

Entornos virtuales y UV

Máster en IA, Cloud Computing y DevOps
Módulo: Machine Learning y Deep Learning

Contenido

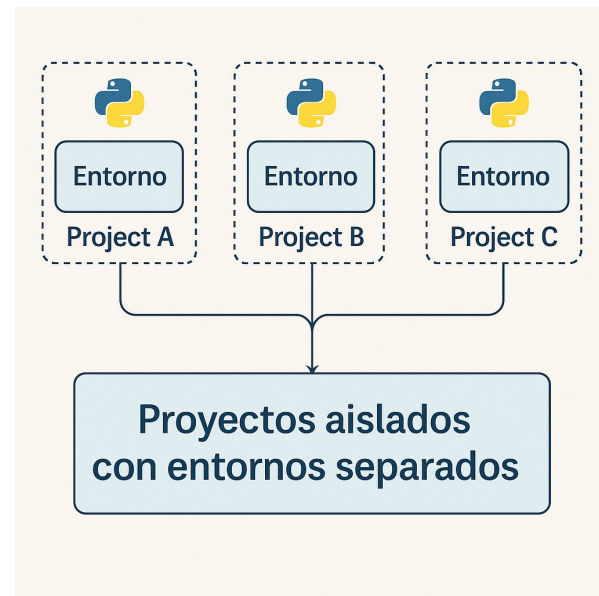
- 01. Entornos virtuales y su importancia
- 02. Herramientas de gestión de entornos y UV
- 03. Cómo usar UV y su configuración
- 04. Instalación entorno asignatura

01. Qué es un entorno virtual

Entornos virtuales y su importancia

Un entorno virtual es un espacio aislado que permite mantener dependencias y paquetes de software separados para cada proyecto en Python. Esto ayuda a evitar conflictos entre versiones de bibliotecas y dependencias, asegurando que cada proyecto funcione con la configuración específica requerida.

- Evita conflictos entre dependencias que requieren versiones distintas.
- Permite reproducir fácilmente un entorno exacto, facilitando la colaboración y el despliegue.
- Previene errores inesperados por actualizaciones incompatibles de paquetes.

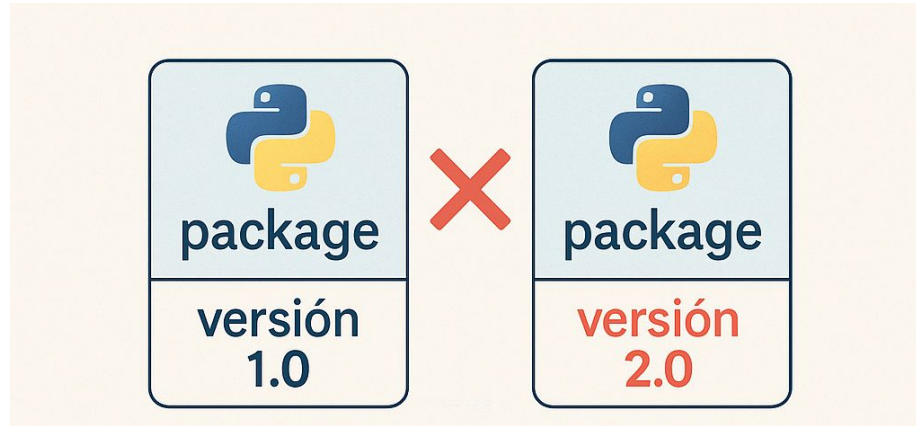


01. Importancia de la Gestión de Versiones

Entornos virtuales y su importancia

Gestionar versiones es fundamental en el desarrollo de software porque permiten aislar las dependencias específicas de cada proyecto, manteniendo cada entorno independiente. De esta manera se obtiene:

- Aislamiento de dependencias específicas
- Independencia de proyectos
- Simplificación del desarrollo colaborativo



02. Entonces... ¿qué herramientas tenemos para poder trabajar?

Herramientas de gestión de entornos y UV

| Característica | pip + venv | conda | Poetry | UV |
|-----------------------------------|---|--|---|----------------------------------|
| Velocidad | Moderada | Moderada | Alta | Muy alta |
| Facilidad de uso | Alta | Moderada | Alta | Alta |
| Compatibilidad | Python | Multi-lenguaje (Python, R, etc) | Python | Python |
| Gestión de dependencias | Manual (requirements.txt) | Automática | Automática | Manual optimizada |
| Resolución de dependencias | Limitada | Avanzada | Avanzada | Rápida, optimizada |
| Consumo de recursos | Bajo | Alto | Moderado | Bajo |
| Uso recomendado | Proyectos simples, pequeños | Ciencia de datos, proyectos complejos | Proyectos estructurados, bibliotecas | Proyectos ágiles y eficientes |
| Integración con pip | Nativa | Compatible (parcial) | Compatible | Nativa y optimizada |

02. ¿Qué es UV?

Herramientas de gestión de entornos y UV

UV (Universal Virtualenv) es una herramienta moderna y ultrarrápida escrita en Rust que permite gestionar entornos virtuales y dependencias de proyectos Python de forma eficaz. En ciencia de datos, donde reproducibilidad, rendimiento y simplicidad son claves, es una excelente alternativa a otros frameworks como *pip*, *virtualenv*, *conda* o *poetry*.

Algunas de sus ventajas:

- Ultra rápido gracias a su implementación en Rust
- Compatible con *pip* y *pyproject.toml*
- Gestiona entornos virtuales integrados
- Resolución de dependencias precisa y eficiente
- Funciona en Linux, macOS y Windows

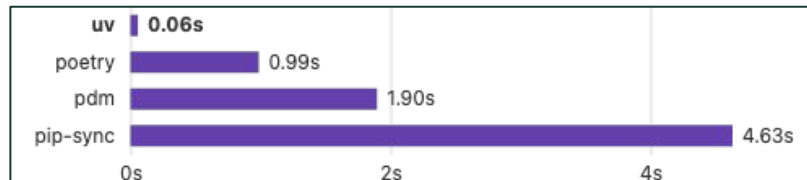


Gráfico de un benchmark comparando diferentes frameworks instalando la librería Trio
Fuente: documentación oficial uv

03. Prerrequisitos a la instalación de UV

Cómo usar UV y su configuración

Antes de instalar UV, es necesario contar con Python correctamente instalado, en la asignatura se usará la versión 3.11.

A continuación se muestran brevemente los pasos de instalación para los diferentes sistemas operativos:



1. Descargar desde:
<https://www.python.org/downloads/windows/>
2. Ejecutar el instalador y marcar la opción "Add Python to PATH".
3. Completar la instalación con configuración por defecto.



1. Descargar desde:
<https://www.python.org/downloads/mac/>
2. Ejecutar el .pkg descargado y seguir los pasos.
3. Verificar en terminal: `python3 --version`



1. Abrir una terminal.
2. Ejecutar los siguientes comandos:
 - a. `sudo apt update`
 - b. `sudo apt install python3.11 python3.11-venv python3.11-dev`

03. Instalación de UV

Cómo usar UV y su configuración

Como hemos visto en la diapositiva anterior, será necesario tener Python instalado y actualizado y además también *pip*.



1. Abre una terminal.
2. Usa el siguiente comando:

```
pip install uv
```
3. En caso de que prefieras puedes usar comandos de *Powershell*, para ello utiliza el siguiente:

```
powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"
```
4. Para comprobar que la instalación funcionó con éxito, utiliza el siguiente comando para ver la versión de *uv*:

```
uv --version
```



1. Abre una terminal.
2. Usa el siguiente comando:

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```
3. Reinicia la terminal o ejecuta el comando:

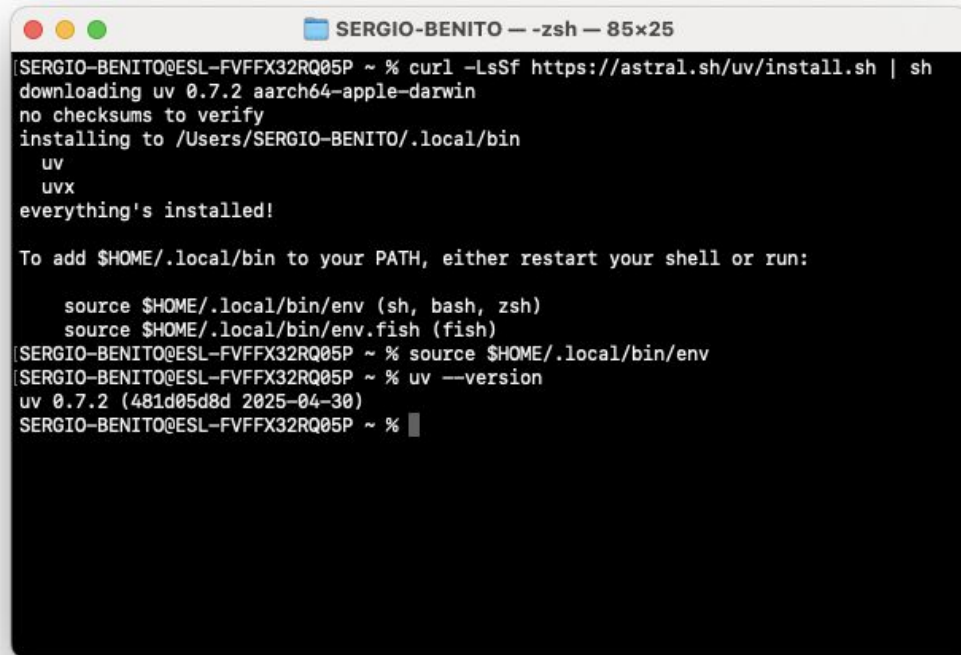
```
source $HOME/.local/bin/env
```
4. Para comprobar que la instalación funcionó con éxito, utiliza el siguiente comando para ver la versión de *uv*:

```
uv --version
```
5. También es posible utilizar *Homebrew* o el *pip*:

```
brew install uv
```


03. Instalación de UV

Cómo usar UV y su configuración



```

SERGIO-BENITO — -zsh — 85x25
SERGIO-BENITO@ESL-FVFFX32RQ05P ~ % curl -Lsf https://astral.sh/uv/install.sh | sh
downloading uv 0.7.2 aarch64-apple-darwin
no checksums to verify
installing to /Users/SERGIO-BENITO/.local/bin
  uv
  uvx
everything's installed!

To add $HOME/.local/bin to your PATH, either restart your shell or run:

    source $HOME/.local/bin/env (sh, bash, zsh)
    source $HOME/.local/bin/env.fish (fish)
SERGIO-BENITO@ESL-FVFFX32RQ05P ~ % source $HOME/.local/bin/env
SERGIO-BENITO@ESL-FVFFX32RQ05P ~ % uv --version
uv 0.7.2 (481d05d8d 2025-04-30)
SERGIO-BENITO@ESL-FVFFX32RQ05P ~ %
```

03. Muestra las opciones con `uv --help`

Cómo usar UV y su configuración

```

SERGIO-BENITO@ESL-FVFFX32RQ05P ~ % uv --help
An extremely fast Python package manager.

Usage: uv [OPTIONS] <COMMAND>

Commands:
  run      Run a command or script
  init     Create a new project
  add      Add dependencies to the project
  remove   Remove dependencies from the project
  sync     Update the project's environment
  lock     Update the project's lockfile
  export   Export the project's lockfile to an alternate format
  tree     Display the project's dependency tree
  tool     Run and install commands provided by Python packages
  python   Manage Python versions and installations
  pip      Manage Python packages with a pip-compatible interface
  venv     Create a virtual environment
  build    Build Python packages into source distributions and wheels
  publish  Upload distributions to an index
  cache    Manage uv's cache
  self     Manage the uv executable
  version  Read or update the project's version
  help     Display documentation for a command

Cache options:
  -n, --no-cache          Avoid reading from or writing to the cache, instead using a temporary directory for the duration of the operation [env: UV_NO_CACHE=]
  --cache-dir <CACHE_DIR> Path to the cache directory [env: UV_CACHE_DIR=]

Python options:
  --managed-python      Require use of uv-managed Python versions [env: UV_MANAGED_PYTHON=]
  --no-managed-python   Disable use of uv-managed Python versions [env: UV_NO_MANAGED_PYTHON=]
  --no-python-downloads Disable automatic downloads of Python. [env: "UV_PYTHON_DOWNLOADS=never"]

Global options:
  -q, --quiet...          Use quiet output
  -v, --verbose...        Use verbose output
  --color <COLOR_CHOICE> Control the use of color in output [possible values: auto, always, never]
  --native-tls            Whether to load TLS certificates from the platform's native certificate store [env: UV_NATIVE_TLS=]
  --offline              Disable network access [env: UV_OFFLINE=]
  --allow-insecure-host <ALLOW_INSECURE_HOST> Allow insecure connections to a host [env: UV_INSECURE_HOST=]
  --no-progress          Hide all progress outputs [env: UV_NO_PROGRESS=]
  --directory <DIRECTORY> Change to the given directory prior to running the command
  --project <PROJECT>     Run the command within the given project directory [env: UV_PROJECT=]
  --config-file <CONFIG_FILE> The path to a 'uv.toml' file to use for configuration [env: UV_CONFIG_FILE=]
  --no-config            Avoid discovering configuration files ('pyproject.toml', 'uv.toml') [env: UV_NO_CONFIG=]
  -h, --help             Display the concise help for this command
  -V, --version          Display the uv version

Use 'uv help' for more details.
SERGIO-BENITO@ESL-FVFFX32RQ05P ~ %
```

03. Algunos comandos básicos

Cómo usar UV y su configuración

Acción



Ubuntu



Crear entorno virtual

```
uv venv
```

Activar entorno virtual

```
.\.venv\Scripts\activate    source .venv/bin/activate
```

Desactivar entorno virtual

```
deactivate
```

Instalar paquetes

```
uv pip install nombre_paquete
```

Listar paquetes instalados

```
uv pip freeze
```

Actualizar paquetes

```
uv pip upgrade
```

04. Crea un entorno virtual vacío

Instalación del entorno de la asignatura



1. Abre una terminal
2. Ve al directorio donde quieres guardar los archivos de la asignatura
`cd ~/ruta/a/tu/directorio`
3. Crea un entorno virtual con `uv`, esto creará un entorno virtual en la carpeta oculta `.venv` en tu directorio actual. Los parámetros usados son los siguientes:
 - a. `venv`: indica que se desea crear un entorno virtual.
 - b. `--python 3.11`: para que la versión de Python utilizada sea la 3.11, se puede utilizar otra.
 - c. `env-pontia-ml`: sería el nombre que tendrá tu entorno virtual, puedes indicar el que desees.

```
uv venv --python 3.11 env-pontia-ml
```

4. Utilizar el comando `ls` para comprobar que se ha creado una carpeta con el nombre que le has dado al entorno virtual.
5. Activa tu entorno virtual, verás que a la izquierda de tu prompt se refleja entre paréntesis el nombre que le has dado:

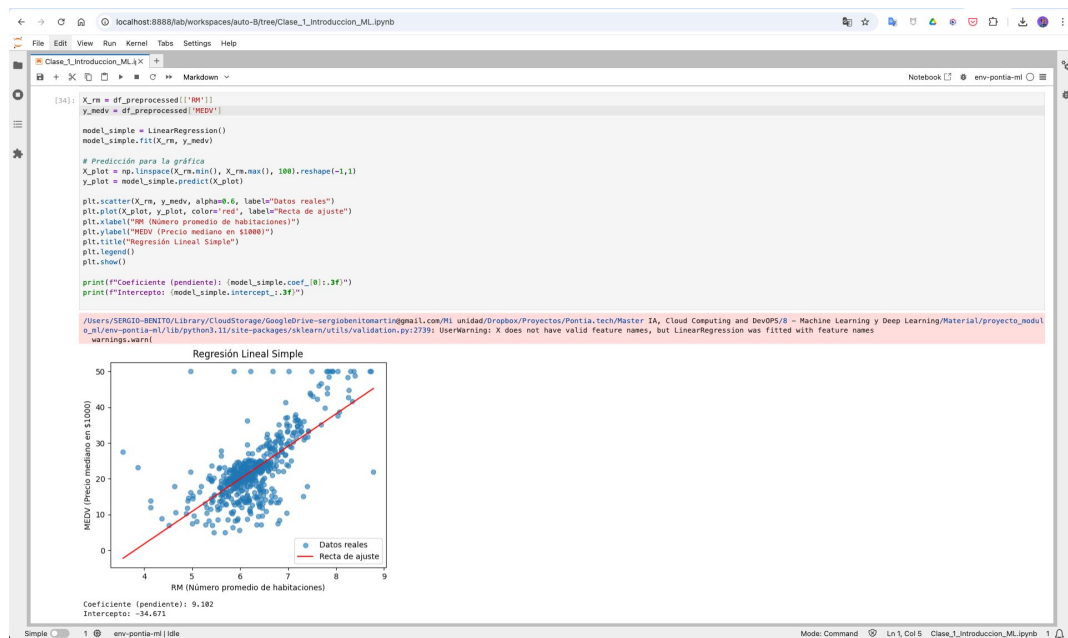
```
source env-pontia-ml/bin/activate
```

```
ejemplo_pontia — zsh — 107x8
SERGIO-BENITO@ESL-FVFFX32RQ05P ejemplo_pontia % uv venv --python 3.11 env-pontia-ml
Using CPython 3.11.12
Creating virtual environment at: env-pontia-ml
Activate with: source env-pontia-ml/bin/activate
SERGIO-BENITO@ESL-FVFFX32RQ05P ejemplo_pontia % ls
env-pontia-ml  requirements.txt
SERGIO-BENITO@ESL-FVFFX32RQ05P ejemplo_pontia % source env-pontia-ml/bin/activate
(env-pontia-ml) SERGIO-BENITO@ESL-FVFFX32RQ05P ejemplo_pontia %
```

04. Qué es Jupyter

Instalación del entorno de la asignatura

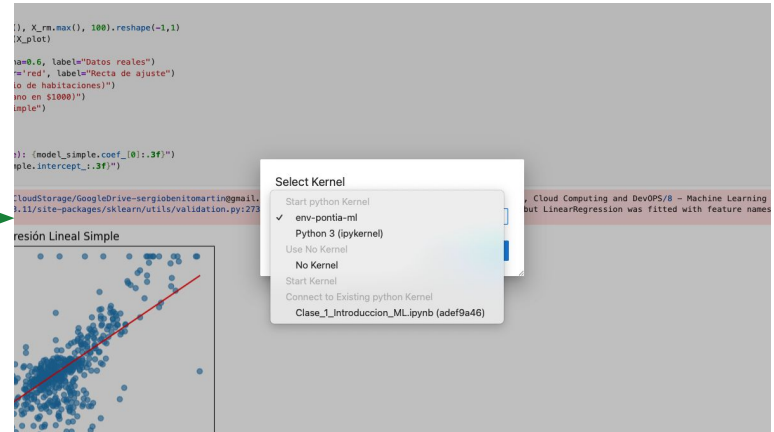
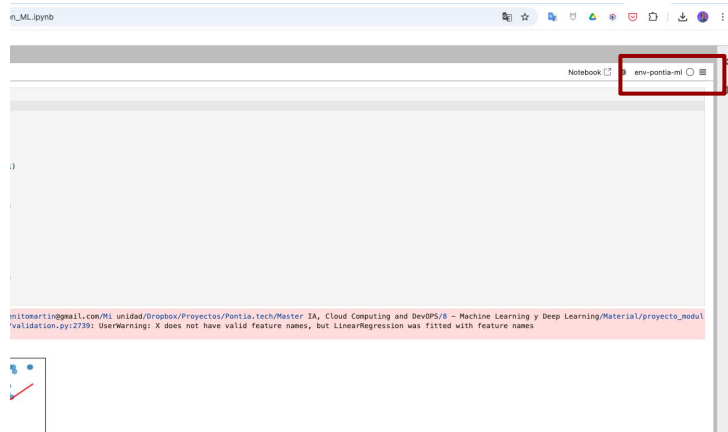
Jupyter es un entorno de desarrollo interactivo de código abierto, especialmente relevante dentro del área de ciencia de datos y del Machine Learning, por su forma en la que permite escribir código mediante celdas, lo que ayuda a ver de manera sencilla el resultado de cada ejecución.



04. Qué son los Kernel de Jupyter

Instalación del entorno de la asignatura

Para poder trabajar de manera diferenciada y aislada entre proyectos, Jupyter integra lo que se conoce como Kernel. Básicamente es el proceso que ejecuta un código de Python, cada uno de ellos puede utilizar un entorno virtual diferente, de este modo las librerías que se utilicen son las específicas que se han definido previamente.



Gracias