Clase 001 — Instalación de Python 3.12+ y entornos virtuales (venv, uv, conda)

Parte: 0 — Prerrequisitos · Fuente: VanderPlas, Python Data Science Handbook, prefacio "Installation Considerations". · Duración estimada: 90 min (45 lectura + 45 práctica).

Clase 001 — Instalación de Python 3.12+ y entornos virtuales (venv, uv, conda)

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, Python Data Science Handbook, prefacio "Installation Considerations". · Duración estimada: 90 min (45 lectura + 45 práctica).*

Objetivo

Que el alumno deje su máquina lista para trabajar en data science: con Python 3.12+ instalado correctamente, con al menos dos gestores de entornos virtuales funcionando (venv nativo + uv o conda), y con la disciplina de nunca instalar paquetes en el Python del sistema. Al final de la clase, deberá poder crear un entorno limpio, activarlo, instalar dependencias declaradas, y reproducirlo exactamente en otra máquina.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Verificar qué versión de Python tiene su máquina y desde qué ruta se ejecuta (python -V, which python / where python, sys.executable).
2. Crear, activar y destruir entornos virtuales con venv, uv venv y conda create — y explicar cuándo conviene cada uno.
3. Instalar dependencias desde requirements.txt y desde pyproject.toml, y congelar versiones reproducibles (pip freeze, uv pip compile, conda env export).
4. Diagnosticar el error más frecuente del principiante: "instalé un paquete pero el import falla" (causa: pip instaló en un Python distinto del que ejecuta el notebook).
5. Justificar por qué un entorno virtual por proyecto es no-negociable en data science (reproducibilidad, conflictos de versiones, aislamiento de experimentos).

Prerrequisitos

- Acceso a una terminal (PowerShell en Windows, Terminal en macOS/Linux).
- Permisos para instalar software en la máquina propia.
- Conocimiento previo: ninguno. Esta es la primera clase del programa.

Temas

#	Tema	Por qué importa
1	Qué es Python "del sistema" y por qué no t	Romperlo deja tu OS inestable (macOS/Linux
2	Instaladores oficiales vs. pyenv / mise	Cómo tener varias versiones de Python conv
3	venv (stdlib)	El más simple, viene incluido — el "defaul
4	uv (Astral)	Rápido (10–100× pip), reemplazo moderno de
5	conda / mamba	Necesario cuando hay dependencias C/CUDA (
6	requirements.txt vs pyproject.toml vs envi	Tres formatos, tres ecosistemas — cuándo u
7	El bug más común: "pip install funciona pe	Diagnóstico con sys.executable y pip -V.

Dataset / recursos

No requiere dataset. Es una clase de setup. Los únicos archivos generados son los propios entornos virtuales (que se ignoran en git vía .gitignore).

Recursos externos:

- Python.org downloads — instalador oficial.
- uv docs — gestor moderno de Astral.
- Miniconda — instalación mínima de conda (evita Anaconda completo, pesa 3 GB).

Ejercicios

1. Diagnóstico inicial. Abre una terminal y reporta: versión de Python (python -V), ruta absoluta (where python en Windows, which python en Unix) y el contenido de sys.path ejecutando un script. Anótalo — lo usarás de baseline.
2. Crea un entorno venv llamado .venv en un directorio nuevo, actívalo, instala numpy==2.1.0 y pandas==2.2.3, y verifica con pip list. Luego desactívalo y comprueba que numpy ya no se importa desde el Python global.
3. Replica el mismo entorno con uv. Instala uv (pipx install uv o el instalador oficial), corre uv venv, uv pip install numpy pandas, y compara la velocidad contra el paso anterior.
4. Genera requirements.txt congelando versiones exactas con pip freeze > requirements.txt. Borra el entorno, recréalo desde cero y reinstala con pip install -r requirements.txt. Verifica que las versiones coinciden.
5. Provoca y resuelve el bug clásico. Desde Jupyter (ejecutando con un Python distinto al del venv activo), corre !pip install seaborn y luego import seaborn. Diagnostica por qué falla (o por qué se instaló en el lugar equivocado) usando import sys; sys.executable en una celda.

Homework verificable

Entrega un repo (o carpeta zip) con:

- [ ] README.md indicando tu OS, versión de Python instalada y gestor elegido (venv / uv / conda).
- [ ] .gitignore con .venv/ y __pycache__/ (mínimo).
- [ ] requirements.txt con exactamente: numpy>=2.0, pandas>=2.2, matplotlib>=3.8, jupyter>=1.0.
- [ ] Un script verify.py que imprima:
- sys.version
- sys.executable
- numpy.__version__, pandas.__version__
- [ ] Output de python verify.py pegado al final del README.md.

Criterio de aceptación: otra persona debe poder clonar tu repo, ejecutar:

```
python -m venv .venv
source .venv/bin/activate    # o .venv\Scripts\activate en Windows
pip install -r requirements.txt
python verify.py
```

…y obtener un output similar al que tú reportaste (mismo Python mayor.menor, mismos paquetes importables).

Definiciones y características

Python "del sistema"

: Intérprete instalado por el OS (macOS y Linux lo traen por default; Windows no). Lo usan herramientas internas del sistema — modificarlo puede romper el OS. Regla: nunca instales paquetes ahí.

Entorno virtual (venv)

: Carpeta autocontenida con su propio Python ejecutable y sus propios paquetes. Aislada del Python global y de otros venvs. Crear/destruir es barato — uno por proyecto es la norma.

pip

: Gestor de paquetes oficial de Python. Instala desde PyPI (Python Package Index). Característica clave: usa el Python con el que se invoca — siempre prefiere python -m pip install sobre pip install solo.

uv (Astral)

: Reemplazo moderno (2024+) escrito en Rust. Drop-in compatible con pip+venv pero 10–100× más rápido. Maneja también versiones de Python (reemplazo de pyenv). API: uv venv, uv pip install, uv python install 3.12.

conda / mamba

: Gestor multilenguaje (no solo Python). Maneja también dependencias binarias C/C++/CUDA (PyTorch+GPU, geopandas, rdkit). Más pesado pero imprescindible para esos casos. mamba es conda en C++ (más rápido).

requirements.txt vs pyproject.toml vs environment.yml

: Tres formatos, tres ecosistemas. requirements.txt (pip, lockfile). pyproject.toml (estándar moderno PEP 621, declarativo). environment.yml (conda).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
ModuleNotFoundError aunque acabo de instal	pip y python apuntan a Pythons distintos.
Permission denied al hacer pip install	Estás instalando en el Python del sistema
.venv\Scripts\Activate.ps1 cannot be loade	Execution policy bloquea scripts. Fix: Set
Pip muy lento descargando	Red lenta o servidor PyPI lejano. Fix: pru
Borré la carpeta .venv/ y pip list sigue m	Tu shell aún tiene el PATH del venv viejo.
conda activate no funciona en script no-in	conda init requiere shell interactivo. Fix

Preguntas frecuentes

¿venv, uv o conda — cuál uso?

venv si quieres simplicidad y solo tocas Python puro (stdlib + pandas + numpy + sklearn). uv si haces lo mismo pero quieres velocidad — recomendado en 2026. conda solo si necesitas dependencias C/CUDA (PyTorch con GPU, geopandas, química).

¿Puedo tener venv y conda en la misma máquina?

Sí — coexisten. Lo único: no actives ambos a la vez. Tu PATH se confunde y python apunta a uno u otro impredeciblemente.

¿Por qué .venv/ en el repo está en .gitignore?

Porque pesa cientos de MB y es reconstruible desde requirements.txt/pyproject.toml. Versionar el venv duplica el repo y crea conflicto si cambias de OS.

¿Debo actualizar pip al crear un venv?

Sí, primer paso: python -m pip install --upgrade pip. El pip dentro de venvs viejos tiene bugs conocidos y es lento.

¿Cómo sé en qué venv estoy?

python -c "import sys; print(sys.executable)" o which python (Unix) / where python (Windows). El prompt suele cambiar ((.venv) C:\...>) pero no siempre — confirma con el comando.

¿Necesito Python 3.12 específicamente?

Para este curso, sí (varias features usadas dependen). En general, usa la última estable. Python 3.10 ya pierde soporte en oct/2026.

Referencias

- VanderPlas, Python Data Science Handbook, Preface § "Installation Considerations".
- PEP 405 — venv (especificación oficial).
- Astral uv — Getting started.
- Conda vs pip vs venv — comparación oficial.

Siguiente clase

Clase 002 — Jupyter y JupyterLab — kernels, magics, debugging, profiling

Apéndice: notebook (primer bloque)

Imports mínimos: sólo stdlib. Esta clase no requiere paquetes externos — el punto es justamente aprender a instalarlos.

```
import sys
import platform
import shutil
from pathlib import Path

print('python      :', sys.version.split()[0])
print('executable  :', sys.executable)
print('platform    :', platform.system(), platform.release())
print('cwd         :', Path.cwd())
```

Archivos complementarios

- notebook.ipynb