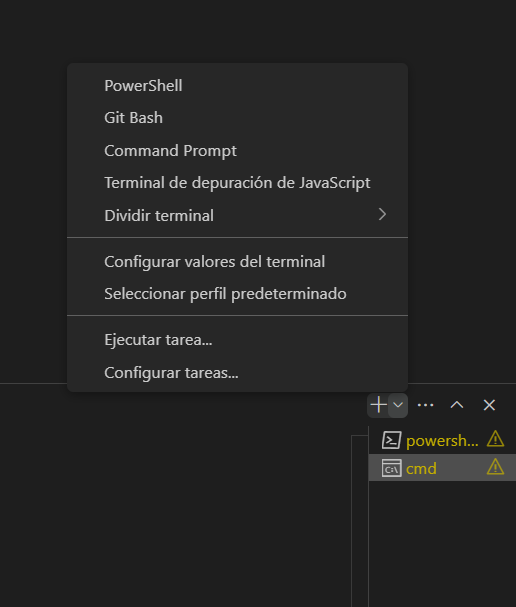
Creación de entorno virtual

Usar el comando python -m venv env esto creara el entorno dentro del proyecto, recomiendo usar la consola de command prompt en vs code



Después de ejecutar vera alcomo como esto

Texto

El contenido generado por IA puede ser incorrecto.

Y se creara una carpeta

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Hasta este punto se asume que previo a todo esto, tiene instalado Git en su comutador y tiene una cuenta en [www.github.com](http://www.github.com) donde debió crear un repositorio y debió haber clonado el mismo en su Pc, por ello mi ejemplo tiene un archivo que github crea y escoji que usara Python este es el archivo gitignore, este archivo evitara se se carguen cambios al repositorio productos de archivos de ejecución del proyecto, a lo que le llamaríamos archivos basura.

No se recomienda cargar el entorno virtual al repositorio por eso se recomienda agregar env dentro del archi gitignore



Cave resaltara que debe escribirlo tal cual y como creo su entorno virtual

Aquí dejare el contenido del git ignore:

# Byte-compiled / optimized / DLL files

\_\_pycache\_\_/

\*.py[codz]

\*$py.class

# C extensions

\*.so

#entorno virtual

env/

# Distribution / packaging

.Python

build/

develop-eggs/

dist/

downloads/

eggs/

.eggs/

lib/

lib64/

parts/

sdist/

var/

wheels/

share/python-wheels/

\*.egg-info/

.installed.cfg

\*.egg

MANIFEST

# PyInstaller

#  Usually these files are written by a python script from a template

#  before PyInstaller builds the exe, so as to inject date/other infos into it.

\*.manifest

\*.spec

# Installer logs

pip-log.txt

pip-delete-this-directory.txt

# Unit test / coverage reports

htmlcov/

.tox/

.nox/

.coverage

.coverage.\*

.cache

nosetests.xml

coverage.xml

\*.cover

\*.py.cover

.hypothesis/

.pytest\_cache/

cover/

# Translations

\*.mo

\*.pot

# Django stuff:

\*.log

local\_settings.py

db.sqlite3

db.sqlite3-journal

# Flask stuff:

instance/

.webassets-cache

# Scrapy stuff:

.scrapy

# Sphinx documentation

docs/\_build/

# PyBuilder

.pybuilder/

target/

# Jupyter Notebook

.ipynb\_checkpoints

# IPython

profile\_default/

ipython\_config.py

# pyenv

#   For a library or package, you might want to ignore these files since the code is

#   intended to run in multiple environments; otherwise, check them in:

# .python-version

# pipenv

#   According to pypa/pipenv#598, it is recommended to include Pipfile.lock in version control.

#   However, in case of collaboration, if having platform-specific dependencies or dependencies

#   having no cross-platform support, pipenv may install dependencies that don't work, or not

#   install all needed dependencies.

#Pipfile.lock

# UV

#   Similar to Pipfile.lock, it is generally recommended to include uv.lock in version control.

#   This is especially recommended for binary packages to ensure reproducibility, and is more

#   commonly ignored for libraries.

#uv.lock

# poetry

#   Similar to Pipfile.lock, it is generally recommended to include poetry.lock in version control.

#   This is especially recommended for binary packages to ensure reproducibility, and is more

#   commonly ignored for libraries.

#   https://python-poetry.org/docs/basic-usage/#commit-your-poetrylock-file-to-version-control

#poetry.lock

#poetry.toml

# pdm

#   Similar to Pipfile.lock, it is generally recommended to include pdm.lock in version control.

#   pdm recommends including project-wide configuration in pdm.toml, but excluding .pdm-python.

#   https://pdm-project.org/en/latest/usage/project/#working-with-version-control

#pdm.lock

#pdm.toml

.pdm-python

.pdm-build/

# pixi

#   Similar to Pipfile.lock, it is generally recommended to include pixi.lock in version control.

#pixi.lock

#   Pixi creates a virtual environment in the .pixi directory, just like venv module creates one

#   in the .venv directory. It is recommended not to include this directory in version control.

.pixi

# PEP 582; used by e.g. github.com/David-OConnor/pyflow and github.com/pdm-project/pdm

\_\_pypackages\_\_/

# Celery stuff

celerybeat-schedule

celerybeat.pid

# SageMath parsed files

\*.sage.py

# Environments

.env

.envrc

.venv

env/

venv/

ENV/

env.bak/

venv.bak/

# Spyder project settings

.spyderproject

.spyproject

# Rope project settings

.ropeproject

# mkdocs documentation

/site

# mypy

.mypy\_cache/

.dmypy.json

dmypy.json

# Pyre type checker

.pyre/

# pytype static type analyzer

.pytype/

# Cython debug symbols

cython\_debug/

# PyCharm

#  JetBrains specific template is maintained in a separate JetBrains.gitignore that can

#  be found at https://github.com/github/gitignore/blob/main/Global/JetBrains.gitignore

#  and can be added to the global gitignore or merged into this file.  For a more nuclear

#  option (not recommended) you can uncomment the following to ignore the entire idea folder.

#.idea/

# Abstra

# Abstra is an AI-powered process automation framework.

# Ignore directories containing user credentials, local state, and settings.

# Learn more at https://abstra.io/docs

.abstra/

# Visual Studio Code

#  Visual Studio Code specific template is maintained in a separate VisualStudioCode.gitignore

#  that can be found at https://github.com/github/gitignore/blob/main/Global/VisualStudioCode.gitignore

#  and can be added to the global gitignore or merged into this file. However, if you prefer,

#  you could uncomment the following to ignore the entire vscode folder

# .vscode/

# Ruff stuff:

.ruff\_cache/

# PyPI configuration file

.pypirc

# Cursor

#  Cursor is an AI-powered code editor. `.cursorignore` specifies files/directories to

#  exclude from AI features like autocomplete and code analysis. Recommended for sensitive data

#  refer to https://docs.cursor.com/context/ignore-files

.cursorignore

.cursorindexingignore

# Marimo

marimo/\_static/

marimo/\_lsp/

\_\_marimo\_\_/

Si todo ha salido bien, entonces la próxima vez que hagas un push en git no se cargara ningún archivo o extensión relacionada en el gitignore.

Interfaz de usuario gráfica, Texto

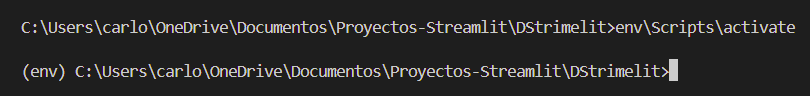
El contenido generado por IA puede ser incorrecto.

Lo sabaras porque en git las carpetas o archivos nuevos se visualizan con color verde y los archivos modificados con amarillo, pero los ignorados salen con letras grises.

INSTALACIÓN DE STREAMLIT

Debe primero activar el entorno virtual , puede usar el comando : env\Scripts\activate

Sabrá que esta activo porque aparece el nombre del entorno en paréntesis (env)



Primero revisemos que dependencias tenemos instaladas

Con el entorno activo ejecuta: pip list

Texto

El contenido generado por IA puede ser incorrecto.

En mi caso tengo la librería de pip y setuptools. Normalmente solo sale pip

Ahora si instalaremos la librería de Streamli

Ejecutar en la consola de comandos pip install streamlit

Para ello asegurece de estar dentro de la carpeta Scripts de su entorno virtual, esta es la forma correcta de instalar dependencias y librerias



Se cargara muchas cosas relacionadas a la libreria

Texto

El contenido generado por IA puede ser incorrecto.

Al finalizar puee hacer de nuevo : pip list y vera las nuevas dependencias que se intalaron con base en el comando

En mi caso esta es un vista de algo del contenido, puede observar que streamlit esta ahora instalado

Texto

El contenido generado por IA puede ser incorrecto.

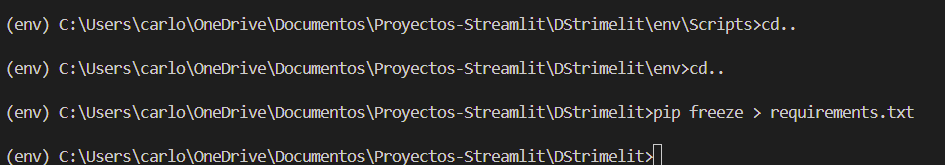
Okay es importante aclarar qwue estas dependencias estarán instaladas en el entorno virtual para evitar instalarlas en nuetro pc, pro como el entorno virtual no lo cargamos a github es importante crear el archivo requirements.txt

Creando Archivo requirements.txt

Para que sirve?

sirve para **guardar la lista de librerías que usa tu proyecto**, con sus versiones exactas, para que cualquier persona (o tú mismo en otra computadora) pueda **instalarlas rápido y sin errores** usando un solo comando (pip install -r requirements.txt), asegurando que el proyecto funcione igual en cualquier entorno.

Antes de ejecutar el comando en la consola debe salir de la carpeta Scrips y env. Asegurarse de que se encuentra en la carpeta principal de su proyecto y allí ejecutar el comando



Comando : pip freeze > requirements.txt

Si no sabe como salir de las carpetas usa cd..

Una vez ejecutado se crear un archivo llamado requirements.txt, si no aparece es posible que no haya salido de Scripts y dentro del entorno virtual lo haya creado por error, es mejor buscarlos manualmente y borrarlo, segurarse como en la imagen de arriba que esta en la carpeta principal y ejecutar el comando

Captura de pantalla de un teléfono celular

El contenido generado por IA puede ser incorrecto.

De esta forma debe verse y si abre el archivo puede ver que están todas las librerías

Pantalla de computadora con letras

El contenido generado por IA puede ser incorrecto.

No se asuste si no ve los mismos nombres suelen variar según sus versiones

ACTUALIZAR REQUIREMENTS.TXT

Básicamente es ejecutar en la carpte aprincipal de nuevo: pip freeze > requirements.txt  
esto reemplazara el archivo anterior

INSTALAR REQUIREMENTS. TEXT DE OTROS PROYECTOS CLONADOS

Opcional- Solo en caso de clonar proyectos

Ejecutas el comando : pip install -r requirements.txt

Pero debes tener el entorno virtual activo