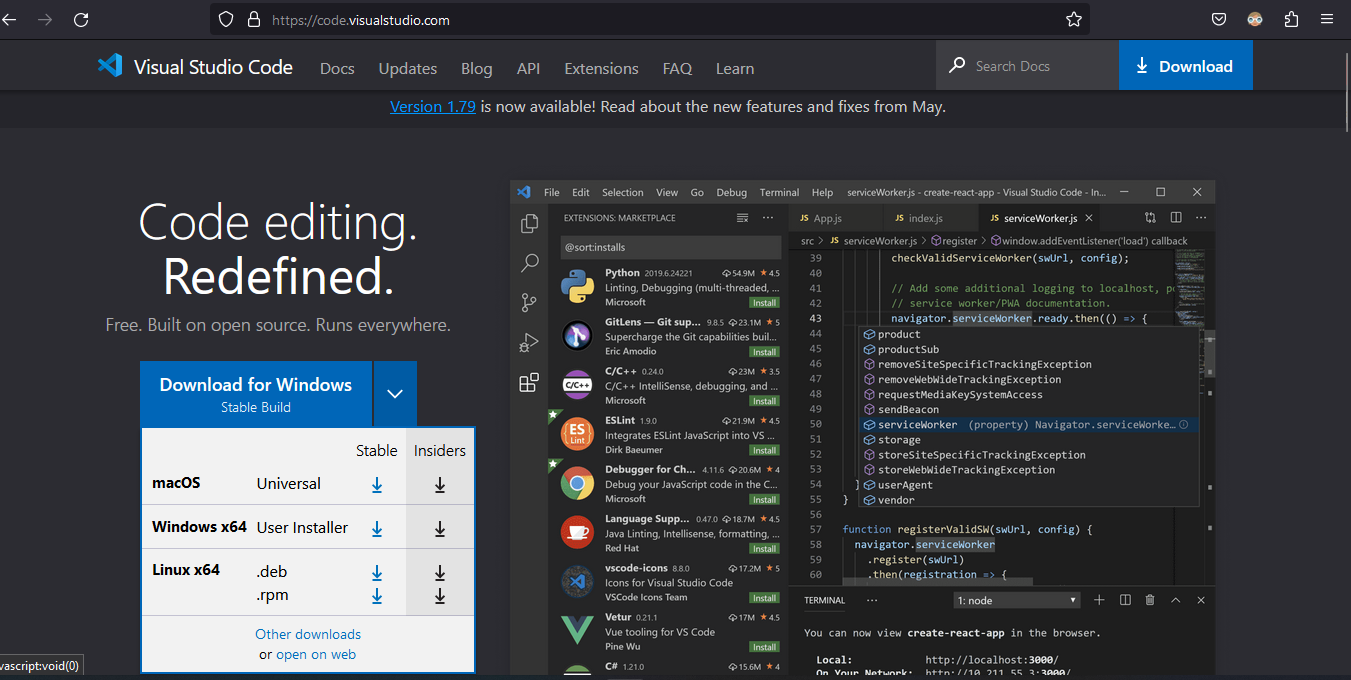
Para el presente proyecto se realizará un Sistema de Recomendación de Películas que se utilizará el lenguaje Python con el gestor Sqlite3 y el editor de visual studio code para lograr el objetivo de construir este Sistema Experto. Todo esto se realizará en el sistema operativo de “Windows”

Herramientas que se utilizara:

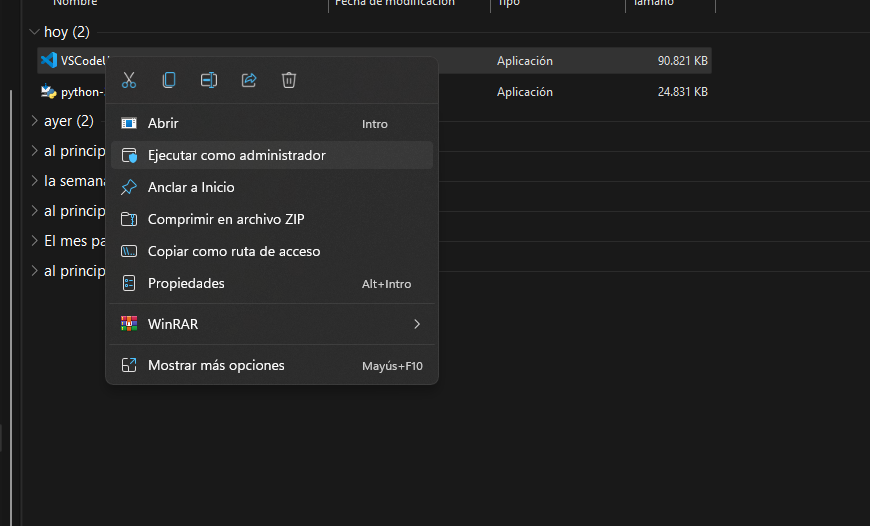
* Visual studio code “editor de texto”.
* Python
* Sqlite3
* Biblioteca Pgmpy

**INSTALACION DEL EDITOR DE TEXTO VISUAL STUDIO CODE**

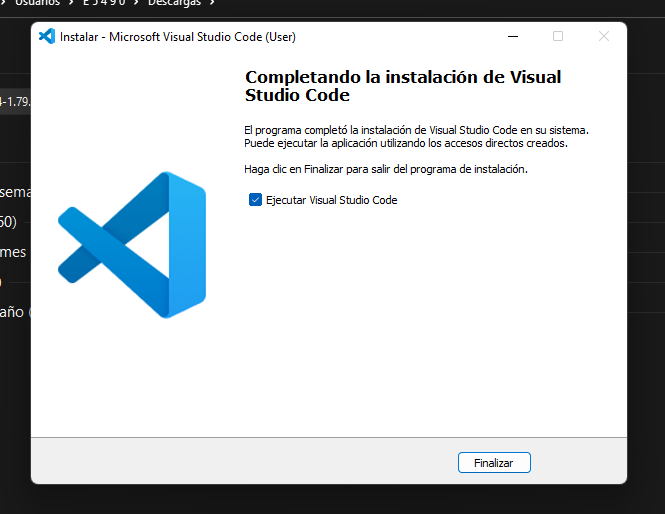
Entramos al sitio oficial de Vscode y le damos descargar según nuestro sistema operativo verificando así también si es de 32 o 64 bits.



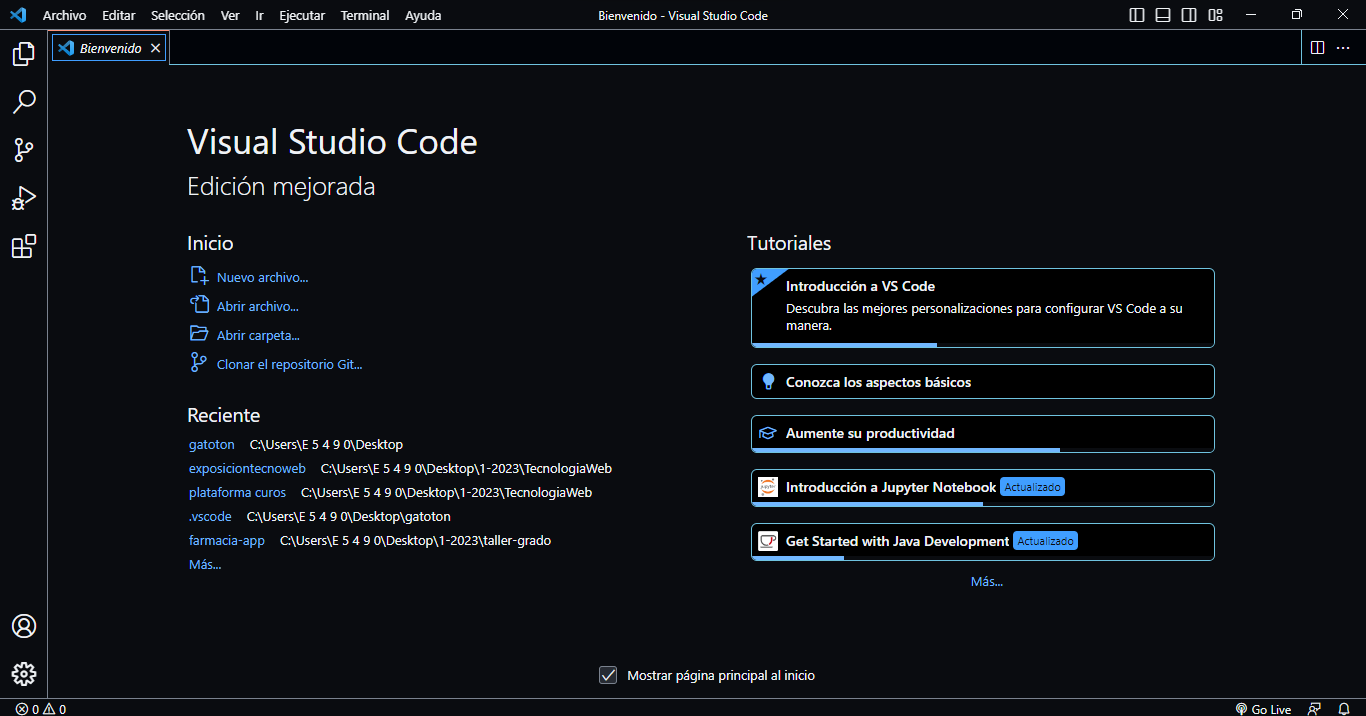
Lo ejecutamos como administrador para que no surjan problemas



Y lo instalamos siguiendo los pasos que nos indique.



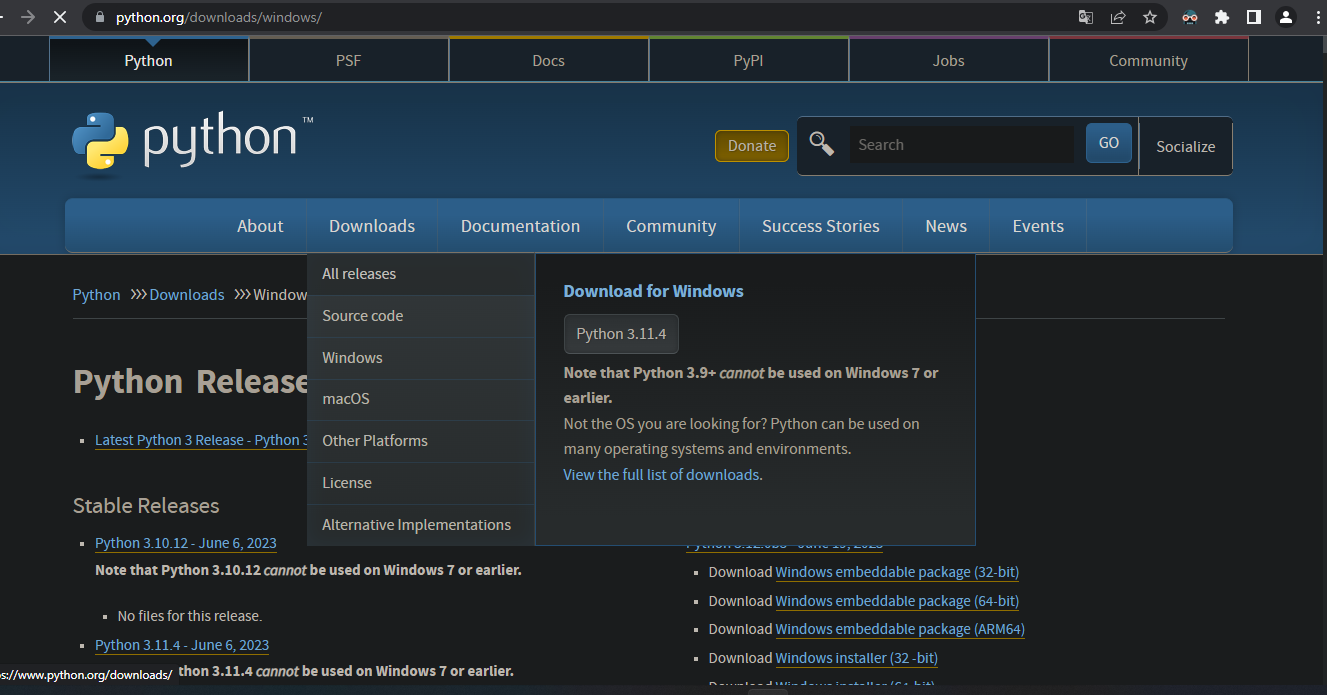
Y ahora todo listo para empezar a utilizarlo



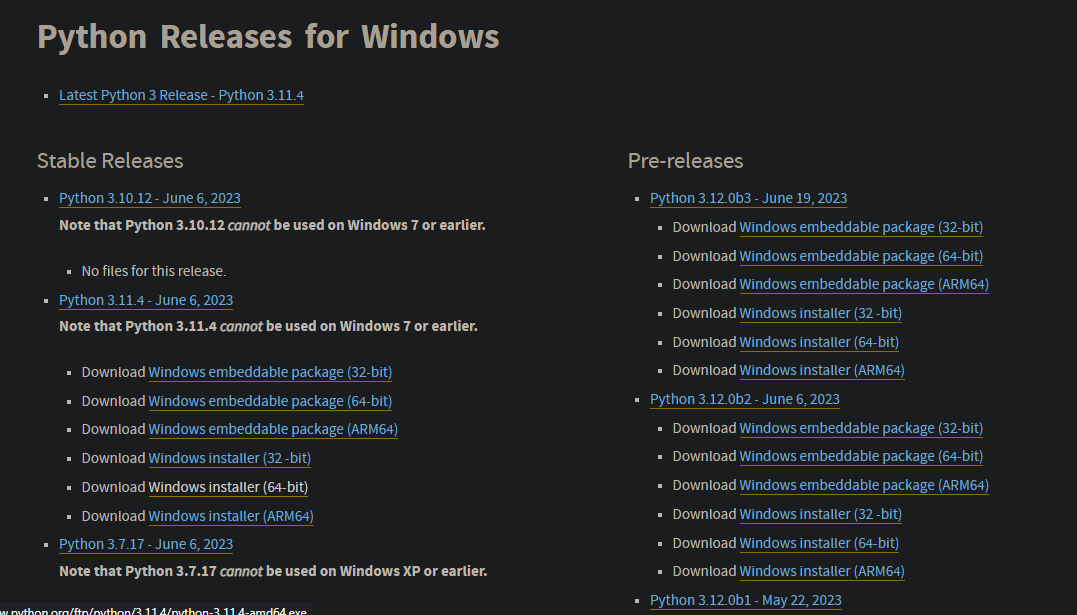
**INSTALACION DE PYTHON**

Instalación de lenguaje de programación en el que se trabajara, en este caso PYTHON

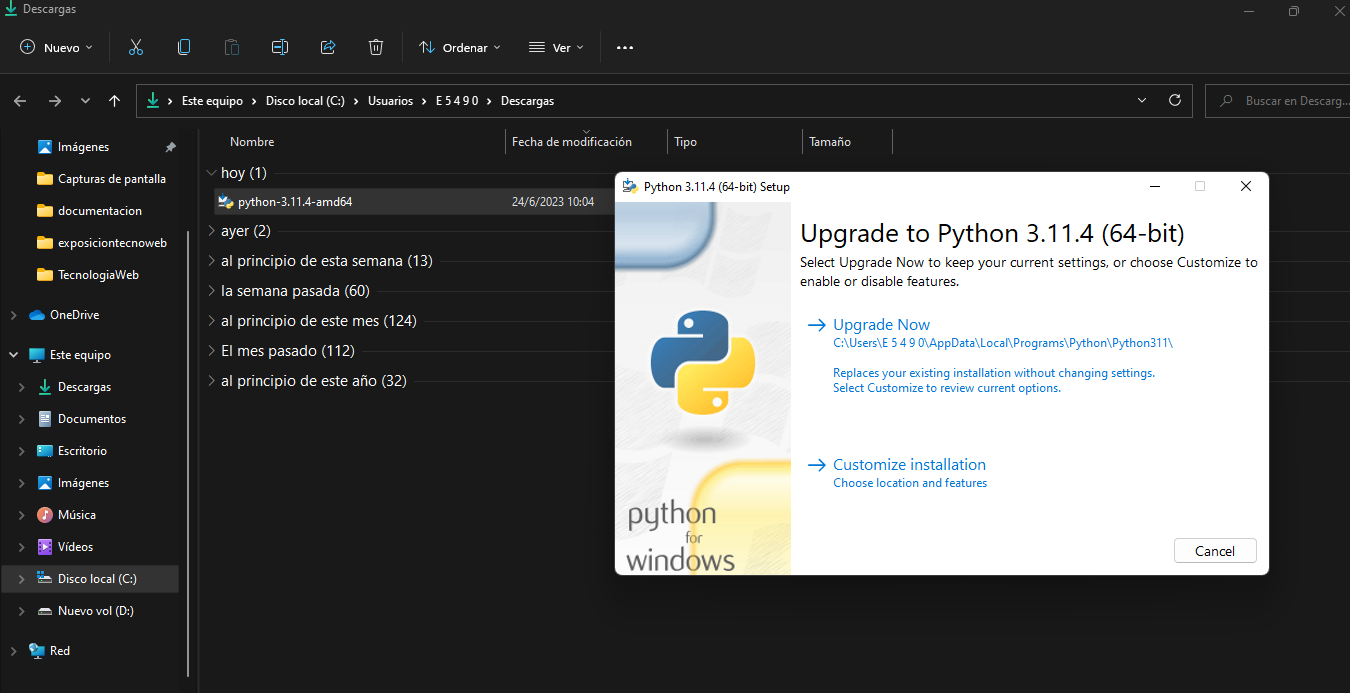
-Se visita la página oficial de Python.



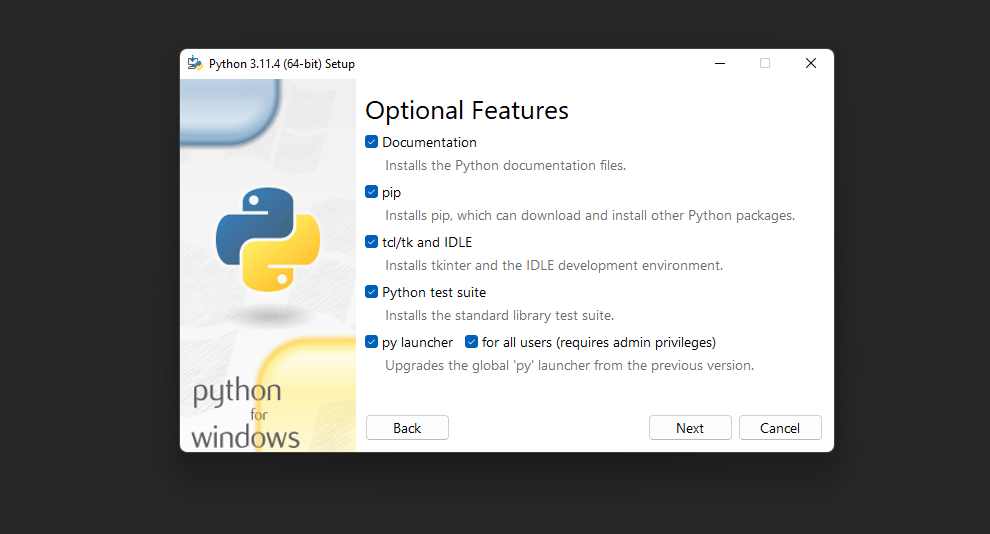
-Se descarga la versión más reciente de Python para evitar problemas de compatibilidad con las dependencias que se utilizara en el proyecto, así mismo tener en cuenta descargar según el sistema operativo del equipo(32bits o 64 bits).



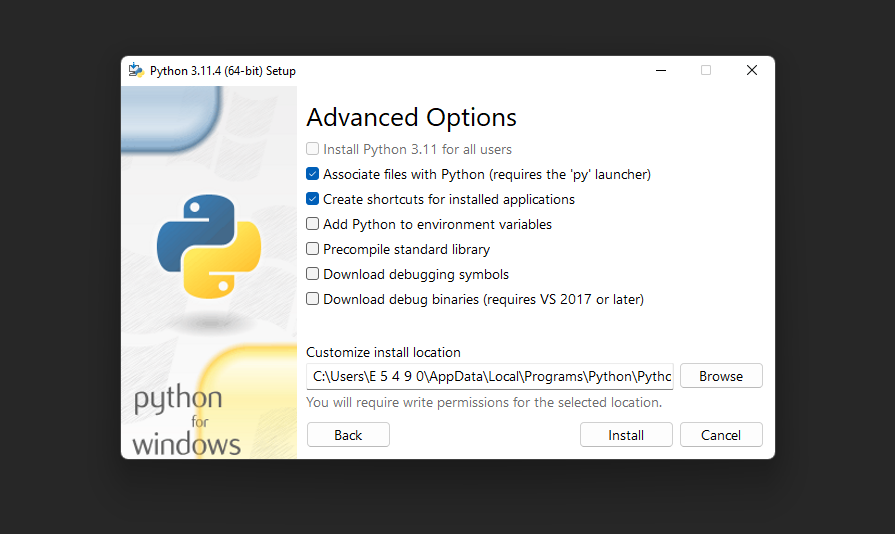
Lo buscamos en nuestros archivo el instalador de Python que descargamos y hacemos doble click en el para ejecutarlo e instalar el software.



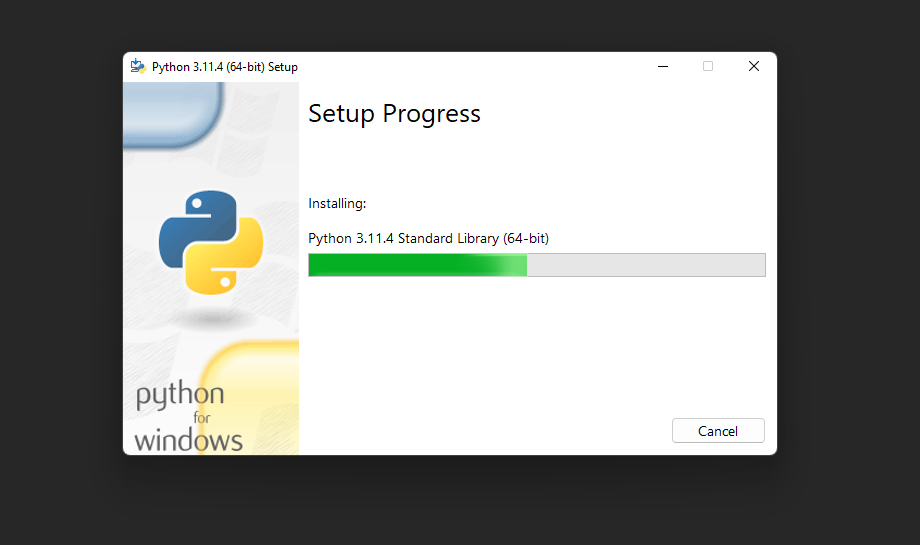
-Al instalar Python automáticamente se estará descargando “PIP” que es un administrador de paquetes que nos permitirá buscar e instalar paquetes específicos de Python con todas sus dependencias automáticamente, lo que facilita la gestión de las bibliotecas y asegura que todas las dependencias requeridas estén correctamente instaladas en tu entorno de desarrollo.



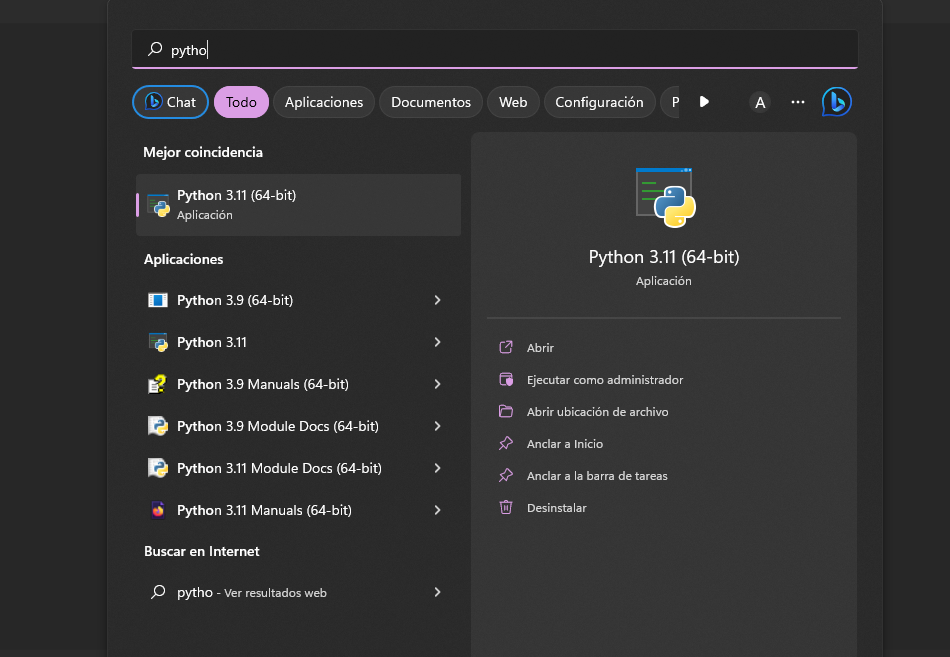
Te muestra en que parte de tu directorio se descargara, donde si deseas puedes cambiarlo.



Presionamos instalar y esperamos que la instalación finalize



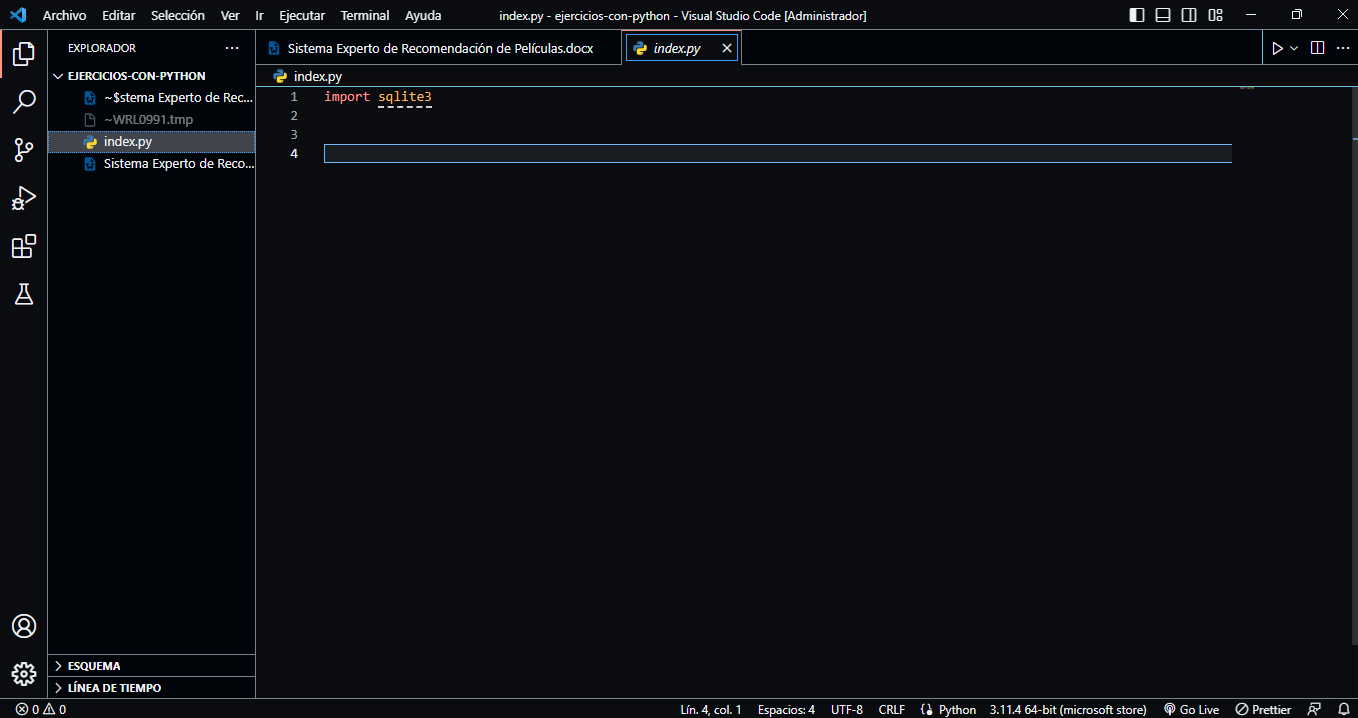
Y listo,cuando se lo busque el software de Python ya estar instalado correctamente en el equipo.



**IMPORTAR EN EL SCRIPT DE PYTHON SQLITE3**

SQLite3 es un módulo incorporado en la biblioteca estándar de Python, por lo que no es necesario instalarlo por separado.

Dentro de nuestro entorno de trabajo importamos la siguiente línea “import sqlite3” , esto para poder trabajar con el gestor de base de datos sqlite3 en nuestro proyecto.



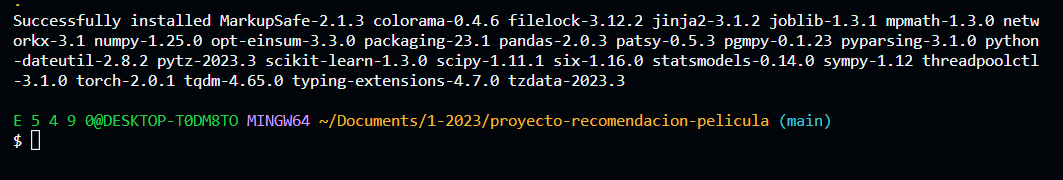
**INSTALACION DE LA BIBLIOTECA “PGMPY”**

La biblioteca pgmpy no viene preinstalado con Python. Entonces debemos instalarlo por separado para poder utilizarlo en el proyecto.

Para instalar pgmpy, utilizaremos el administrador de paquetes de Python, pip, ejecutando el siguiente comando en la línea de comandos de la terminal:

“pip install pgmpy”





Este comando descargará e instalará el paquete pgmpy y todas sus dependencias en el entorno de Python.

Después de la instalación, procedemos a importar pgmpy en el código Python utilizando la declaración “import pgmpy”.

import pgmpy

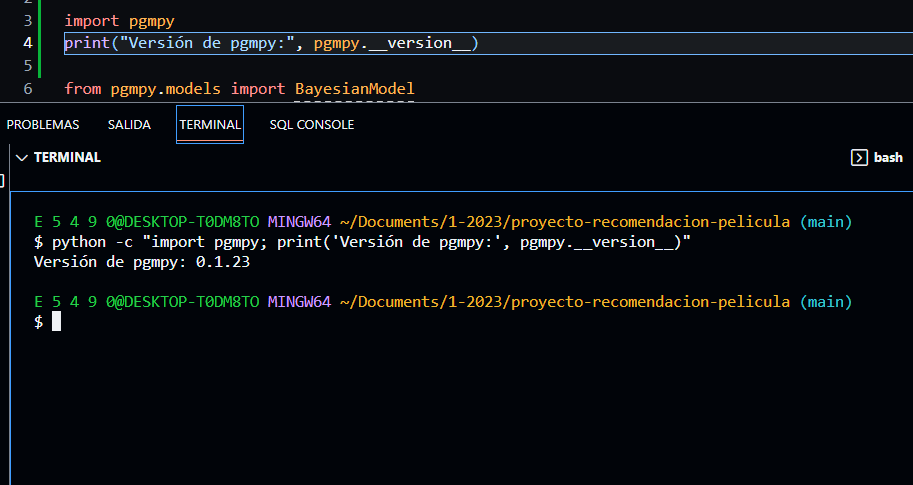
Para verificar la versión de la biblioteca pgmpy instalada en el entorno de Python, yu ver si se instalo correctamente procederemos a incoporar el siguiente código:

print("Versión de pgmpy:", pgmpy.\_\_version\_\_)

Luego en la terminal debería lanzar la versión de nuestra biblioteca pgmpy que hemos instalado, para mostrar utilizaremos este comando usando el interprete de python:

**python -c "import pgmpy; print('Versión de pgmpy:', pgmpy.\_\_version\_\_)"**

Mostraria una salida con la versión que tenemos de pgmpy:



Listo ahora el módulo de sqlite3 y la biblioteca de pgmpy ya están listas, ahora se procederá realizar el sistema experto de recomendación de películas, con estas herramientas ya establecidas podemos empezar a trabajar sin problemas.

**DESARROLLO**

Primero definimos las clases correspondientes con las que constara nuestro sistema experto:

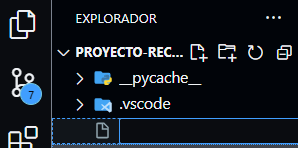
* Usuario.
* RecomendaciónPelicula.
* Calificacion.
* Pelicula.
* Clasificacion.
* Genero

Aclarar que para mantener estructurado y ordenado el código se opta por crear archivos dependiendo de lo que se utilizara en el proyecto de sistema experto de recomendación de películas.

**Base de Conocimiento**

**CREACION DEL ARCHIVO “BASE\_CONOCIMIENTO.PY”**

Dentro de nuestro proyecto decimos “crear nuevo archivo”



Colocamos el nombre en este caso será”base\_conocimiento” y vemos como se crea , ahora esta listo para crear lo necesario dentro de este archivo.py



Dentro de este crearemos la base de conocimiento utilizando el “Modelo bayesiano”

base\_conocimiento = create\_bayesian\_model()

A continuación lo veremos mejor representado en el código completo.

Primeramente, importamos:

from pgmpy.models import BayesianModel

from pgmpy.factors.discrete import TabularCPD

Con esto podremos trabajar con el modelo bayesiano sin problemas.

**CODIGO COMPLETO DE EL ARCHIVO “BASE\_CONOCIMIENTO.PY”**

from pgmpy.models import BayesianModel

from pgmpy.factors.discrete import TabularCPD

def create\_bayesian\_model():

    # Crear el objeto de la red bayesiana

    modelo\_bayesiano = BayesianModel()

    # Definir las variables

    variables = ["Usuario", "RecomendacionPelicula", "Calificacion", "Pelicula", "Clasificacion", "Genero"]

    # Agregar las variables al modelo

    modelo\_bayesiano.add\_nodes\_from(variables)

    # Definir las relaciones entre las variables

    modelo\_bayesiano.add\_edges\_from([

        ("Usuario", "RecomendacionPelicula"),

        ("Usuario", "Calificacion"),

        ("Pelicula", "RecomendacionPelicula"),

        ("Pelicula", "Calificacion"),

        ("Clasificacion", "Pelicula"),

        ("Genero", "Pelicula")

    ])

    # Definir las distribuciones de probabilidad condicional (CPDs) de las variables

     # Definir CPD para la variable "Usuario"

    cpd\_usuario = TabularCPD(variable="Usuario", variable\_card=2, values=[[0.6, 0.4]])

    # Definir CPD para la variable "RecomendacionPelicula"

    cpd\_recomendacion = TabularCPD(variable="RecomendacionPelicula", variable\_card=2,

                                   values=[[0.3, 0.6, 0.1, 0.9], [0.7, 0.4, 0.9, 0.1]],

                                   evidence=["Usuario", "Pelicula"],

                                   evidence\_card=[2, 2])

    # Definir CPD para la variable "Calificacion"

    cpd\_calificacion = TabularCPD(variable="Calificacion", variable\_card=2,

                                  values=[[0.2, 0.8], [0.7, 0.3]],

                                  evidence=["Usuario", "Pelicula"],

                                  evidence\_card=[2, 2])

    # Definir CPD para la variable "Pelicula"

    cpd\_pelicula = TabularCPD(variable="Pelicula", variable\_card=2,

                              values=[[0.3, 0.7], [0.6, 0.4], [0.8, 0.2], [0.1, 0.9]],

                              evidence=["Clasificacion", "Genero"],

                              evidence\_card=[2, 2])

    # Definir CPD para la variable "Clasificacion"

    cpd\_clasificacion = TabularCPD(variable="Clasificacion", variable\_card=2, values=[[0.5, 0.5]])

    # Definir CPD para la variable "Genero"

    cpd\_genero = TabularCPD(variable="Genero", variable\_card=2, values=[[0.5, 0.5]])

    # Asociar los CPDs al modelo

    modelo\_bayesiano.add\_cpds(cpd\_usuario, cpd\_recomendacion, cpd\_calificacion, cpd\_pelicula, cpd\_clasificacion,

                              cpd\_genero)

    return modelo\_bayesiano

# Crear la base de conocimiento utilizando el modelo bayesiano

base\_conocimiento = create\_bayesian\_model()

**CREACION DE EL ARCHIVO “CONEXIÓN\_SQLITE3.PY”**

Al igual que al anterior , creamos el archivo de la misma manera.

En este archivo se vera la conexión con el gestor de base de datos “sqlite3” mediante una función opara abrir y cerrar la conexión con la base de datos.

Asi mismo la creación de las tablas que utilizara el sistema experto de recomendación de película con sus respectivos campos o atributos.

Las librerías :

import hashlib

import os

sirven para poder encriptar el campo de contraseña.

import sqlite3

import pgmpy

print("Versión de pgmpy:", pgmpy.\_\_version\_\_)

from pgmpy.models import BayesianModel

from pgmpy.factors.discrete import TabularCPD

# para encriptado de contraseña

import hashlib

import os

# Conectar a la base de datos SQLite

def connect\_to\_database():

    conn = sqlite3.connect('database.db')

    cursor = conn.cursor()

    return conn, cursor

# Obtener la conexión y el cursor

conn, cursor = connect\_to\_database()

# Tabla Usuario

cursor.execute('''CREATE TABLE IF NOT EXISTS Usuario (

                    id INTEGER PRIMARY KEY AUTOINCREMENT,

                    nombre TEXT,

                    email TEXT,

                    password TEXT,

                    edad INTEGER)''')

# Tabla RecomendacionPelicula

cursor.execute('''CREATE TABLE IF NOT EXISTS RecomendacionPelicula (

                    id INTEGER PRIMARY KEY AUTOINCREMENT,

                    usuario\_id INTEGER,

                    pelicula\_id INTEGER,

                    FOREIGN KEY (usuario\_id) REFERENCES Usuario (id),

                    FOREIGN KEY (pelicula\_id) REFERENCES Pelicula (id))''')

# Tabla Calificacion

cursor.execute('''CREATE TABLE IF NOT EXISTS Calificacion (

                    id INTEGER PRIMARY KEY AUTOINCREMENT,

                    usuario\_id INTEGER,

                    pelicula\_id INTEGER,

                    descripcion TEXT,

                    FOREIGN KEY (usuario\_id) REFERENCES Usuario (id),

                    FOREIGN KEY (pelicula\_id) REFERENCES Pelicula (id))''')

# Tabla Pelicula

cursor.execute('''CREATE TABLE IF NOT EXISTS Pelicula (

                    id INTEGER PRIMARY KEY AUTOINCREMENT,

                    titulo TEXT,

                    director TEXT,

                    clasificacion\_id INTEGER,

                    genero\_id INTEGER,

                    FOREIGN KEY (clasificacion\_id) REFERENCES Clasificacion (id),

                    FOREIGN KEY (genero\_id) REFERENCES Genero (id))''')

# Tabla Clasificacion

cursor.execute('''CREATE TABLE IF NOT EXISTS Clasificacion (

                    id INTEGER PRIMARY KEY AUTOINCREMENT,

                    nombre TEXT)''')

# Tabla Genero

cursor.execute('''CREATE TABLE IF NOT EXISTS Genero (

                    id INTEGER PRIMARY KEY AUTOINCREMENT,

                    nombre TEXT)''')

# Cerrar conexión a la base de datos SQLite

def close\_connection(conn):

    conn.close()

# Función para encriptar la contraseña

def encrypt\_password(password):

    salt = os.urandom(16)  # Generar un salt aleatorio de 16 bytes

    password\_hash = hashlib.pbkdf2\_hmac('sha256', password.encode('utf-8'), salt, 100000)

    return salt.hex() + password\_hash.hex()

Ahora que tenemos bien definidas nuestras tablas con sus respectivos atributos pasamos al siguiente paso que trata de “recolección de datos”, esto con la finalidad de poder hacer uso de los datos recolectados para las consultas.

Llenamos los datos para cada tabla que creamos para nuestro sistema experto, respetando las relaciones que tienen cada una de ellas:

# Ejemplo de recolección de datos y guardado en la base de datos

def collect\_usuario\_data():

    usuario\_data = [

        ("Ayelen", "ayelenestevezs@gmail.com", encrypt\_password("ayelen"), 22),

        ("Jherlin", "mandeporayerling@gmail.com", encrypt\_password("jherlin"), 22),

        ("Neil", "neil20cuellar1@gmail.com", encrypt\_password("neil"), 22),

        ("Susane", "susane@gmail.com", encrypt\_password("susane"), 23),

        ("Carlos", "carlos@gmail.com", encrypt\_password("carlos"), 27),

        # Agrega más datos de usuario aquí

    ]

    cursor.executemany('INSERT INTO Usuario (nombre, email, password, edad) VALUES (?, ?, ?, ?)', usuario\_data)

def collect\_calificacion\_data():

    calificacion\_data = [

        (1, 1, "Excelente"),

        (2, 2, "Malo"),

        (3, 3, "Neutro"),

        (4, 4, "Malo"),

        (5, 5, "Excelente"),

        # Agrega más datos de calificación aquí

    ]

    cursor.executemany('INSERT INTO Calificacion (usuario\_id, pelicula\_id, descripcion) VALUES (?, ?, ?)', calificacion\_data)

def collect\_recomendacion\_data():

    recomendacion\_data = [

        (1, 1),

        (2, 2),

        (3, 3),

        (4, 4),

        (5, 5),

        # Agrega más datos de recomendación aquí

    ]

    cursor.executemany('INSERT INTO RecomendacionPelicula (usuario\_id, pelicula\_id) VALUES (?, ?)', recomendacion\_data)

def collect\_pelicula\_data():

    pelicula\_data = [

        ("Francotirador", "Clint Eastwood", 3, 1),

        ("Atraccion Peligrosa", "Ben Affleck", 3, 2),

        ("E.T", "Steven Spielberg", 1, 3),

        ("Corazonada", "Alejandro Montiel", 3, 4),

        ("Harry Potter", "Chris Columbus", 2, 5),

        # Agrega más datos de película aquí

    ]

    cursor.executemany('INSERT INTO Pelicula (titulo, director, clasificacion\_id, genero\_id) VALUES (?, ?, ?, ?)', pelicula\_data)

def collect\_clasificacion\_data():

    clasificacion\_data = [

        ("APT",),

        ("Personas menores de 15 años",),

        ("Personas mayores de 15 años",),

        # Agrega más datos de clasificación aquí

    ]

    cursor.executemany('INSERT INTO Clasificacion (nombre) VALUES (?)', clasificacion\_data)

def collect\_genero\_data():

    genero\_data = [

        ("Acción",),

        ("Suspenso",),

        ("Ciencia Ficción",),

        ("Suspenso",),

        ("Fantasía",),

        # Agrega más datos de género aquí

    ]

    cursor.executemany('INSERT INTO Genero (nombre) VALUES (?)', genero\_data)

# Ejecutar las funciones de recolección de datos

collect\_usuario\_data()

collect\_recomendacion\_data()

collect\_calificacion\_data()

collect\_pelicula\_data()

collect\_clasificacion\_data()

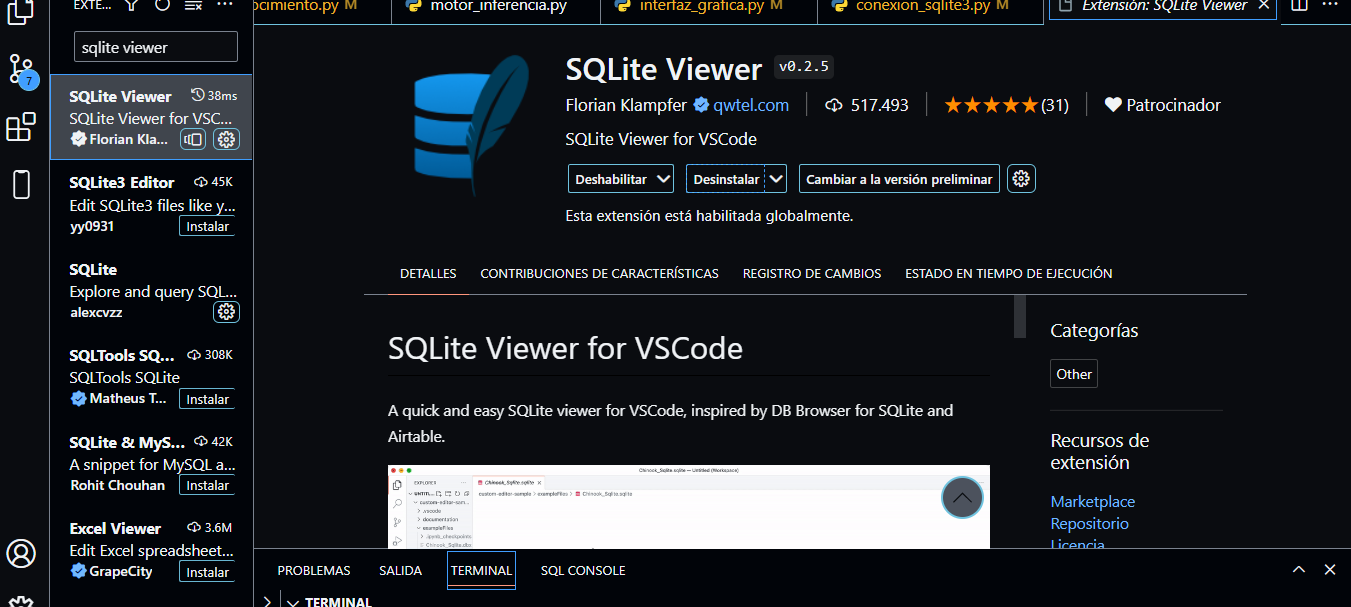
collect\_genero\_data()

# Guardar los cambios en la base de datos

conn.commit()

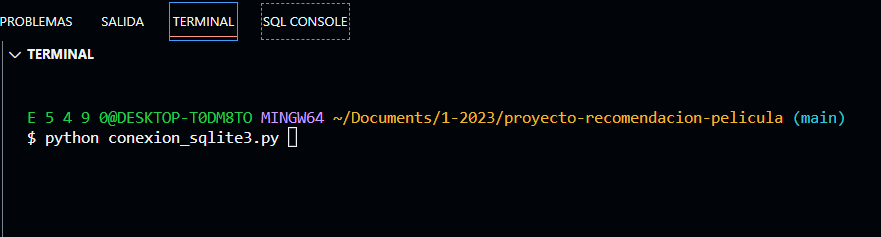
Ahora podremos visualizar los datos cuando lo carguemos a la base de datos ,pero para esto primero instalaremos una extensión en visual studio code que es una herramienta para administrar los datos de una BD.

Buscamos la extensión con el nombre de: SQLite viewer y lo instalamos

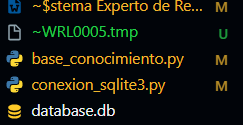


Ahora para cargar estos datos en la base de datos y para que se muestren utilizaremos el intérprete de Python con el siguiente comando:

“Python conexión\_sqlite3.py”



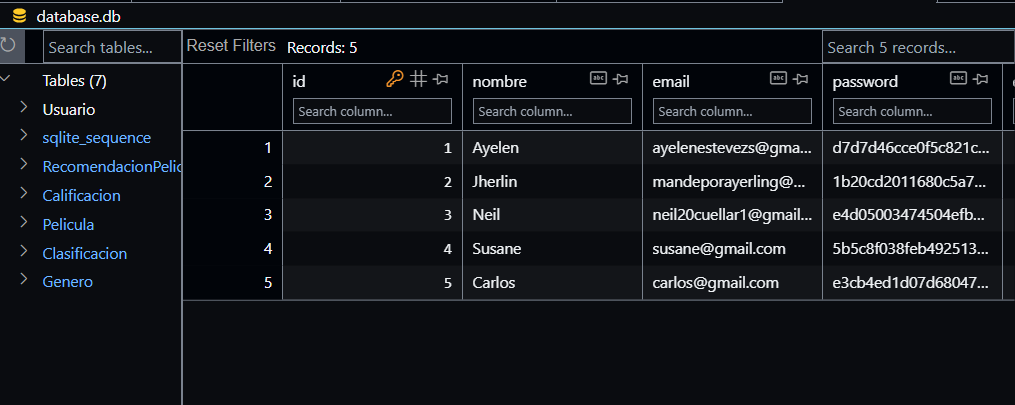
Con este comando y la extensión SQLite viewer instalada podemos visualizar y manipular los datos cargados. Al ejecutar este comando se creara un archivo llamado”database.db”



Y dentro de este archivo visualizaremos nuestros datos de todas nuestras tablas y podremos manipularlos.

Aclarar que algunas tablas al llamar las foránea se muestran las id , pero esto puede ser cambiado para que muestre nombre en ves de id.

**TABLA USUARIO**



**TABLA RECOMENDACIONPELICULA**

****

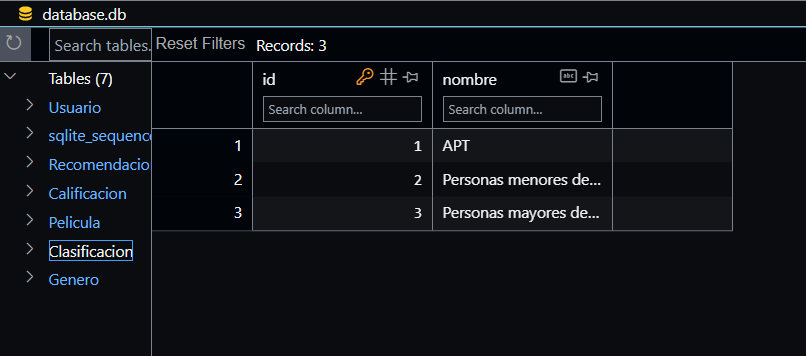
**TABLA CALIFICACION**

****

**TABLA PELICULA**

****

**TABLA CLASIFICACION**

****

**TABLA GENERO**

 Así mismo podemos crear consultas para ver los datos cargados mediante consola

**CONSULTAS PARA CADA TABLA:** Aquíen el código cambiamos para que en ves de mostrarse por las id se muestre por su nombre o descripción.

# Consulta para mostrar los nombres en lugar de IDs en la tabla RecomendacionPelicula

cursor.execute('''

    SELECT Usuario.nombre, Pelicula.titulo

    FROM RecomendacionPelicula

    JOIN Usuario ON RecomendacionPelicula.usuario\_id = Usuario.id

    JOIN Pelicula ON RecomendacionPelicula.pelicula\_id = Pelicula.id

''')

recomendaciones = cursor.fetchall()

print("Recomendaciones:")

for recomendacion in recomendaciones:

    print(f"Usuario: {recomendacion[0]}, Película: {recomendacion[1]}")

# Consulta para mostrar los nombres en lugar de IDs en la tabla Usuario

cursor.execute('SELECT nombre, email, password, edad FROM Usuario')

usuarios = cursor.fetchall()

print("\nTabla Usuario:")

for usuario in usuarios:

    print(f"Nombre: {usuario[0]}, Email: {usuario[1]}, Password: {usuario[2]}, Edad: {usuario[3]}")

# Consulta para mostrar los nombres en lugar de IDs en la tabla Pelicula

cursor.execute('''

    SELECT titulo, director, Clasificacion.nombre, Genero.nombre

    FROM Pelicula

    JOIN Clasificacion ON Pelicula.clasificacion\_id = Clasificacion.id

    JOIN Genero ON Pelicula.genero\_id = Genero.id

''')

peliculas = cursor.fetchall()

print("\nTabla Pelicula:")

for pelicula in peliculas:

    print(f"Título: {pelicula[0]}, Director: {pelicula[1]}, Clasificación: {pelicula[2]}, Género: {pelicula[3]}")

# Consulta para mostrar los nombres en lugar de IDs en la tabla Clasificacion

cursor.execute('SELECT nombre FROM Clasificacion')

clasificaciones = cursor.fetchall()

print("\nTabla Clasificacion:")

for clasificacion in clasificaciones:

    print(f"Nombre: {clasificacion[0]}")

# Consulta para mostrar los nombres en lugar de IDs en la tabla Genero

cursor.execute('SELECT nombre FROM Genero')

generos = cursor.fetchall()

print("\nTabla Genero:")

for genero in generos:

    print(f"Nombre: {genero[0]}")

# Consulta para mostrar los nombres en lugar de IDs en la tabla Calificacion

cursor.execute('''

    SELECT Usuario.nombre, Pelicula.titulo, Calificacion.descripcion

    FROM Calificacion

    JOIN Usuario ON Calificacion.usuario\_id = Usuario.id

    JOIN Pelicula ON Calificacion.pelicula\_id = Pelicula.id

''')

calificaciones = cursor.fetchall()

print("\nCalificaciones:")

for calificacion in calificaciones:

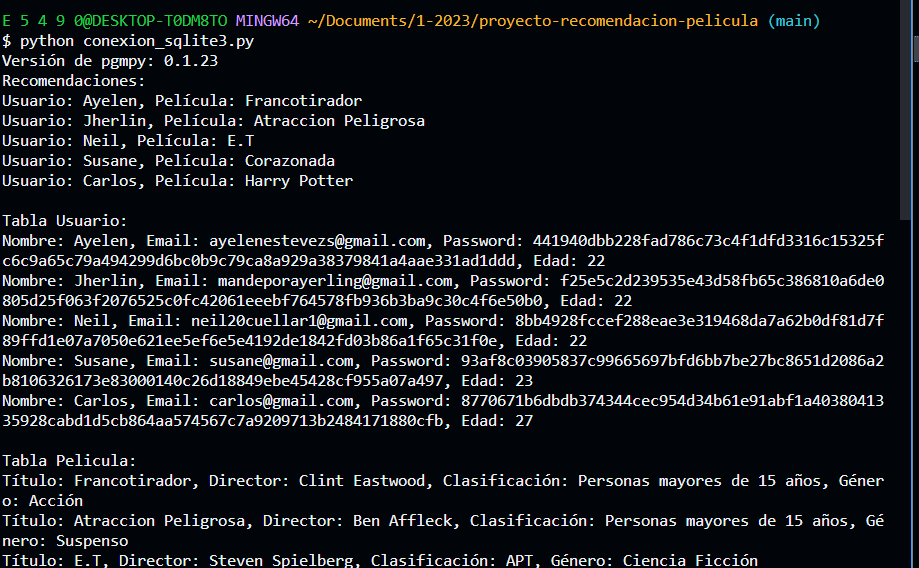
    print(f"Usuario: {calificacion[0]}, Película: {calificacion[1]}, Descripción: {calificacion[2]}")

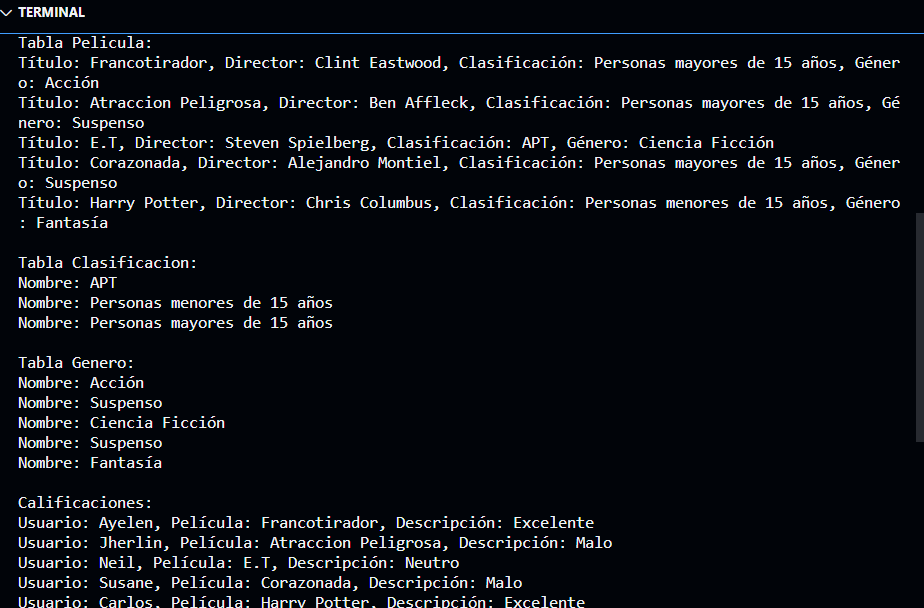
# Cerrar la conexión a la base de datos

close\_connection(conn)

Para visualizar estas consultas podemos utilizar el intérprete de Python

Comando: python conexion\_sqlite3.py





**CREACION DE EL ARCHIVO “MOTOR\_INFERENCIA.PY”**

Utilizaremos el modelo bayesiano para realizar el motor de inferencia del sistema experto de recomendación de películas.

En este archivo se trabajará con tareas o funcionalidades que el sistema experto de Recomendación de Películas realice cuando interactúe con el usuario, este archivo hará uso de la base de conocimiento con el modelo bayesiano.

Importamos las librerías correspondientes para utilizar la base de conocimiento.

from pgmpy.inference import VariableElimination

from base\_conocimiento import base\_conocimiento

inferencia = VariableElimination(base\_conocimiento)

#Aquí se añadirá mas funcionalidades

**CREACION DE EL ARCHIVO “INTERFAZ\_GRAFICA.PY”**

Aquí se trabajará la parte visual el Shell del sistema experto donde el usuario podrá interactuar de manera mas intuitiva con el sistema.

Haremos uso de la librería “tkinter” que sirve para la creación de interfaces en Python

import tkinter as tk

import sqlite3

# Función para iniciar sesión

def login():

    # Obtener los valores de correo electrónico y contraseña ingresados por el usuario

    email = email\_var.get()

    password = password\_var.get()

    # Realizar la lógica de autenticación y validación de datos

    conn = sqlite3.connect('database.db')

    cursor = conn.cursor()

    # Verificar si el correo electrónico y la contraseña coinciden en la base de datos

    cursor.execute('SELECT \* FROM Usuario WHERE email = ? AND password = ?', (email, password))

    user = cursor.fetchone()

    if user:

        # Cerrar la conexión a la base de datos

        conn.close()

        login\_window.destroy()  # Cerrar la ventana de inicio de sesión

        # Mostrar mensaje de éxito

        success\_label = tk.Label(login\_window, text="Inicio de sesión exitoso", fg="green")

        success\_label.place(relx=0.5, rely=0.9, anchor="center")

        # Aquí puedes continuar con la lógica de tu programa, como mostrar la ventana principal, realizar recomendaciones, etc.

        show\_main\_window()

    else:

        # Cerrar la conexión a la base de datos

        conn.close()

        # Mostrar mensaje de error

        error\_label = tk.Label(login\_window, text="Credenciales incorrectas", fg="red")

        error\_label.place(relx=0.5, rely=0.9, anchor="center")

    # Cerrar la ventana de inicio de sesión después de verificar las credenciales

    login\_window.after(2000, login\_window.destroy)

# Ventana de inicio de sesión

def show\_login\_window():

    global email\_var, password\_var, login\_window  # Definir las variables como globales

    login\_window = tk.Tk()

    # Configurar la ventana de inicio de sesión

    login\_window.title("Inicio de sesión")

    login\_window.configure(bg="lightgray")  # Establecer el color de fondo

    # Obtener el tamaño de la pantalla

    screen\_width = login\_window.winfo\_screenwidth()

    screen\_height = login\_window.winfo\_screenheight()

    # Calcular las coordenadas para centrar la ventana en la pantalla

    window\_width = 400

    window\_height = 300

    x = (screen\_width // 2) - (window\_width // 2)

    y = (screen\_height // 2) - (window\_height // 2)

    # Establecer la geometría de la ventana

    login\_window.geometry(f"{window\_width}x{window\_height}+{x}+{y}")

    # Variables de control para los campos de correo electrónico y contraseña

    email\_var = tk.StringVar()

    password\_var = tk.StringVar()

    # Crear etiqueta y campo de entrada para correo electrónico

    email\_label = tk.Label(login\_window, text="Correo electrónico:")

    email\_label.place(relx=0.5, rely=0.35, anchor="center")

    email\_entry = tk.Entry(login\_window, textvariable=email\_var)

    email\_entry.place(relx=0.5, rely=0.4, anchor="center")

    # Crear etiqueta y campo de entrada para contraseña

    password\_label = tk.Label(login\_window, text="Contraseña:")

    password\_label.place(relx=0.5, rely=0.55, anchor="center")

    password\_entry = tk.Entry(login\_window, show="\*", textvariable=password\_var)

    password\_entry.place(relx=0.5, rely=0.6, anchor="center")

    # Crear botón de inicio de sesión

    login\_button = tk.Button(login\_window, text="Iniciar sesión", command=login)

    login\_button.place(relx=0.5, rely=0.75, anchor="center")

    # Ejecutar el bucle principal de la ventana de inicio de sesión

    login\_window.mainloop()

    #############################################

# Ventana de registro de usuario

def show\_register\_window():

    register\_window = tk.Tk()

    # Configurar la ventana de registro de usuario

    # ...

    register\_window.mainloop()

# Ventana principal

def show\_main\_window():

    main\_window = tk.Tk()

    # Configurar la ventana principal

    # ...

    main\_window.mainloop()

# Función principal

def main():

    # Lógica principal del programa

    show\_login\_window()

# Punto de entrada del programa

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

    main()

Con esto creamos el login para que el usuario pueda autentificarse

Para ver la interfaz utilizamos el interprete de Python “python interfaz\_grafica.py”

