Como ejecutar script o proyecto 3

Este proyecto fue desarrollado en Python 3 utilizando el framework Django dentro de un entorno virtual (`venv`) y posteriormente subido a un repositorio en GitHub. Para ejecutarlo correctamente en otro equipo, se deben seguir los **siguientes pasos**:

1. Descarga o clonar el GitHub:

- Descarga o clona la carpeta “Página” de gitHub compartido , posteriormente se abrirá la carpeta “Página” en visual studio code

2. Crear y Activar el entorno virtual

Navega a la carpeta del proyecto(“Página”) en tu terminal (CMD, PowerShell o bash):

Ejecute el siguiente comando en la terminal para crear el entorno virtual

- `python -m venv venv`

Nota: en caso no tener venv instalarlo con pip install virtualenv

Posterior a la creación activa el entorno virtual

- `.venv\Scripts\activate` **ó** `venv\Scripts\activate`

3. Instalación de dependencias del proyecto

Dentro del entorno virtual (Terminal), asegúrate de que todas las dependencias requeridas estén instaladas. Para ello, ejecuta el siguiente comando en la terminal:

- `pip install -r requirements.txt`

Nota: El entorno debería contener todas las librerías necesarias preinstaladas con el comando -r requirements.txt, en caso de no tenerlas instalarlas.

4. Inicio del servidor

Ejecute el servidor de desarrollo de Django con el comando (en la terminal):

- `python manage.py runserver`

Nota: Tener cuidado con el puerto que no esté empleando en otra app o servicio (especificar el puerto en dado caso al lado de runserver)

5. Acceso a la aplicación

- En su navegador web, ingrese a la dirección mostrada en la terminal (normalmente http://127.0.0.1:8000/)
- Alternativamente, haga clic en el enlace del servidor que aparece en la terminal de VS Code

Descripción Técnica del proyecto 3 (Servicio Web)

Este proyecto ha sido desarrollado en **Python** 3.12.10 utilizando el **framework Django 5.1.4**, la cual permite construir aplicaciones web robustas de manera eficiente. El proyecto se ejecuta dentro de un **entorno virtual (venv)** para asegurar que todas las dependencias estén controladas y aisladas del sistema operativo. Además, se incorporó una funcionalidad de inteligencia artificial mediante un modelo previamente entrenado y almacenado en formato `.pkl` usando la librería **Joblib**, lo que permite integrar predicciones en tiempo real dentro de la aplicación web.

La estructura del proyecto es la estándar en Django, lo cual indica que se trata de una **aplicación monolítica** y no una API. Esto se deduce por la existencia de plantillas HTML que se renderizan directamente desde las vistas (por ejemplo, `index.html`), lo que es típico de aplicaciones con interfaz web tradicional (basadas en servidor), en lugar de un enfoque de API RESTful con respuestas JSON.

El proyecto se organiza de la siguiente manera:

- **proyecto_3/**: Es la carpeta raíz del proyecto.
    - **formulario/**: Aplicación registrada dentro del proyecto principal. Agrupa toda la lógica relacionada con los formularios y el proceso de

predicción.

- `views.py`: Contiene la lógica principal del servidor para procesar los formularios. En este archivo se cargó el modelo de machine learning usando la línea `pipeline, le = joblib.load(MODEL_PATH)`, lo que permite aplicar un modelo previamente entrenado directamente sobre los datos del usuario. Esta vista recibe los datos enviados por el formulario HTML, los transforma y aplica el modelo, luego envía el resultado de vuelta al usuario.

- **modelo/**: Carpeta que almacena el archivo del modelo de machine learning serializado en formato `.pkl` (por ejemplo, `modelo_catboost.pkl`). Este archivo fue generado a partir de un modelo entrenado con la librería CatBoost, y es utilizado en producción para obtener predicciones rápidas y precisas.

- `modelo_pred.py`: Script auxiliar que encapsula la lógica del modelo predictivo. Este archivo tiene dos funciones principales:

  predecir(data_dict): Recibe los datos del formulario como diccionario, los transforma en un DataFrame, aplica mapeos categóricos definidos manualmente y ejecuta la predicción mediante el modelo cargado.

  agrupar_codigo(codigo): Función adicional que permite mapear códigos CIE-10 a categorías clínicas generalizadas. Este procesamiento es útil para estandarizar entradas médicas en grupos significativos para el modelo.

  Este módulo o script actúa como una **interfaz de preprocesamiento y predicción** que permite mantener la lógica del modelo desacoplado de las vistas de Django.

- **templates/formulario/index.html**: Archivo HTML que define la interfaz de usuario donde se ingresan los datos necesarios para que el modelo haga la predicción.

- `urls.py`: Define las rutas específicas para esta aplicación. Se encarga de enrutar las peticiones HTTP a la vista correspondiente.

        - `models.py`, `admin.py`, `apps.py`, `tests.py`: Archivos creados por Django al iniciar la aplicación. En este proyecto no fueron modificados, por lo tanto mantienen su funcionalidad por defecto.

    - **proyect3/**: Es la configuración principal del proyecto Django.

        - `settings.py`: Archivo fundamental donde se definen parámetros como la configuración de aplicaciones instaladas (por ejemplo, `formulario`), las rutas de los archivos estáticos, el uso de la base de datos SQLite (`db.sqlite3`), y otros ajustes importantes.

        - `urls.py`: Es el archivo que define las rutas principales del proyecto, incluyendo la inclusión de rutas de la aplicación `formulario`.

        - `wsgi.py` y `asgi.py`: Archivos necesarios para el despliegue del proyecto en servidores compatibles con WSGI o ASGI.

- **venv/**: Carpeta que contiene el entorno virtual, con todas las dependencias necesarias del proyecto instaladas de manera aislada.

- `db.sqlite3`: Base de datos liviana SQLite creada automáticamente por Django para persistir los datos. En este proyecto, no se usó para almacenar predicciones, pero se mantiene como parte de la estructura.

Este proyecto combina herramientas de Django con técnicas de machine learning, permitiendo a los usuarios ingresar datos en una interfaz web sencilla y obtener resultados generados por un modelo previamente entrenado, todo esto de manera interactiva y en tiempo real. La modularidad del sistema permite mantener una estructura clara, reutilizable y escalable para futuras ampliaciones y mejoras de la misma.