Parte 0 — Prerrequisitos: Python + NumPy + pandas + visualización + SQL + APIs

49 clases · Parte 0 del programa

Parte 0 — Prerrequisitos: Python + NumPy + pandas + visualización + SQL + APIs

> *49 clases · bundle consolidado del currículo v3.*

Índice de clases

- Clase 040 — Clase 040 — Matplotlib: 3D plotting
- Clase 041 — Clase 041 — Seaborn: distribuciones, relaciones, categóricas, facetas
- Clase 042 — Clase 042 — Visualización geográfica (Plotly / folium)
- Clase 043 — Clase 043 — SQL fundamental: SELECT, WHERE, JOIN, GROUP BY, HAVING
- Clase 044 — Clase 044 — SQL avanzado: CTEs, window functions, subqueries correlacionadas
- Clase 045 — Clase 045 — SQL desde Python: sqlite3, SQLAlchemy, DuckDB
- Clase 046 — Clase 046 — NoSQL: MongoDB con pymongo
- Clase 047 — Clase 047 — APIs REST con requests
- Clase 048 — Clase 048 — Web scraping con BeautifulSoup
- Clase 049 — Clase 049 — async / httpx / aiohttp para data scientists

---

Clase 001 — 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.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

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

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, Python Data Science Handbook, cap. 1 — IPython: Beyond Normal Python.*

> *Duración estimada: 90 min.*

Objetivo

Que el alumno deje de usar Jupyter como un editor de texto con botón "play" y empiece a usarlo como un entorno exploratorio profesional: con magics que ahorran horas, debugger interactivo (%debug), y profiling real (%timeit, %prun). Al final debe poder diagnosticar por qué un notebook es lento sin adivinar.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Diferenciar kernel, frontend (Notebook vs JupyterLab vs VS Code) y servidor — y saber qué pasa cuando uno se cuelga.
2. Usar magics esenciales: %timeit, %%time, %run, %load, %matplotlib inline, %debug, %who, %xmode.
3. Conectar un kernel específico a un notebook (ipykernel install --user --name <env>) sin pelearse con el venv equivocado.
4. Debuggear una excepción con %debug y pdb (n, s, c, q, p, l).
5. Profilar código lento con %timeit (microbenchmark) y %prun (line profiler) para decidir dónde optimizar.

Temas

#	Tema	Por qué importa
1	Kernel ↔ frontend ↔ servidor	Saber cuál murió cuando el notebook se cue
2	Modo comando vs modo edición + atajos	Velocidad real: A/B/X/M/Y/Esc/Enter.
3	Magics line (%) vs cell (%%)	El 80% del valor de Jupyter está en las ma
4	%timeit y %%time	Microbenchmark riguroso (varias corridas,
5	%debug + pdb	Inspección post-mortem sin re-correr todo
6	%prun y %lprun	Saber qué función pesa antes de optimizar.
7	Registro de kernels por venv	Cada proyecto, su propio kernel — evita el

Definiciones y características

Kernel

: Proceso Python (u otro lenguaje) que ejecuta el código de las celdas. Vive separado del frontend; si lo matas, pierdes el estado en memoria pero los archivos siguen intactos. Cada notebook se asocia a UN kernel, normalmente el del venv del proyecto.

Frontend

: La interfaz visual (Notebook clásico, JupyterLab, VS Code, Cursor, Colab). Todas hablan el mismo protocolo con el kernel — puedes cambiar de frontend sin perder datos si guardas el .ipynb.

Magic

: Comando especial de IPython, no de Python. Empieza con % (afecta una línea) o %% (afecta la celda entera). Ejemplos: %timeit, %matplotlib inline, %%time, %debug. No funcionan fuera de IPython/Jupyter.

%timeit vs %%time

: %timeit corre la expresión muchas veces, descarta outliers y reporta el mejor → microbenchmark estadísticamente serio. %%time mide una sola corrida del bloque → bueno para operaciones largas donde repetir cuesta. Característica clave: usa %timeit para algo en milisegundos, %%time para algo en segundos.

pdb / %debug

: Debugger interactivo de Python. %debug lo lanza en modo post-mortem después de una excepción — entras al stack en el punto del error sin re-correr nada. Comandos: n siguiente línea, s entra a función, c continúa, p var imprime, u/d sube/baja en stack, q salir.

Dataset / recursos

No requiere dataset externo. Usamos arreglos sintéticos con numpy.random para benchmarks. Para el

ejercicio de debug, generamos un ValueError intencional.

Ejercicios

1. Atajos sin mouse. Crea 5 celdas, navega solo con teclado: convierte 2 a markdown, ejecuta todo en orden, borra una, deshaz. Cronométrate.

2. Registra tu kernel. Desde un venv recién creado: python -m ipykernel install --user --name ds-lab-001 --display-name 'DS Lab 001'. Abre Jupyter, selecciona ese kernel, verifica con import sys; sys.executable.

3. Benchmark vectorización. Con %timeit, compara sumar range(10_000) con un for vs np.arange(10_000).sum(). Anota cuántas veces más rápido es NumPy.

4. Post-mortem. Provoca un ZeroDivisionError, luego ejecuta %debug en la siguiente celda y navega el stack con u/d, inspecciona variables con p.

5. Profila una función. Escribe una función que ordene una lista 1000 veces con sort burbuja. Ejecuta %prun -s cumulative tu_func(). Identifica la línea más cara.

Homework verificable

Entrega un notebook homework.ipynb con: (a) celda que muestra sys.executable confirmando que usas un kernel registrado por ti; (b) benchmark %timeit comparando sum(range(N)) vs np.arange(N).sum() para N=10k, 100k, 1M; (c) tabla markdown con los resultados; (d) gráfico simple del speedup.

Criterio de aceptación: El notebook abre con kernel propio (no el global), las 3 mediciones corren sin errores, y la conclusión incluye un número concreto ("NumPy es ~50× más rápido para N=1M").

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
ModuleNotFoundError aunque acabo de instal	El kernel activo NO es el venv donde corri
El notebook está "congelado" / la barra di	Una celda quedó atrapada en bucle infinito
Cambié código de un módulo importado y el	Python cachea módulos importados. Fix: %lo
%timeit en una celda con asignación da err	Las variables creadas dentro de %timeit no
Outputs gigantes hacen el .ipynb pesado y	Cada output (imagen, tabla) queda guardado

Preguntas frecuentes

¿Notebook clásico o JupyterLab o VS Code?

Para aprender, VS Code (mismo backend, mejor UX: autocomplete con type hints, debug gráfico, git inline). Para reuniones colaborativas en navegador, JupyterLab. El Notebook clásico es legacy — sigue funcionando pero ya no recibe features.

¿Debo crear un kernel por proyecto o usar uno global?

Uno por proyecto. Cada proyecto tiene dependencias distintas que entran en conflicto: el kernel global tarde o temprano se rompe. Comando: python -m ipykernel install --user --name <proyecto>.

¿Cuándo %timeit no es confiable?

Cuando lo que mides toca disco/red/GPU — la varianza es enorme y el min no representa típico. Usa %%time con varios runs manuales y reporta mediana. Tampoco confiable si la primera corrida hace JIT (numba) — calienta con un run previo.

%debug no funciona, no muestra prompt

Necesita haber ocurrido una excepción en el kernel justo antes. Si la celda falló pero el kernel se reinició, perdiste el stack. También: en VS Code Jupyter, usa el panel de debug en su lugar (más cómodo).

¿Por qué mi notebook tarda 30 segundos en abrir si pesa solo 200 KB?

Probablemente trae outputs binarios grandes (imágenes inline en base64). El JSON parece chico pero al renderizar el navegador procesa MB. Limpia outputs y guarda.

Referencias

- VanderPlas, cap. 1 — IPython: Beyond Normal Python.
- IPython magics reference
- JupyterLab user guide

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 003 — Clase 003 — Git y GitHub para data scientists

> *Parte: 0 — Prerrequisitos · Fuente: Pro Git (Chacon & Straub) — caps. 2 y 3 · GitHub docs.*
>
> *Duración estimada: 120 min.*

Objetivo

Que el alumno use git no como "botón save" sino como un sistema serio de versionado: commits atómicos con mensajes útiles, branches por feature, PRs con review, y resolución de conflictos sin pánico. Adicionalmente: ignorar correctamente los archivos típicos de DS (datos pesados, notebooks con output, secrets).

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Inicializar un repo, hacer commits atómicos con mensajes en formato convencional.
2. Trabajar con branches: crear, cambiar, mergear y resolver un conflicto sin perder código.
3. Configurar .gitignore para un proyecto de DS (datos, .venv, secrets, outputs de notebooks).
4. Abrir y revisar un PR en GitHub desde la línea de comandos con gh.
5. Recuperar trabajo perdido con git reflog (la red de seguridad invisible).

Temas

#	Tema	Por qué importa
1	Modelo de git: working tree → staging → re	Sin este modelo mental, todo parece magia.
2	Commits atómicos + mensajes convencionale	Un commit = un cambio lógico revertible.
3	Branches y merge vs rebase	Cuándo usar cada uno; por qué no rebasear
4	.gitignore para data science	Datos, modelos, notebooks con output, .env

5	Conflictos: anatomía y resolución	<<<<<<<, =======, >>>>>>> y cómo no entrar
6	Pull Requests + review en GitHub	El review es donde se transfiere conocimie
7	git reflog — la red de seguridad	Aunque borres una rama, los commits viven

Definiciones y características

Repositorio (repo)

: Carpeta con un subdirectorio .git/ que guarda toda la historia. Características: contenido inmutable identificado por SHA-1, ramas son punteros móviles, todo cambio publicado es eterno (aunque borres el commit, vive en reflog 90 días).

Commit

: Snapshot inmutable del estado del repo en un momento. Tiene SHA-1, padre(s), autor, fecha, mensaje. Característica: atómico — debería poder revertirse solo sin romper nada.

Branch (rama)

: Puntero móvil a un commit. Mover el puntero es barato. HEAD apunta a la rama actual. La rama main no es especial; solo es la rama por defecto del proyecto.

Working tree / Staging / Repo / Remote

: Las 4 zonas: working tree (lo que editas) → staging area (lo preparado con git add) → repo local (lo commiteado) → remote (GitHub/GitLab). Cada git mueve cosas entre estas 4 zonas.

Merge vs Rebase

: Merge crea un commit nuevo que junta dos historias (preserva ambas). Rebase reescribe los commits de tu rama encima de otra (historia lineal pero modificada). Característica clave: nunca rebases ramas compartidas — reescribir SHAs rompe a tus compañeros.

Conventional Commits

: Convención que prescribe tipo(scope): descripción. Tipos: feat, fix, docs, refactor, test, chore, perf, style. Beneficio: changelogs y semver automáticos.

Dataset / recursos

No requiere dataset. El "dataset" son los propios cambios que el alumno hace en archivos de prueba. Para el ejercicio del .gitignore, simulamos archivos típicos de DS (csv pesado, .env, .ipynb_checkpoints/).

Ejercicios

1. Repo desde cero. git init, crea 3 archivos (README.md, data.csv, notebook.ipynb), haz 3 commits con mensajes en formato tipo: descripción (feat/fix/docs/chore).

2. Branch + conflicto. Crea rama feature/x, modifica una línea en README.md. Vuelve a main, modifica la misma línea distinto. Mergea, resuelve el conflicto a mano.

3. .gitignore profesional. Genera uno que ignore: .venv/, __pycache__/, .ipynb_checkpoints/, .csv en data/raw/, .env, models/.pkl. Verifica con git status que no aparecen.

4. PR desde la CLI. Crea repo en GitHub (con gh repo create), push, crea PR con gh pr create y descripción no trivial.

5. Recuperación. Borra una rama con commits. Recupera el HEAD con git reflog + git checkout <sha> + git switch -c rescate.

Homework verificable

Repo público en GitHub con: 5+ commits en formato convencional, al menos 1 branch mergeada, un .gitignore de DS completo, README con badges (build status si aplica) y un PR cerrado.

Criterio de aceptación: El historial (git log --oneline) se lee como cambios atómicos coherentes. git status limpio después de un experimento. PR mergeado con descripción legible.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
error: failed to push some refs to 'origin	El remote tiene commits que no tienes loca
fatal: refusing to merge unrelated histori	Estás juntando dos repos sin ancestro comú
Hice git reset --hard y perdí mi trabajo	Si fue local y no había commit: perdido. S
Please tell me who you are al hacer commit	Falta config global. Fix: git config --glo
Commit con archivo enorme; ahora git push	GitHub bloquea blobs >100 MB. Fix: NO bast
Mergeé un PR pero ahora hay conflictos en	Alguien mergeó algo antes y tu base local
.gitignore no funciona — el archivo sigue	Si el archivo ya estaba trackeado antes de

Preguntas frecuentes

¿Merge o rebase?

Regla simple: merge para todo lo público, rebase solo localmente antes de PR para limpiar tus propios commits. Nunca rebases una rama que alguien más usa.

¿Force push (git push -f) es siempre malo?

En main o ramas compartidas: catástrofe. En tu propia rama de feature después de rebase: aceptable. Mejor usar --force-with-lease que falla si alguien más empujó mientras.

¿Cómo deshago el último commit?

Si NO empujaste: git reset --soft HEAD~1 (mantiene cambios staged) o --hard (los borra). Si YA empujaste y quieres revertirlo sin reescribir historia: git revert HEAD (crea commit nuevo que deshace).

¿Squash o no squash al mergear?

Squash = un solo commit final con todo el PR. Bueno para mantener historia limpia en main. Pierdes el detalle de pasos intermedios. Política común: squash en PRs pequeños, merge commit en grandes.

¿Está bien commitear el .venv/ o el data/raw/customers.csv?

NO. .venv/ se reconstruye con requirements.txt. Datos grandes/sensibles van fuera del repo (DVC, S3, etc.) — ver clase 159 Parte 4.

Tengo 30 commits "wip" en mi rama, ¿qué hago antes del PR?

git rebase -i main para entrar al rebase interactivo. Cambia pick por squash (o fixup) en los commits intermedios; quedará un historial limpio.

Referencias

- Pro Git book — cap. 2 Git Basics, cap. 3 Branching.
- Conventional Commits
- GitHub CLI manual

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 004 — Clase 004 — Estructura reproducible de proyecto (cookiecutter-data-science)

> *Parte: 0 — Prerrequisitos · Fuente: cookiecutter-data-science v2 · Hidden Technical Debt in ML Systems (Sculley et al., 2015).*

> *Duración estimada: 60 min.*

Objetivo

Que el alumno deje de crear proyectos como "una carpeta con notebooks" y empiece a usar una estructura estándar que separa código, datos, modelos, notebooks de exploración y documentación. Esto no es estética — es lo que permite que un compañero entienda el proyecto en 5 minutos y que el código viva más allá del notebook donde nació.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Generar un proyecto con la plantilla cookiecutter-data-science (CCDS v2).
2. Justificar la separación data/raw (inmutable) ↔ data/interim ↔ data/processed.
3. Mover código de un notebook a src/ cuando deja de ser exploratorio.
4. Documentar dependencias en pyproject.toml (no en requirements.txt suelto).
5. Reconocer los olores de un proyecto mal estructurado (notebooks con números 01/02/03, código duplicado, datos en git).

Temas

#	Tema	Por qué importa
1	Estructura CCDS v2	Convención > improvisación.
2	data/raw es sagrado	Nunca se modifica; siempre se puede regene
3	notebooks/ vs src/	Exploración vs producción.
4	pyproject.toml como fuente de verdad de de	Reemplaza requirements.txt suelto.
5	Makefile como interfaz	make data, make train, make test — lo lee
6	Olores típicos	Untitled27.ipynb, datos en git, final_FINA

Complemento: Testing con pytest

El folder tests/ que genera CCDS suele quedar vacío — y es un error. Un data scientist necesita tests por tres razones concretas: las funciones de feature engineering se reusan en producción y un bug silencioso te corrompe el dataset; los pipelines reproducibles se rompen sin avisar cuando cambias una dependencia; y al refactorizar código de notebook a src/ querés saber que el comportamiento sigue siendo el mismo. Tests bien escritos son la red que te deja moverte rápido sin romper lo que ya funciona.

Estructura mínima: archivos test_.py dentro de tests/, funciones test_() adentro, y assert para verificar. pytest descubre todo automáticamente.

Concepto	Para qué sirve
assert	Verificación básica: assert resultado == e
pytest.fixture	Datos o objetos reutilizables entre tests
pytest.mark.parametrize	Corre el mismo test con varios inputs dist
pytest.raises	Verifica que una función levante la excepc
conftest.py	Fixtures compartidas entre varios archivos
--cov	Reporta cobertura de código (requiere pyte

Mini-ejemplo. Función en src/mi_proyecto/features.py:

```
def normalize(x):
    return (x - x.mean()) / x.std()
```

Test en tests/test_features.py:

```
import numpy as np
import pandas as pd
from mi_proyecto.features import normalize

def test_normalize_media_cero():
    s = pd.Series([1.0, 2.0, 3.0, 4.0, 5.0])
    result = normalize(s)
    assert np.isclose(result.mean(), 0.0)
    assert np.isclose(result.std(), 1.0)
```

Correr los tests: pytest -v para ver todo lo que corre, pytest --cov=src tests/ para incluir cobertura.

¿Qué testear en DS? Funciones puras de transformación (limpieza, encoding, feature engineering) sí — son determinísticas y rápidas. Modelos entrenados con aleatoriedad no se testean directamente sobre métricas exactas; o fijás random_state/seeds y verificás contra un valor congelado, o usás snapshots tolerantes (pytest.approx). Lo que importa es testear la lógica que escribiste, no reinventar scikit-learn.

Definiciones y características

Cookiecutter

: Generador de proyectos a partir de plantillas. Tomas una plantilla (URL de un repo), respondes 3-5 preguntas y obtienes un proyecto con estructura pre-armada. Característica: idempotente — la plantilla no sabe ni le importa el contenido futuro del proyecto.

CCDS (cookiecutter-data-science)

: Plantilla específica para proyectos de DS, v2 (2023+). Separa data/raw, data/interim, data/processed, src/, notebooks/, reports/, docs/. Es convención, no dogma.

Editable install (pip install -e .)

: Instala el paquete pero apuntando al código fuente — los cambios se reflejan sin reinstalar. Habilita from mi_proyecto.features import x desde notebooks dentro del repo.

pyproject.toml

: Estándar moderno (PEP 621) para metadata + dependencias + config de tools (ruff, mypy, pytest). Reemplaza setup.py + setup.cfg + requirements.txt suelto + configs sueltas.

Makefile

: Archivo con "recetas" nombradas (make data, make train). Escrito en tabs (no espacios), las dependencias se declaran arriba (target: dep1 dep2). En proyectos DS funciona como interfaz humana a comandos típicos.

Dataset / recursos

Para el ejercicio principal, descarga el dataset de Palmer Penguins (https://github.com/allisonhorst/palmerpenguins) — pequeño (~13 KB), público, ideal para que data/raw/penguins.csv ocupe poco y se pueda commitear sin mala práctica.

Ejercicios

1. Genera un proyecto CCDS. pipx run cookiecutter https://github.com/drivendataorg/cookiecutter-data-science con nombre ds-lab-004. Explora la estructura.

2. Mueve código a src/. Toma una función de un notebook tuyo previo y muévela a src/<proyecto>/features.py. Importa desde el notebook con from <proyecto>.features import ….

3. Convierte requirements.txt a pyproject.toml (sección [project.dependencies]).

4. Refactoriza un notebook caótico. Toma uno con bloques copy-paste y extrae 2 funciones a src/.

5. Lista 5 olores en un repo público que conozcas y propón cómo arreglarlos.

Homework verificable

Un repo público con estructura CCDS, data/raw/ con un dataset pequeño, al menos 1 notebook que importa funciones desde src/, pyproject.toml con deps, Makefile con make setup y make data.

Criterio de aceptación: Un compañero clona, corre make setup && make data y obtiene la misma estructura de datos procesados que tú reportaste. README explica el flujo.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
ModuleNotFoundError: No module named 'mi_p	El paquete no está instalado en el venv de
CSV en data/raw/ apareció en git status y	El .gitignore no cubre data/raw/ o no se a
Tengo 3 notebooks con la misma función lim	Copy-paste. Fix: extrae a src/mi_proyecto/
El compañero clonó el repo y make data fal	Make no está instalado en Windows por defa
Edité pyproject.toml y los imports siguen	Tras cambios en metadata o entry-points de
pytest corre pero dice collected 0 items	Los archivos o funciones no respetan la co

Preguntas frecuentes

¿Realmente necesito una plantilla? ¿No puedo improvisar?

Puedes, pero perderás el 80% de la ganancia: el compañero que ya conoce CCDS sabe dónde buscar; sin plantilla, cada proyecto es una caja de sorpresas.

¿data/raw/ o data/01_raw/?

CCDS v2 usa data/raw/, data/interim/, data/processed/, data/external/. La numeración 01_/02_/03_ la verás en Kedro y en algunas variantes — ambas son válidas, sigue una sola convención por proyecto.

¿requirements.txt o pyproject.toml?

pyproject.toml para declarar las deps de tu paquete. requirements.txt (generado con pip freeze o uv pip compile) es el lockfile con versiones exactas para reproducir. No están en conflicto; conviven.

¿Y si el proyecto es solo un notebook exploratorio?

No fuerces CCDS. Carpeta con notebook.ipynb, data.csv y README.md está bien. La estructura completa aporta cuando el proyecto vive >3 meses o tiene >1 persona.

¿Por qué los notebooks tienen prefijo numérico como 0.01-jvp-eda.ipynb?

Convención CCDS: <fase>.<orden>-<iniciales>-<tema>. Fase 0=exploración, 1=features, 2=modelos, 3=reportes. Iniciales del autor evitan conflictos cuando varias personas crean notebooks.

¿Hace falta testear notebooks?

Los notebooks no se testean directamente — son scratchpads exploratorios y cambian todo el tiempo. Lo que sí se testea es el código que migrás del notebook a src/: en el momento que una función deja de ser exploración y pasa a src/mi_proyecto/, le escribís su test_* correspondiente. Regla práctica: si la función vive en src/, tiene test; si vive en una celda, no.

Referencias

- cookiecutter-data-science v2 docs
- Sculley et al., Hidden Technical Debt in ML Systems (NeurIPS 2015).
- Palmer Penguins dataset
- pytest docs

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 005 — Clase 005 — VS Code / Cursor para Python y Jupyter

> *Parte: 0 — Prerrequisitos · Fuente: VS Code Python docs · Cursor docs · The Pragmatic Programmer cap. "Power Editing".*

> *Duración estimada: 60 min.*

Objetivo

Que el alumno deje de usar VS Code como Notepad y lo configure como un IDE serio para Python + Jupyter: selector de intérprete, debugger gráfico, linter (ruff), formatter (ruff format), tests integrados y notebooks editables. Bonus: cuándo conviene Cursor (VS Code + IA integrada).

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Configurar VS Code con la extensión Python + Jupyter, seleccionando el intérprete del venv del proyecto.
2. Debuggear un script Python paso a paso desde el panel gráfico (breakpoints, watch, call stack).

3. Editar y ejecutar notebooks sin Jupyter web — con autocompletado, type hints y debug de celda.
4. Configurar ruff como linter + formatter (reemplaza black + isort + flake8 en un solo tool).
5. Decidir cuándo usar Cursor (idéntico a VS Code + IA integrada con autorización por chat).

Temas

#	Tema	Por qué importa
1	Selección de intérprete por workspace	El bug "funciona en terminal pero no en VS
2	Debugger gráfico vs print	Breakpoints, watch, evaluación expresiones
3	Notebooks nativos en VS Code	Mejor UX que Jupyter web para edición; mis
4	ruff = linter + formatter en uno	Reemplaza black/isort/flake8/pylint. Más r
5	Tests integrados (pytest)	Run/debug tests con un click; coverage inl
6	Extensiones esenciales	Python, Jupyter, GitLens, ruff, Even Bette
7	Cursor: cuándo sí	Cuando quieres pair-programming con IA sin

Definiciones y características

Workspace

: Concepto de VS Code = una carpeta (o conjunto de carpetas) con configuración asociada en .vscode/settings.json. La configuración del workspace override a la del usuario. Característica: pones .vscode/ en git para que todos los colaboradores hereden la misma config.

Intérprete Python

: Ejecutable concreto (/path/to/.venv/bin/python). VS Code recuerda uno por workspace. Es el origen del 90% de los "funciona en mi máquina" entre IDE y terminal.

ruff

: Linter + formatter en un solo binario, escrito en Rust. Reemplaza black + isort + flake8 + (parte de) pylint con un único tool 10–100× más rápido. Config en [tool.ruff] de pyproject.toml.

Breakpoint

: Marca en una línea (F9) que pausa la ejecución cuando llega ahí. Permite inspeccionar variables, paso a paso, evaluar expresiones — mil veces más eficiente que print.

launch.json

: Config de debug de VS Code. Define perfiles: "debug archivo actual", "debug tests", "debug Django", etc. Cada perfil tiene su program, args, env, justMyCode.

Dataset / recursos

Sin dataset externo. Usamos un script Python con un bug intencional para practicar debug gráfico, y un notebook trivial para verificar autocompletado/type hints.

Ejercicios

1. Selecciona intérprete. En VS Code: Ctrl+Shift+P → "Python: Select Interpreter" → elige el del .venv del proyecto. Verifica con print(sys.executable) en una celda.

2. Debug paso a paso. Toma un script con un bug, pon breakpoint (F9), ejecuta con F5, navega con F10 (next), F11 (step in), Shift+F11 (step out). Inspecciona variables en panel.

3. Configura ruff. En pyproject.toml: [tool.ruff] con line-length = 100, [tool.ruff.lint] con select = ["E", "F", "I", "UP"]. Habilita format on save.

4. Edita un notebook. Abre notebook.ipynb en VS Code, ejecuta una celda, comprueba que el autocompletado funciona con type hints de pandas.

5. Tests con un click. Instala pytest, crea tests/test_simple.py con 2 tests (uno OK, uno FAIL). Usa el panel "Testing" para correr/debuggear.

Homework verificable

Repo con pyproject.toml que incluye [tool.ruff], .vscode/settings.json con python.defaultInterpreterPath y format-on-save habilitado, screenshot del debugger gráfico mostrando un breakpoint activo y variables inspeccionadas.

Criterio de aceptación: Otro alumno clona el repo, abre en VS Code, y al guardar un .py se aplica ruff format automáticamente. El screenshot muestra al menos 1 breakpoint y la variable inspeccionada.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
"Python interpreter is not selected" al ab	Workspace nuevo, VS Code no eligió uno. Fi
Format-on-save no aplica ruff aunque está	Falta declarar ruff como formatter por def
Debugger arranca pero se salta mis breakpo	Estás corriendo el archivo (Ctrl+F5 = sin
Tests no aparecen en el panel "Testing"	VS Code no detectó pytest. Fix: Ctrl+Shift
Cambié interpreter y los imports siguen ro	VS Code cachea symbols del intérprete viej

Preguntas frecuentes

¿VS Code o Cursor?

Cursor = VS Code + IA integrada (chat con contexto del repo, edición multi-archivo). Si pagas Copilot o no te interesa IA, quédate en VS Code. Si quieres pair-programming con IA sin saltar a otra app, Cursor. Las extensiones son las mismas.

¿Debo commitear .vscode/?

Sí la parte compartida: settings.json (interpreter path relativo, formatter, etc.), extensions.json (recomendaciones). No lo personal: .vscode/launch.json con paths absolutos del tester.

¿Notebook en VS Code o en JupyterLab?

VS Code para escribir/refactorizar (autocomplete con type hints, debug por celda, git inline). JupyterLab cuando alguien necesita un navegador y no quiere instalar VS Code (alumno, demo en proyector).

¿Para qué justMyCode: false?

Por default, el debugger se salta código de librerías de terceros (numpy, pandas) — útil para no perderte. Pero a veces el bug viene desde dentro de pandas (datos malformados); con false puedes entrar a ver.

¿Ruff reemplaza todo el stack? ¿No necesito black?

Sí — ruff format es drop-in replacement de black (mismo output prácticamente). Mismo con isort (ruff check --select I --fix) y flake8 (ruff check). Único caso donde aún conviene black: si tu org ya tiene CI con black configurado y no quieres tocar.

Referencias

- VS Code Python docs

- VS Code Jupyter docs
- ruff docs
- Cursor docs

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 006 — Clase 006 — Python: tipos, estructuras, control de flujo

> *Parte: 0 — Prerrequisitos · Fuente: Python Tutorial oficial (caps. 3-5) · Fluent Python (Ramalho, 2ª ed.) cap. 1.*

> *Duración estimada: 120 min.*

Objetivo

Refrescar (o instalar) los cimientos de Python que el resto del programa asume: tipos primitivos, las 4 estructuras built-in (list, tuple, set, dict), control de flujo (if/for/while), unpacking, truthiness y la diferencia entre mutables e inmutables — la fuente del 90% de bugs sutiles.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Diferenciar tipos mutables (list, dict, set) vs inmutables (tuple, str, int, frozenset) y predecir el efecto en asignaciones.
2. Usar las 4 estructuras eligiendo bien: list (orden + duplicados), tuple (inmutable, rápida), set (unicidad), dict (lookup O(1)).
3. Aplicar unpacking en for, returns múltiples y args/*kwargs.
4. Evaluar truthiness correctamente ([], {}, 0, '', None son falsy; el resto es truthy).
5. Identificar el bug del default mutable en funciones (def f(x, lst=[])) y por qué es trampa.

Temas

#	Tema	Por qué importa
1	Mutables vs inmutables	Define qué pasa con a = b.
2	list, tuple, set, dict — cuándo cada uno	Complejidad y semántica distintas.
3	Iteración: for, enumerate, zip	Idiomático > C-style.
4	Unpacking y starred expressions	a, *b, c = [1,2,3,4,5].
5	Truthiness y operadores and/or	Evalúan al objeto, no al booleano.
6	Default mutables: el clásico	def f(x, lst=[]) comparte la lista entre l

Definiciones y características

Mutable

: Objeto cuyo estado puede modificarse después de crearlo (list, dict, set, bytearray). Consecuencia: si dos variables apuntan al mismo objeto mutable, cambiar una cambia la otra (alias).

Inmutable

: Objeto que NO se puede modificar; cualquier 'modificación' produce un objeto nuevo (int, float, str, tuple, frozenset). Característica clave: son hashables y pueden ser claves de dict o miembros de set.

Hashable

: Objeto con __hash__ definido y constante durante su vida. Los inmutables built-in son hashables; las listas y dicts NO lo son. Es lo que permite el O(1) lookup de set/dict.

Unpacking

: Sintaxis para asignar múltiples variables desde un iterable (a, b = (1, 2) o a, resto = [1, 2, 3, 4]). El captura el resto en una lista. Funciona en for, return, llamadas a funciones (f(lista), g(*dict)).

Truthy / Falsy

: Cómo evalúa Python un objeto en contexto booleano (if x:). Falsy: False, None, 0, 0.0, '', [], {}, set(). Truthy: todo lo demás (incluso ' ' con espacio, [0], {None: None}).

Dataset / recursos

Datos sintéticos pequeños generados en el notebook (lista de diccionarios simulando estudiantes). No requiere descarga.

Ejercicios

1. Cuenta palabras. Dado un texto, devuelve un dict[str, int] con frecuencias. Sin usar Counter.

2. Unique con orden. Recibe list[int], devuelve la lista de únicos manteniendo el orden de primera aparición.

3. Reproduce el bug del default mutable. Escribe def add(item, target=[]), llámala 3 veces con add('x'). Observa. Explica por qué y arregla.

4. Top-K palabras. Mismo texto del ejercicio 1, devuelve las 5 más frecuentes ordenadas por frecuencia descendente.

5. Grupos por inicial. Dado list[str], devuelve dict[str, list[str]] agrupando por primera letra (case-insensitive).

Homework verificable

Notebook homework.ipynb con las 5 funciones de los ejercicios, cada una con: (a) implementación, (b) 3 casos de prueba (incluyendo edge cases — lista vacía, string vacío), (c) docstring corto explicando complejidad.

Criterio de aceptación: Las 5 funciones pasan sus casos de prueba; los edge cases manejados sin excepción.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
TypeError: unhashable type: 'list'	Intentaste usar una lista como clave de di
Modifiqué una lista y otra variable tambié	Las dos apuntan al mismo objeto (alias). F
Función con default =[] comparte la lista	El default se evalúa una vez al definir la
KeyError al acceder a dict que no tiene la	d['x'] lanza KeyError si no existe. Fix: u
if items != None: en vez de if items is no	Para singletons (None, True, False) usa si

Preguntas frecuentes

¿Cuándo tupla y cuándo lista?

Tupla para records heterogéneos de tamaño fijo (punto = (3.5, 7.1), (nombre, edad)). Lista para colecciones homogéneas que crecen (temperaturas = [22.1, 22.5, ...]). Tupla además sirve como clave de dict; lista no.

¿Set o dict para chequear pertenencia?

Ambos son O(1). Set si solo necesitas saber si está. Dict si además necesitas asociar valor. Una list para esto es O(n) — usa set/dict si haces if x in coleccion con N grande.

¿copy() me da una copia completa? ¿Y con listas anidadas?

.copy() es shallow: copia el contenedor pero comparte las referencias internas. Para anidamiento, import copy; copy.deepcopy(x) — más lento pero independiente.

¿Por qué 0.1 + 0.2 != 0.3?

Floats son IEEE 754 binarios; 0.1 y 0.2 no tienen representación exacta. Fix: para igualdad usa math.isclose(a, b). Para precisión decimal financiera, from decimal import Decimal.

¿Qué pasa con dict si insertas la misma clave dos veces?

El segundo valor sobrescribe al primero. El orden de inserción se preserva (Python 3.7+), así que list(d) da las keys en el orden en que se insertaron por primera vez.

Referencias

- Python Tutorial — Data Structures
- Ramalho, Fluent Python 2e — cap. 1 The Python Data Model.
- Python Tutorial — Control flow

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 007 — Clase 007 — Comprehensions y generadores

> *Parte: 0 — Prerrequisitos · Fuente: Ramalho, Fluent Python 2e — caps. 2 (Sequences) y 17 (Iterators, Generators, Coroutines).*

> *Duración estimada: 90 min.*

Objetivo

Que el alumno escriba código Python idiomático: list/dict/set comprehensions en vez de for+append, generadores cuando el dataset no cabe en memoria, y entienda la diferencia fundamental entre construir una lista y producir un iterable perezoso.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Convertir loops for+append a list/dict/set comprehensions sin perder legibilidad.

2. Usar generadores (yield y generator expressions) para procesar datos que no caben en RAM.
3. Distinguir [x for x in xs] (lista) vs (x for x in xs) (generador): memoria y consumo.
4. Encadenar generadores con itertools (chain, islice, takewhile, groupby).
5. Identificar cuándo NO usar comprehension (lógica compleja, side effects, debug difícil).

Temas

#	Tema	Por qué importa
1	List comprehension: [expr for x in xs if c	Idiomático, eficiente, legible si es simpl
2	Dict/set comprehensions	Mismo patrón, otra estructura.
3	Generator expressions: (expr for x in xs)	Perezoso, memoria O(1).
4	Funciones generadoras con yield	Reescribe procesos como streams.
5	itertools — la caja de herramientas	chain, islice, groupby, accumulate, combin
6	Comprehension vs loop: cuándo NO	Lógica >2 líneas, side effects, debug.

Definiciones y características

List comprehension

: Expresión [expr for x in iterable if cond] que construye una lista completa en memoria. Equivalente idiomático a for + append + if. Más rápida y legible si la expresión es simple.

Generator expression

: Lo mismo pero con paréntesis (expr for x in iterable if cond). Devuelve un generador perezoso: produce cada valor on-demand, memoria O(1). Solo puedes recorrerlo una vez.

Iterador

: Objeto con método __next__() que devuelve el siguiente valor o lanza StopIteration. Es lo que está detrás de un for. Características: lazy, single-pass, memoria O(1).

Generador

: Función con yield (o generator expression) que produce un iterador. Cada yield pausa la función y emite un valor; al siguiente next() retoma desde ahí. Mantiene el estado local entre llamadas.

itertools

: Librería stdlib con bloques perezosos componibles: chain (concatena), islice (slicing perezoso), groupby (agrupa consecutivos), accumulate (sum/prod corridos), takewhile, combinations, product.

Dataset / recursos

Datos sintéticos: rango grande de números (1M elementos) para mostrar diferencia memoria lista vs generador. Sin descarga.

Ejercicios

1. De for a comprehension. Toma 3 loops for+append (cuadrados, filtra pares, mapea a strings) y conviértelos.

2. Generador de Fibonacci infinito. Función con yield que produce Fibonacci. Úsala con itertools.islice para tomar los primeros 20.

3. Memoria: lista vs generador. Mide RAM (con tracemalloc) de sum([ii for i in range(10_000_000)]) vs sum(ii for i in range(10_000_000)). Reporta la diferencia.

4. Procesa CSV línea por línea. Lee un archivo grande con yield línea por línea, filtra por una condición,

cuenta sin cargar todo en memoria.

5. Pivot con dict comprehension. Dada list[tuple[str, int]] (nombre, puntaje), construye dict[str, list[int]] agrupando puntajes por nombre.

Homework verificable

Notebook que: (1) reescribe 3 loops como comprehensions, (2) implementa generador Fibonacci con islice, (3) comparativa RAM lista vs generador con tracemalloc y tabla de resultados, (4) lee un CSV ≥10k filas con generador y filtra sin cargar entero.

Criterio de aceptación: La medición de RAM muestra >100× menos memoria con generador. CSV se procesa sin OOM.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Iteré un generador y la segunda vez está v	Los generadores son single-pass. Tras cons
MemoryError al hacer [expensive(x) for x i	Construyes lista completa en RAM. Fix: usa
itertools.groupby me agrupa raro	Solo agrupa elementos consecutivos con mis
Generator expression dentro de función fal	Estás haciendo gen[0] — generadores no son
List comprehension con if-else no funciona	if al final filtra; if-else va al principi

Preguntas frecuentes

¿Cuándo comprehension y cuándo for clásico?

Comprehension cuando es una expresión clara en 1 línea. For clásico cuando hay >2 statements, side effects (print, mutación), o la lógica es más legible explícita.

¿Generator expression o list?

Generator si el resultado solo se consume una vez y N es grande (RAM importa). List si necesitas recorrer 2+ veces, indexar, o hacer len(). Truco: sum(xx for x in xs) evita lista temporal vs sum([xx for x in xs]).

¿Cuánto más rápido es vs un for tradicional?

~10-30%, no mil veces más. La ganancia real es legibilidad. Si necesitas mil veces, no es comprehension lo que buscas — es numpy/vectorización (clase 015).

¿Generador infinito (while True: yield ...) es buena idea?

Sí, pero ojo: nunca lo conviertas a lista directo (list(gen)) — bucle infinito hasta OOM. Acopla con itertools.islice(gen, N) para truncar.

yield from vs yield?

yield from sub_iterable delega: yieldea todos los valores del sub-iterable. Equivale a for x in sub: yield x pero más rápido y propaga send()/throw() correctamente. Útil para componer generadores.

Referencias

- Ramalho, Fluent Python 2e — caps. 2 y 17.
- PEP 202 — List Comprehensions
- PEP 255 — Simple Generators
- itertools docs

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 008 — Clase 008 — Funciones: args, kwargs, lambdas, closures

> *Parte: 0 — Prerrequisitos · Fuente: Ramalho, Fluent Python 2e — cap. 7 (Functions as First-Class Objects), cap. 9 (Decorators and Closures).*

> *Duración estimada: 90 min.*

Objetivo

Que el alumno use funciones como ciudadanos de primera clase: pasarlas como argumento, retornarlas, escribir lambdas cuando aportan, y entender closures — la base de los decoradores que verán más adelante. Sin esto, el código pandas/sklearn parece magia.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Definir funciones con argumentos posicionales, keyword-only, args y *kwargs.
2. Pasar funciones como argumento (callbacks: sorted(xs, key=fn), df.apply(fn)).
3. Usar lambdas donde son legibles (callbacks cortos) y evitarlas donde no (lógica).
4. Explicar y escribir closures (función que captura variables del scope exterior).
5. Anticipar la diferencia entre args y , args (keyword-only marker).

Temas

#	Tema	Por qué importa
1	Argumentos: posicional, keyword, default	Cuatro modos, una sintaxis.
2	args y *kwargs	Funciones que aceptan número variable.
3	Keyword-only con * separador	def f(a, *, b) → b solo nombrado.
4	Funciones como objetos	Asignables, pasables, retornables.
5	Lambdas: dónde sí y dónde no	Callbacks cortos sí; lógica compleja no.
6	Closures: capturando scope	Base mental de los decoradores.

Definiciones y características

First-class object

: En Python, funciones son ciudadanos de primera clase: se asignan a variables (f = saludar), se pasan como argumento (sorted(xs, key=f)), se retornan de otras funciones. Esto habilita callbacks, decoradores y closures.

args / kwargs*

: args captura argumentos posicionales sobrantes en una tupla. kwargs captura argumentos nombrados sobrantes en un dict. Convención: solo el y ** importan; los nombres args/kwargs son convención.

Keyword-only argument

: Argumento que solo puede pasarse nombrado: declarado después de o args en la signatura. def f(a, *, b) obliga a f(1, b=2). Mejora legibilidad en APIs con muchos params.

Lambda

: Función anónima de UNA expresión: lambda x: x*2. Sin nombre, sin docstring, sin múltiples statements. Útil para callbacks cortos (sorted(xs, key=lambda p: p['edad'])). Si necesitas más, usa def.

Closure

: Función que captura variables del scope donde fue definida y las mantiene vivas aunque ese scope termine. Base mental de los decoradores. Para modificar la variable capturada, usa nonlocal.

Decorador

: Función que recibe función y retorna función (típicamente envuelta). Sintaxis: @dec antes de def. Implementado típicamente con closure + @functools.wraps para preservar metadata original.

Dataset / recursos

Datos sintéticos pequeños (lista de dicts simulando ventas). Sin descarga.

Ejercicios

1. Función con todo. Define f(a, b=10, args, c, *kwargs). Llámala de 3 formas distintas que sean válidas. Identifica qué llamadas son inválidas y por qué.

2. sorted con key. Dada list[dict] de personas, ordena por edad (asc) y por nombre alfabético. Usa lambda primero, luego operator.itemgetter.

3. Closure contador. Escribe make_counter() que retorna una función que cada vez que se llama incrementa y retorna un contador interno. ¿Por qué funciona?

4. Memoización manual. Implementa un decorador @memoize usando closure + dict. Aplícalo a Fibonacci recursivo y mide el speedup con %timeit.

5. Compose. Escribe compose(f, g, h) que retorna una función equivalente a lambda x: f(g(h(x))).

Homework verificable

Notebook con: (a) implementación y demo de make_counter explicando con comentario por qué el contador persiste; (b) @memoize aplicado a Fibonacci recursivo con benchmark (N=35) antes/después; (c) ordenamiento de list[dict] por 2 criterios usando itemgetter.

Criterio de aceptación: memoize reduce Fibonacci(35) de segundos a milisegundos. Counter independiente entre instancias.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
UnboundLocalError: local variable 'x' refe	Asignaste a x dentro de la función → Pytho
SyntaxError: positional argument follows k	Llamaste f(a=1, 2) — posicionales primero.
TypeError: f() got multiple values for arg	Pasaste x posicional Y nombrado: f(5, x=10
Mi decorador rompe help(funcion)	Sin @functools.wraps(fn), el wrapper pierd
Lambda en loop captura el último valor	[lambda: i for i in range(3)] — todas las

Preguntas frecuentes

¿Cuándo args, kwargs y cuándo argumentos explícitos?*

Argumentos explícitos siempre que conozcas la signatura — el IDE te ayuda y el lector entiende. args, *kwargs solo en wrappers genéricos (decoradores, factories) que deben aceptar cualquier llamada.

¿Lambda o def?

Lambda solo si: (a) cabe en una expresión, (b) la usas inmediatamente (callback), (c) un nombre no aportaría. En todos los demás casos, def con nombre — más debuggeable, soporta docstring y type hints.

¿Closure es lo mismo que decorador?

Decorador suele estar implementado con closure, pero closure ≠ decorador. Closure es cualquier función que captura su entorno; decorador es un patrón específico (función → función).

¿Por qué necesito nonlocal en make_counter?

Sin nonlocal, count += 1 dentro de inner se interpretaría como variable local nueva y daría UnboundLocalError. nonlocal le dice: 'esa variable vive en el scope inmediato exterior, modifícala'.

¿Cuál es el costo de pasar funciones como argumento?

Mínimo (es solo una referencia). Lo costoso es la invocación repetida en bucles tight (cada llamada Python tiene overhead). Para esto, NumPy/Cython/Numba.

Referencias

- Ramalho, Fluent Python 2e — caps. 7 y 9.
- Python docs — More on Defining Functions
- PEP 3102 — Keyword-Only Arguments

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 009 — Clase 009 — Manejo de excepciones y context managers

> *Parte: 0 — Prerrequisitos · Fuente: Python Tutorial cap. 8 (Errors and Exceptions) · Ramalho, Fluent Python 2e — cap. 18 (Context Managers).*

> *Duración estimada: 75 min.*

Objetivo

Que el alumno maneje excepciones con criterio (sin except: pass), construya jerarquías de excepciones propias cuando aporta, y use context managers (with) — tanto los built-in como propios con @contextmanager — para garantizar limpieza de recursos. Sin esto, el código de carga de datos es una bomba de relojería.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Diferenciar los 3 tipos de errores (Syntax, runtime exceptions, logical) y dónde se manejan.
2. Capturar excepciones específicas (except ValueError, no except:) y propagar las que no sabes manejar.
3. Crear una excepción propia heredando de la jerarquía estándar (class DatasetCorruptoError(Exception)).
4. Usar with para archivos, sesiones HTTP, transacciones DB.
5. Escribir un context manager propio con @contextmanager (timer, supress, change_dir).

Temas

#	Tema	Por qué importa
1	Jerarquía de excepciones built-in	BaseException → Exception → ValueError/Key
2	try/except/else/finally	Cada bloque tiene un rol específico.
3	Capturar específico, no genérico	except: esconde bugs.
4	Excepciones propias	Comunican intención en vez de cargar mensa
5	Context managers: protocolo __enter__/__ex	Garantiza cleanup.
6	@contextmanager de contextlib	Crear cms con función + yield.

Definiciones y características

Excepción

: Objeto que se 'lanza' (raise) cuando algo anómalo ocurre. Sube por el stack hasta que un except lo captura, o termina el programa. Todas heredan de BaseException; las que debes capturar heredan de Exception.

try/except/else/finally

: try: código riesgoso. except: maneja una excepción específica. else: corre si try no lanzó (raro pero útil). finally: cleanup garantizado, lanzó o no.

Jerarquía de excepciones

: BaseException → SystemExit / KeyboardInterrupt / Exception → ValueError / TypeError / LookupError (→ KeyError, IndexError) / OSError / ArithmeticError (→ ZeroDivisionError). Captura siempre la más específica que sepas manejar.

Excepción propia (custom)

: Subclase de Exception (o subclase específica). Permite tipificar errores de tu dominio (class DatasetCorruptoError(Exception): ...) en vez de strings. El caller puede except DatasetCorruptoError con precisión.

Context manager

: Objeto con __enter__ y __exit__ que se usa con with. Garantiza setup/teardown (abrir/cerrar archivo, conectar/desconectar BD, lock/unlock). El __exit__ corre incluso si hay excepción.

@contextmanager

: Decorador de contextlib que convierte una función con yield en context manager. Pre-yield = __enter__, post-yield = __exit__. Mucho más corto que escribir la clase completa.

Dataset / recursos

Archivo temporal generado en el notebook. Sin descarga.

Ejercicios

1. Captura específica. Escribe una función parse_int_safe(s, default=0) que use try/except solo para ValueError. Demuestra que no esconde otros errores (ej. TypeError si pasas un dict).

2. Excepción propia. Define class DatasetCorruptoError(Exception) con un atributo linea. Lanzala desde una función cargar_csv cuando una línea no tenga el número correcto de columnas.

3. with para archivo. Lee un archivo línea por línea contando palabras. Compara con la versión sin with (manual open/close) y muestra qué pasa si hay excepción a mitad.

4. Context manager propio: timer. Con @contextmanager, escribe with timer("carga"): que imprima cuánto duró el bloque.

5. Context manager: change_dir. with cd("/tmp"): cambia de directorio al entrar y vuelve al salir — incluso si hay excepción.

Homework verificable

Notebook con: (a) parse_int_safe con tests de los 3 casos (válido, inválido, otro tipo); (b) DatasetCorruptoError usada en una función cargar_csv que valida #columnas; (c) decorador-context manager timer aplicado a 2 operaciones; (d) cd context manager.

Criterio de aceptación: Excepciones se capturan solo donde sabes manejarlas. timer reporta segundos correctamente.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
except: o except Exception: ciego esconde	Cualquier error queda silenciado (incluso
finally con return traga la excepción	Si finally ejecuta return, la excepción de
with open(f) as f, open(g) as g: falla por	Esa sintaxis requiere Python 3.10+. Fix: e
Capturé KeyError pero el código sigue romp	Tu except está más arriba del try, o el er
Excepción dentro de generator no se captur	Las excepciones en generators son tricky;

Preguntas frecuentes

¿except Exception o except?

Casi siempre except Exception — más explícito. except: desnudo captura también KeyboardInterrupt y SystemExit, lo cual rompe Ctrl+C y sys.exit().

¿Cuándo creo una excepción propia?

Cuando el caller necesita diferenciar este error de otros. raise ValueError('CSV corrupto') obliga al caller a parsear strings; raise DatasetCorruptoError(linea=42) le da un tipo y un atributo concreto.

¿with solo para archivos?

No — para CUALQUIER recurso que necesite cleanup. Files (open), conexiones DB (engine.connect()), locks (threading.Lock), sesiones HTTP (requests.Session), transacciones (db.atomic()), timing (with timer(...)).

¿Debo capturar y re-lanzar para añadir contexto?

Sí, con raise ExceptionNueva(...) from e — preserva el stacktrace original como '__cause__'. El log muestra ambos errores en cadena.

¿pass en except es siempre malo?

Casi siempre sí. Si tienes que silenciar, al menos log.warning(...). Solo OK cuando la excepción es esperada (except FileNotFoundError: pass al limpiar archivos opcionales).

Referencias

- Python Tutorial — Errors and Exceptions
- Ramalho, Fluent Python 2e — cap. 18 Context Managers and else Blocks.
- contextlib docs

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 010 — Clase 010 — OOP básico, dataclasses, herencia

> *Parte: 0 — Prerrequisitos · Fuente: Ramalho, Fluent Python 2e — caps. 5 (Data Class Builders) y 14 (Inheritance) · Python Tutorial cap. 9.*

> *Duración estimada: 90 min.*

Objetivo

Que el alumno escriba clases cuando aportan (no por hábito Java), use @dataclass para records sin boilerplate, entienda herencia con criterio (preferir composición), y conozca los métodos dunder más usados (__repr__, __eq__, __lt__, __len__).

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Definir clases con __init__, atributos de instancia y métodos.
2. Usar @dataclass para records inmutables/mutables sin escribir __init__/__repr__/__eq__.
3. Heredar y sobreescribir métodos con super().
4. Implementar dunders esenciales: __repr__, __str__, __eq__, __lt__, __len__, __iter__.
5. Decidir entre clase, dataclass o NamedTuple según el caso.

Temas

#	Tema	Por qué importa
1	Clase mínima: __init__ + atributos + métod	El bloque básico.
2	@dataclass(frozen=True)	Records inmutables sin boilerplate.
3	Herencia + super()	Reutilizar implementación de la clase base
4	Composición > herencia	"Has-a" generalmente mejor que "is-a".
5	Métodos dunder	Integran tu clase con len(), ==, repr(), s
6	dataclass vs NamedTuple vs TypedDict	Elegir según necesidad de mutabilidad/comp

Definiciones y características

Clase / instancia

: Una clase es una plantilla (class Punto:); una instancia es un objeto concreto (p = Punto(3, 4)). __init__ se llama al crear la instancia. self es la convención para referirse a la instancia dentro de los métodos.

Método dunder ("magic method")

: Método con doble underscore (__init__, __repr__, __eq__, __lt__, __len__, __iter__, __add__). Python los invoca implícitamente con sintaxis especial (len(obj) → obj.__len__()).

@dataclass

: Decorador que genera __init__, __repr__, __eq__ automáticamente desde las anotaciones de tipo de la clase. Reduce boilerplate. Con frozen=True la hace inmutable y hashable.

Herencia

: class B(A) — B hereda atributos y métodos de A; puede sobreescribirlos. super() invoca al método de la clase padre. Múltiple herencia existe pero se complica (MRO).

Composición

: "B tiene un A" (atributo) en vez de "B es un A" (herencia). Generalmente preferible: menos acoplamiento, sin problemas de herencia múltiple/diamante.

Polimorfismo

: Distintas clases responden al mismo método con comportamiento distinto (Animal.hablar() → 'guau' o 'miau' según la subclase). Permite tratar instancias heterogéneas uniformemente.

NamedTuple vs dataclass vs TypedDict

: NamedTuple: tupla con nombres, inmutable, hashable, sin métodos custom. dataclass: clase con boilerplate auto, mutable por default (mejor con métodos). TypedDict: dict con esquema (estructural, no nominal).

Dataset / recursos

Sintético: lista de objetos Punto y Estudiante. Sin descarga.

Ejercicios

1. Clase Punto. Define Punto(x, y) con __repr__, __eq__, distancia al origen y __add__ para sumar puntos.

2. Dataclass Estudiante. @dataclass con nombre, notas: list[float], método promedio(). Crea 3 instancias, ordena por promedio.

3. Frozen Vector. @dataclass(frozen=True) para un vector 2D inmutable. Intenta modificar un atributo y observa la excepción.

4. Herencia. Animal con hablar() → 'genérico'. Perro(Animal) que sobreescribe a 'guau'. Gato(Animal) a 'miau'.

5. Composición. Coche que tiene un Motor (composición) en vez de heredar de Motor. Justifica por qué.

Homework verificable

Notebook con: (a) Punto con 4 dunders y tests; (b) @dataclass Estudiante con sort por promedio; (c) @dataclass(frozen=True) Vector que demuestra inmutabilidad lanzando excepción; (d) jerarquía Animal → Perro/Gato con polimorfismo (lista mixta llamando hablar()).

Criterio de aceptación: Las 4 clases pasan tests; frozen=True lanza FrozenInstanceError al asignar.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Olvidé self en un método y el error es con	Cualquier método de instancia recibe self
@dataclass con field mutable default rompe	@dataclass class X: items: list = [] lanza
__eq__ definido pero __hash__ rompe	Definir __eq__ sin __hash__ hace la clase
super().__init__(...) olvidado en subclase	Atributos del padre quedan sin inicializar
Modifico atributo y otra instancia también	Asignaste un mutable como class attribute,

Preguntas frecuentes

¿Cuándo necesito OOP en data science?

Menos de lo que crees. Para análisis exploratorio, funciones + dicts/dataclasses bastan. Necesitas clases cuando hay: estado mutable complejo (modelos sklearn), polimorfismo (varios algoritmos misma interfaz), o frameworks que lo exigen (PyTorch nn.Module).

¿@dataclass o NamedTuple o pydantic?

NamedTuple: record inmutable simple, sin validación. dataclass: record con métodos opcionales. pydantic (no en stdlib): cuando además quieres validación de tipos en runtime, parsing desde JSON, etc. (lo verás en MLOps).

¿Composición > herencia siempre?

Como regla. Usa herencia solo cuando is-a sea genuino (PerroLabrador is-a Perro is-a Animal). Para has-a (Coche tiene un Motor), composición. Para reutilizar comportamiento sin jerarquía, considera mixins o protocols.

¿property y getters/setters Java-style?

En Python no escribes getNombre()/setNombre(). Usa atributo público (self.nombre = ...). Si después necesitas lógica, conviertes a @property sin cambiar el caller. Es la magia.

¿Cuándo __slots__?

Optimización: define los atributos permitidos y ahorra memoria (~50%) al no usar __dict__ por instancia. Útil solo en clases con millones de instancias. Costo: pierde herencia múltiple y dinamismo.

Referencias

- Ramalho, Fluent Python 2e — caps. 5, 11, 14.
- dataclasses docs
- Python Tutorial — Classes

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 011 — Clase 011 — pathlib, lectura y escritura de archivos

> *Parte: 0 — Prerrequisitos · Fuente: Python Tutorial cap. 10 · pathlib docs · Effective Python (Slatkin) ítem 38.*

Duración estimada: 60 min.

Objetivo

Que el alumno deje de usar os.path.join + strings y adopte pathlib.Path — API orientada a objetos, multiplataforma (Windows/Unix), con métodos legibles para todas las operaciones de filesystem que hace todo el tiempo en DS (leer CSV, listar archivos, crear carpetas).

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Construir paths con Path(...) / 'subdir' / 'file.csv' (operador /).
2. Leer/escribir archivos texto y binarios con métodos de Path (read_text, write_bytes).
3. Listar y filtrar archivos con iterdir, glob, rglob (recursivo).
4. Crear/eliminar estructuras de directorios sin pelear con os.makedirs(exist_ok=True).
5. Manejar rutas relativas vs absolutas y entender __file__.

Temas

#	Tema	Por qué importa
1	Path vs strings	Objetos con métodos > concatenación manual
2	Operador / para componer	Legible y multiplataforma.
3	read_text / write_text / read_bytes	One-liners para operaciones simples.
4	glob y rglob	Patrones tipo shell: .csv, /.py.
5	mkdir(parents=True, exist_ok=True)	Crea árbol completo idempotente.
6	Path(__file__).parent y resolve()	Localizar recursos relativos al script.

Versión profundizada — 2026

El tema moderno que vivía como complemento dentro de esta clase ahora tiene clase propia dedicada:

- Clase 032b — Parquet, Arrow, PyArrow, DuckDB

Definiciones y características

Path

: Objeto que representa una ruta de filesystem orientada a objetos (pathlib.Path). Sobreescribe / para componer rutas, multiplataforma (Windows usa \, Unix /, transparente). Tiene métodos para casi todo: leer, escribir, listar, mover, borrar.

Ruta absoluta vs relativa

: Absoluta: empieza desde la raíz (C:\dev\proyecto\data.csv o /home/user/data.csv). Relativa: parte del cwd (data.csv). Las rutas relativas dependen de dónde se ejecuta — fuente de bugs.

Path.cwd() vs Path(__file__).parent

: cwd() es el directorio donde se ejecutó el script (cambia según el invocante). __file__ es la ruta al archivo Python actual; .parent su carpeta. Usa __file__ para recursos que viven junto al script.

glob vs rglob

: Patrones tipo shell. glob('.csv') busca en el directorio actual (1 nivel). rglob('.csv') busca recursivo (todo el árbol). ** significa 'cualquier número de directorios'.

read_text / write_text

: One-liners para texto: Path('x.txt').write_text(contenido, encoding='utf-8'). Para binario: read_bytes / write_bytes. Para JSON/CSV usa librerías especializadas.

Dataset / recursos

Carpeta temporal creada en el notebook con archivos sintéticos. Sin descarga.

Ejercicios

1. Construye una ruta multiplataforma. Dado Path.home() / 'datos' / '2026' / 'enero.csv', imprime cómo se ve en Windows vs Unix.

2. Lista CSVs. En una carpeta con archivos mixtos (.csv, .txt, .py), lista solo los .csv ordenados por tamaño.

3. Búsqueda recursiva. En un árbol de carpetas, encuentra todos los .py que contengan la palabra TODO en su contenido.

4. Escribe + lee. Genera 3 archivos txt con write_text, léelos con read_text, concaténalos en uno solo.

5. Ruta del script. Escribe un script que cargue un dataset que vive al lado del script (no del cwd), usando Path(__file__).parent / 'data.csv'.

Homework verificable

Script inventario.py que recibe un directorio y produce un reporte CSV con: nombre, tamaño_bytes, extensión, última_modificación para cada archivo recursivamente, usando solo pathlib (no os).

Criterio de aceptación: El script corre tanto en Windows como en Linux/macOS sin cambios.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
FileNotFoundError aunque el archivo existe	Estás usando ruta relativa y el cwd no es
PermissionError al escribir	Carpeta read-only, OneDrive sincronizando,
mkdir() falla si el directorio ya existe	Default es exist_ok=False. Fix: p.mkdir(pa
Mezclo os.path y pathlib	Algunos funciones esperan strings (open(),
glob('*.CSV') no encuentra archivo.csv	Case-sensitive en Linux, insensible en Win
ImportError: Missing optional dependency '	pandas delega Parquet/Arrow a pyarrow (o f

Preguntas frecuentes

¿pathlib o os.path?

pathlib para código nuevo. os.path es la API funcional vieja (strings + funciones); pathlib es OO y mucho más legible. Solo usa os.path para compatibilidad con código viejo.

¿Cómo evito el típico 'C:/Users/...' vs '/home/...' cross-platform?

No hardcodees rutas absolutas. Usa Path.home(), Path(__file__).parent, tempfile.gettempdir(). Y siempre Path + /, nunca strings concatenados.

¿Cómo leo un CSV grande con pathlib?

Pathlib es para paths, no parsing. Combina: pd.read_csv(Path('data') / 'big.csv'). El Path se convierte a string automáticamente.

¿Path('a') / 'b/c' o Path('a') / 'b' / 'c'?

Ambas funcionan: Path parsea separadores. Pero la primera es menos explícita; prefiere la segunda.

¿shutil o pathlib para mover/copiar?

shutil para operaciones recursivas (copytree, rmtree, move) — pathlib solo cubre operaciones simples. Combinable: shutil.copy(src_path, dst_path) acepta Path directo.

¿Cuándo dejo de usar CSV?

Regla práctica: CSV solo para intercambio con no-técnicos (alguien lo va a abrir en Excel o mandarlo por mail). Para cualquier dataset >100 MB, o uno que vas a releer más de una vez en tu pipeline, pasalo a Parquet: ahorrás disco, ganás velocidad de lectura y no perdés tipos. Si lo único que querés es cachear un DataFrame entre dos scripts de la misma máquina, usá Feather.

Referencias

- pathlib docs
- PEP 428 — Object-oriented filesystem paths
- Slatkin, Effective Python 2e — ítem 38 Use Pathlib instead of os.path.
- Apache Arrow / Parquet

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 012 — Clase 012 — Logging

> *Parte: 0 — Prerrequisitos · Fuente: Python Tutorial logging HOWTO · The Pragmatic Programmer — "Programming by Coincidence".*

> *Duración estimada: 60 min.*

Objetivo

Que el alumno deje de usar print para debug y aprenda el módulo logging estándar: niveles (DEBUG/INFO/WARNING/ERROR/CRITICAL), handlers (consola, archivo), formatters, y configuración por módulo. Es la diferencia entre código que se debuggea reiniciando el notebook y código que se debuggea leyendo logs.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Diferenciar los 5 niveles de logging y cuándo usar cada uno.
2. Configurar un logger con logging.basicConfig y entender por qué basicConfig solo funciona una vez.
3. Crear loggers por módulo con logging.getLogger(__name__).
4. Agregar handlers: uno a consola (INFO+), otro a archivo (DEBUG+).
5. Formatear logs con timestamp, módulo y nivel.

Temas

#	Tema	Por qué importa
1	print vs logging	print() es output; logging es observabilid
2	Niveles: DEBUG/INFO/WARNING/ERROR/C	Filtran qué se ve sin tocar código.
3	Logger jerárquico por módulo	getLogger(__name__) para herencia natural.
4	Handlers: consola, archivo, rotating	Mismo log → múltiples destinos.
5	Formatters	Timestamp + nivel + módulo + mensaje.
6	logging.basicConfig y sus límites	Solo afecta el primero; preferir config ex

Definiciones y características

Logger

: Punto de entrada para emitir logs. Se obtiene con logging.getLogger(__name__) — esto crea un logger nombrado por el módulo. Característica clave: los loggers son jerárquicos (separados por .); la config de root propaga a hijos.

Handler

: Define a dónde van los logs (consola, archivo, syslog, sentry…). Un logger puede tener N handlers. Cada handler tiene su propio nivel y formatter.

Formatter

: Define cómo se renderiza el log: '%(asctime)s [%(levelname)s] %(name)s: %(message)s'. Campos comunes: asctime, levelname, name (logger), message, funcName, lineno, pathname.

Nivel (DEBUG/INFO/WARNING/ERROR/CRITICAL)

: Severidad ascendente. Filtran qué se emite. DEBUG: detalle interno; INFO: progreso normal; WARNING: algo raro pero no fatal; ERROR: operación falló; CRITICAL: sistema no puede continuar.

dictConfig

: Forma declarativa de configurar logging desde un dict (o YAML/JSON). Más mantenible que llamadas basicConfig/addHandler dispersas. Convención: una sola llamada en el entrypoint.

Dataset / recursos

Genera un log file de demo. Sin descarga.

Ejercicios

1. Reemplaza prints. Toma una función con 5 prints y conviértelos a logger con niveles apropiados.

2. Logger por módulo. Crea 2 archivos .py que cada uno usa getLogger(__name__). Configura el root logger una vez; verifica que ambos heredan.

3. Handler doble. Configura: consola = INFO+, archivo app.log = DEBUG+. Genera 5 logs de niveles distintos y verifica qué aparece en cada destino.

4. Formato con timestamp. Cambia el formato a '%(asctime)s [%(levelname)s] %(name)s: %(message)s'. Inspecciona output.

5. Logger en notebook. Pelea con basicConfig no recordando estado entre reinicios — usa dictConfig o force=True.

Homework verificable

Notebook + 2 módulos .py que importan y loguean. Un logging_config.py con dictConfig que define: consola (INFO+, formato corto) y app.log (DEBUG+, formato verbose con timestamp). El notebook ejecuta funciones que generan logs de distintos niveles desde ambos módulos. Adjunta el app.log resultante.

Criterio de aceptación: app.log contiene timestamp y módulo correcto en cada línea; consola filtra DEBUG.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
logging.basicConfig(...) no tiene efecto	basicConfig solo aplica si root no tiene h
Logs duplicados (cada mensaje aparece 2 ve	Configuraste el mismo handler dos veces (c
log.debug(f'valor: {expensive_call()}') si	El f-string se construye antes de pasar a
Mi logger emite en INFO pero quiero ver DE	El nivel está en handler o logger raíz. Fi
logging rompe en multiprocessing	Handlers no son fork-safe. Fix: en cada pr

Preguntas frecuentes

¿print o logging?

logging siempre en código que vivirá >1 día. print solo para REPL/scripts one-shot. Logging te da niveles, timestamps, módulo origen, múltiples destinos, integración con observabilidad.

¿Dónde configuro logging?

Una sola vez en el entrypoint (__main__, app.py, cli.py). Cada módulo solo hace log = logging.getLogger(__name__); nunca llama a basicConfig/addHandler desde un módulo importable.

¿Por qué getLogger(__name__)?

Crea un logger jerárquico nombrado por el módulo. Permite silenciar uno específico (logging.getLogger('mi_app.db').setLevel(WARNING)) sin tocar el resto. Pattern estándar.

¿Cómo formato un dict/object en el mensaje?

log.info('user=%s data=%s', user_id, data) (mejor que f-string por el lazy). Para JSON estructurado, usa python-json-logger o stdlib con custom formatter.

¿logging propaga al root logger?

Por default, sí — cada logger propaga al padre hasta root. Si configuras handlers en root y en hijos, verás el mensaje dos veces. Fix: logger.propagate = False en el hijo, o solo configura root.

Referencias

- Logging HOWTO
- Logging Cookbook
- logging.config — dictConfig

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 013 — Clase 013 — Type hints y mypy

> *Parte: 0 — Prerrequisitos · Fuente: Fluent Python 2e cap. 8 (Type Hints in Functions) · typing docs · mypy docs.*

> *Duración estimada: 75 min.*

Objetivo

Que el alumno anote tipos en sus funciones y dataclasses — no por dogma, sino porque permiten que el IDE autocomplete bien, que mypy detecte bugs antes de runtime, y que el lector entienda la intención. Tipos como documentación verificable.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Anotar funciones con tipos en parámetros y retorno (def f(x: int) -> str).
2. Usar tipos compuestos: list[int], dict[str, float], tuple[int, str], Optional[X], X | None.
3. Definir tipos personalizados con TypeAlias y Protocol (structural typing).
4. Ejecutar mypy sobre código y interpretar sus errores.
5. Reconocer cuándo type hints aportan (APIs públicas, data classes) y cuándo no (notebooks exploratorios).

Temas

#	Tema	Por qué importa	
1	Sintaxis básica: x: int, -> bool	Solo anotaciones — no afectan ru	
2	Tipos compuestos modernos (3.9·	Sin from typing import List.	
3	Optional[X] y `X	None` (3.10+)	Cuando algo puede ser None.
4	Literal, TypedDict, Protocol	Tipos avanzados útiles.	
5	mypy: instalar y correr	Static type checker.	
6	reveal_type(x) y # type: ignore	Diagnóstico y escape hatch.	
7	Cuándo SÍ y cuándo NO	API pública sí; notebook exploratc	

Definiciones y características

Type hint (annotation)

: Anotación de tipo en signature o variable: def f(x: int) -> str. Python NO verifica en runtime — son metadata leída por IDE/linter/mypy. Disponibles en f.__annotations__.

Optional[X] / X | None

: Indica que el valor puede ser X o None. Optional[X] ≡ Union[X, None] ≡ X | None (PEP 604, 3.10+). Usa la sintaxis con | en código nuevo.

TypedDict

: Esquema para diccionarios: class PersonaDict(TypedDict): nombre: str; edad: int. Permite tipear dicts que vienen de JSON/API sin convertirlos a dataclass.

Literal

: Restringe a un conjunto de valores: Literal['asc', 'desc'] solo acepta esas 2 strings. Útil para flags, modos,

enums simples.

Protocol (structural typing)

: Define interfaz por estructura (duck typing tipado): class TienePromedio(Protocol): def promedio(self) -> float: .... Cualquier clase con .promedio() la satisface, sin heredarla.

mypy

: Static type checker oficial. Lee tu código, sigue las anotaciones, reporta inconsistencias antes de ejecutar. Modo --strict lo hace exigente; recomendado para librerías.

Dataset / recursos

Funciones de ejemplo en el notebook. Sin descarga.

Ejercicios

1. Anota una función. Toma una función de los ejercicios de clase 008 (sin tipos) y anótala completa.

2. Optional vs default. Distingue def f(x: int = 0) (default 0) de def f(x: int | None = None) (puede no haber valor).

3. TypedDict. Define class PersonaDict(TypedDict) con nombre: str, edad: int. Úsala como tipo de un parámetro.

4. Corre mypy. Instala mypy, créate un archivo con un bug de tipo intencional (def f(x: int) -> str: return x + 1) y corre mypy archivo.py. Lee y explica el error.

5. Protocol. Define class TienePromedio(Protocol) con método promedio() -> float. Acepta cualquier clase que lo implemente (duck typing tipado).

Homework verificable

Repo con un módulo analytics.py (5+ funciones completamente anotadas), pyproject.toml que incluye mypy en [tool.mypy] con strict = true, y screenshot/log de mypy analytics.py sin errores.

Criterio de aceptación: mypy --strict corre sin errores ni warnings. Tipos consistentes y precisos.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar	
from typing import List da deprecation war	Python 3.9+ usa lowercase: list[int] en ve	
Optional[int] = 0 confunde	Optional[int] admite None. Si tu default e	None = None` (admite None).
mypy se queja "Cannot find module 'libreri	Lib sin type stubs publicados. Fix: pip in	
Anoté pero mypy no encuentra errores	mypy no se llamó. Fix: mypy archivo.py. No	
reveal_type(x) no existe en runtime	Es exclusivo de mypy: lo lees como output	

Preguntas frecuentes

¿Tipo hints son obligatorios?

No — Python no los verifica. Pero son fuertemente recomendados en: APIs públicas (funciones que importan otros), librerías, código compartido. En notebooks exploratorios, casi nunca aportan.

¿Slow my code los type hints?

No — son metadata, no impactan runtime. from __future__ import annotations además las hace lazy (strings,

evaluadas solo si introspeccionas).

¿mypy en CI: estricto o permisivo?

Empieza permisivo (sin --strict), arregla lo obvio, luego activa --strict gradualmente. Si arrancas estricto en un repo viejo, te ahogas en errores y termina ignorado.

¿Y si no sé qué tipo poner?

Any (de typing) es válido — desactiva el check para ese caso. Mejor que mentir con un tipo incorrecto. Convención: comentario # TODO: tipo correcto para revisión futura.

¿Pydantic vs dataclass + type hints?

dataclass + hints: tipos solo a nivel mypy, no en runtime. Pydantic: valida en runtime (parsing de JSON con coerción y errores legibles). Para input externo (API, config) → Pydantic. Para internal record → dataclass.

Referencias

- Ramalho, Fluent Python 2e — cap. 8.
- typing docs
- mypy docs
- PEP 484 — Type Hints
- PEP 604 — X | Y syntax

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 014 — Clase 014 — NumPy: tipos, creación, atributos

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, Python Data Science Handbook, cap. 2 — Introduction to NumPy, §§ 2.1–2.2.*

> *Duración estimada: 75 min.*

Objetivo

Que el alumno entienda el modelo mental de un ndarray — bloque contiguo de memoria con shape, dtype y strides — y sepa crear arrays de las 6 formas más útiles (array, zeros, arange, linspace, random, desde lista). Sin este modelo, todo el rendimiento de NumPy parece magia.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Explicar por qué ndarray es 50–100× más rápido que list (memoria contigua + dtype fijo + sin overhead Python).
2. Crear arrays con np.array, np.zeros, np.ones, np.full, np.arange, np.linspace.
3. Inspeccionar un array con shape, dtype, ndim, size, nbytes, itemsize.
4. Cambiar dtype explícitamente con astype y entender promociones implícitas (int + float = float).

5. Generar arrays aleatorios reproducibles con np.random.default_rng(seed).

Temas

#	Tema	Por qué importa
1	ndarray: memoria contigua + dtype fijo	Lo que lo hace rápido.
2	Creación: array, zeros, arange, linspace	Las 6 formas más usadas.
3	dtype: int8/16/32/64, float32/64, bool	Memoria y precisión.
4	Atributos: shape, dtype, ndim, size, nbyte	Diagnóstico instantáneo.
5	astype y promoción de tipos	El bug clásico de overflow int8.
6	random moderno: default_rng(seed)	El API legacy np.random.seed está deprecat

Definiciones y características

ndarray

: Estructura de datos central de NumPy. Bloque contiguo de memoria con dtype fijo + metadata (shape, strides). 50–100× más rápido que list por evitar el overhead de PyObject por elemento y permitir vectorización SIMD.

dtype

: Tipo de elemento del array: int8/16/32/64, uint8 (imágenes RGB), float32/64, bool, complex64/128. Determina memoria por elemento (itemsize) y precisión/rango.

shape

: Tupla con tamaño de cada dimensión: (10,) 1D, (3, 4) matriz, (2, 3, 4) tensor 3D. len(shape) = ndim. prod(shape) = size (total elementos).

strides

: Bytes que el array salta para moverse 1 paso en cada dimensión. Permite vistas eficientes sin copiar memoria (transpose, slicing). Detalle interno pero útil para entender por qué algunas operaciones son gratis.

Generator (random)

: API moderno para aleatoriedad: rng = np.random.default_rng(seed). Reemplaza al legacy np.random.seed() + funciones globales. Características: PCG64 (mejor algoritmo), múltiples generadores independientes, API consistente.

Promoción de tipo

: Cuando operas arrays de dtypes distintos, NumPy promueve al más amplio: int + float = float, int8 + int16 = int16. La regla evita pérdida silenciosa pero puede generar overflow en otros casos (clase: overflow int8).

Dataset / recursos

Sintético: arrays generados en el notebook (escalares, matrices, aleatorios reproducibles). Sin descarga.

Ejercicios

1. Memoria. Crea list(range(1_000_000)) y np.arange(1_000_000). Compara sys.getsizeof y arr.nbytes. Calcula el ratio.

2. Las 6 formas. Crea: vector 100 ceros, matriz 5×5 unos, vector 0..1 con 50 puntos equiespaciados, matriz 3×3 de 7s, vector de 100 aleatorios uniformes [0,1).

3. Bug de dtype. Crea np.array([100, 200, 50], dtype=np.int8) y suma 200 a cada elemento. Observa el

resultado y explica.

4. Diagnóstico. Dado un array, escribe una función que imprima shape, dtype, ndim, size, nbytes y memoria humana (KB/MB).

5. Random reproducible. Genera 1000 normales N(0,1) con seed=42. Calcula media y std. Repite — debe dar exactamente lo mismo.

Homework verificable

Notebook que: (a) compara memoria list vs ndarray para N=1M con tabla; (b) crea las 6 formas y reporta dtype default de cada una; (c) reproduce el bug de overflow int8 con explicación; (d) función info(arr) con diagnóstico completo.

Criterio de aceptación: El ratio memoria list/ndarray es >5×. La función info reporta todos los atributos.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
OverflowError silencioso con int8/uint8	NumPy no lanza excepción — hace wrap-aroun
np.array([1, 2, 'x']) queda como dtype='<U	NumPy promueve TODO a string para ser homo
np.empty(3) con basura en vez de ceros	empty NO inicializa — más rápido que zeros
Reproduzco un experimento con seed y da re	Estás usando el API legacy (np.random.seed
np.arange(0.1, 1.0, 0.1) no incluye 1.0	Floats — el último step da 0.9999... por i

Preguntas frecuentes

¿np.array([1,2,3]) o np.asarray([1,2,3])?

asarray no copia si ya es ndarray (más eficiente); array siempre copia por default. Usa asarray cuando aceptas cualquier 'array-like' y no necesitas garantizar copia.

¿float32 o float64?

Default es float64 — máxima precisión. float32 cuando trabajas con redes neuronales en GPU (la mitad de memoria, suficiente para gradientes), imágenes, o necesitas duplicar la velocidad de IO.

¿Cuándo int32 y cuándo int64?

Default depende del OS (int64 Unix/macOS, int32 Windows pre-numpy 2.0). En 2026, NumPy 2+ usa int64 en todas las plataformas. Solo bajes a int32 si memoria es crítica y sabes que tus valores caben.

¿np.random.seed(42) ya no se usa?

Funciona pero está deprecated en favor del nuevo API. Razones: estado global (peligroso), Mersenne Twister (lento), no permite múltiples streams independientes. Usa default_rng(seed).

¿Por qué arr.nbytes no coincide con sys.getsizeof(arr)?

nbytes cuenta solo los datos (size * itemsize). getsizeof cuenta también el header del objeto ndarray (~100 bytes). Para arrays grandes la diferencia es despreciable.

Referencias

- VanderPlas, cap. 2, §§ 2.1–2.2 Understanding Data Types + The Basics of NumPy Arrays.
- NumPy user guide — Array creation
- NumPy dtypes

- Generator random API

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 015 — Clase 015 — NumPy: ufuncs y vectorización

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 2, § 2.3 Computation on NumPy Arrays: Universal Functions.*

> *Duración estimada: 75 min.*

Objetivo

Que el alumno abandone los for loops sobre arrays NumPy y use ufuncs (universal functions) para operaciones elementwise — la fuente real del speedup. Ufuncs son C compilado vectorizado; un for Python sobre array es lo peor de ambos mundos.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Identificar una ufunc (np.add, np.multiply, np.sin, np.exp, np.log, comparadores).
2. Reemplazar un for+append por una expresión vectorizada y medir el speedup.
3. Usar el parámetro out= para escribir el resultado in-place (evita allocar memoria extra).
4. Combinar ufuncs con operadores aritméticos (+, -, , /, *).
5. Reconocer las trampas de la vectorización (overflow, NaN propagación, división por cero).

Temas

#	Tema	Por qué importa
1	¿Qué es una ufunc?	Función C vectorizada elementwise.
2	Ufuncs unarias y binarias	np.exp(x) vs np.add(x, y).
3	Operadores → ufuncs	a + b ≡ np.add(a, b).
4	out= para in-place	Memoria O(1) extra.
5	Trampas: overflow, NaN, inf, división por	NumPy avisa pero no para.
6	np.where(cond, a, b)	Ternario vectorizado.

Definiciones y características

Ufunc (universal function)

: Función NumPy implementada en C que opera elementwise y vectorizada (SIMD cuando posible). Características: rápida (10-100× vs Python), broadcasting automático, soporta out= para in-place.

Vectorización

: Operar sobre arrays completos en vez de loops Python: arr 2 en vez de [x2 for x in arr]. La operación corre en C compilado sobre memoria contigua, sin overhead del intérprete por elemento.

In-place (out=)

: Escribir el resultado de una ufunc en un array existente, sin allocar memoria nueva: np.multiply(a, 2, out=a). Útil con arrays grandes donde la copia temporal duplicaría la memoria pico.

Propagación de NaN

: Cualquier operación que tenga NaN como input produce NaN. np.array([1, np.nan, 3]).sum() → nan. Para ignorar usa variantes nan*: nansum, nanmean, nanmedian.

np.where(cond, a, b)

: Ternario vectorizado: para cada elemento, si cond es True usa a, si no usa b. Equivale a [a if c else b for c, a, b in zip(cond, a, b)] pero ~100× más rápido.

Dataset / recursos

Sintético: arrays grandes para benchmark. Sin descarga.

Ejercicios

1. Benchmark. Calcula [xx + 2x + 1 for x in range(1_000_000)] vs arrarr + 2arr + 1. Mide con %timeit.

2. Logaritmo y exponencial. Con np.exp y np.log, verifica que log(exp(x)) ≈ x para 1000 valores. Reporta el error máximo.

3. In-place vs alloc. arr = arr * 2 + 1 vs np.multiply(arr, 2, out=arr); np.add(arr, 1, out=arr). Compara tracemalloc.

4. np.where ternario. Dado un array de notas, crea otro array con 'aprobado' si nota >= 4, 'reprobado' si no.

5. Trampa NaN. Crea np.array([1, 2, np.nan, 4]).sum() y .mean(). Compara con np.nansum y np.nanmean.

Homework verificable

Notebook: (a) reescribe 3 loops como expresiones vectorizadas + tabla con %timeit (3 N distintos); (b) demuestra out= con tracemalloc; (c) usa np.where para clasificar datos; (d) maneja NaN con nansum/nanmean y compara con propagación.

Criterio de aceptación: Speedup >50× en N=1M. out= muestra memoria ≈ 0 extra. NaN-handling correcto.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
for i in range(len(arr)): arr[i] = ... es	Loops Python sobre array NumPy = lo peor d
RuntimeWarning: divide by zero / invalid v	NumPy avisa pero no para: 1/0 → inf, 0/0 →
out= con dtype incompatible	np.add(int_arr, 0.5, out=int_arr) falla —
Resultado de np.where no es lo esperado	Los 3 args se evalúan completos: np.where(
arr.sum() da NaN y no sé por qué	Hay un NaN escondido en el array. Fix: pri

Preguntas frecuentes

¿Cuánto más rápido es vectorizar?

Típicamente 50-100× para arrays de 1M elementos. Para arrays pequeños (<100), la ganancia es menor o nula (overhead constante). Mide con %timeit, no asumas.

¿arr + 1 o np.add(arr, 1)?

Equivalentes. Operadores son sintaxis dulce sobre ufuncs. Usa np.add(...) cuando necesitas out= (in-place) o where= (mask).

¿NumPy aprovecha mi GPU?

No — solo CPU. Para GPU: CuPy (drop-in replacement), PyTorch tensors, JAX. NumPy 2 está mejorando vectorización CPU (SIMD wider, BLAS) pero sigue siendo CPU.

¿Por qué arr 2 es más rápido que arr arr?*

Suelen empatar (** también es ufunc). Para potencias enteras pequeñas (2, 3), NumPy a veces usa atajos. Mide con %timeit en tu caso específico.

¿np.where o boolean mask?

Mask (arr[cond] = valor) si vas a modificar in-place o filtrar (arr[arr>0]). np.where(cond, a, b) si necesitas un array nuevo con dos valores posibles según condición.

Referencias

- VanderPlas, cap. 2 § 2.3 Computation on NumPy Arrays.
- NumPy ufuncs reference

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 016 — Clase 016 — NumPy: agregaciones

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 2 § 2.4 Aggregations: Min, Max, and Everything in Between.*

> *Duración estimada: 60 min.*

Objetivo

Que el alumno reduzca arrays a estadísticos (sum, mean, std, percentile, min, max) controlando el axis correcto — la fuente del 50% de los bugs de pandas/sklearn cuando alguien se confunde de eje. También: variantes nan* y reducciones acumulativas.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Calcular sum, mean, std, var, median, percentile sobre arrays.
2. Controlar el eje con axis=0 (a lo largo de filas, da resultado por columna) y axis=1 (a lo largo de columnas, da por fila).
3. Usar variantes nan* (nansum, nanmean, etc.) cuando hay datos faltantes.
4. Reducciones acumulativas con cumsum y cumprod.
5. Encontrar índice del min/max con argmin/argmax.

Temas

#	Tema	Por qué importa
1	Reducciones básicas	sum, mean, std, var, min, max, median, per
2	Eje: el bug más común	axis=0 reduce filas (resultado por columna
3	Variantes NaN-aware	nansum, nanmean, nanmedian, nanstd.
4	Acumulativas	cumsum, cumprod — útiles para series tempo
5	argmin/argmax	Posición del extremo.
6	all y any	Reducciones booleanas.

Definiciones y características

Agregación / reducción

: Operación que colapsa un array a menos dimensiones: sum, mean, std, min, max, argmax. Sin axis, reduce a un escalar; con axis=N, elimina la dimensión N.

axis=0 vs axis=1

: Regla: el axis que pasas es el que desaparece. En matriz (filas, cols): axis=0 colapsa filas → un valor por columna; axis=1 colapsa cols → un valor por fila. Mnemónico inverso al que muchos esperan.

argmin / argmax

: Devuelven el índice del extremo (no el valor). arr.argmax() = posición del máximo; arr[arr.argmax()] = valor máximo. Con axis= devuelven array de índices por fila/columna.

Variantes nan*

: Versiones que ignoran NaN en vez de propagarlo: nansum, nanmean, nanstd, nanmin, nanmax, nanmedian, nanpercentile, nanargmax. Útiles cuando los datos tienen missing.

Acumulativas (cumsum, cumprod)

: No colapsan — devuelven array de igual shape con valores acumulados hasta cada posición. Útiles para series temporales (precio acumulado, drawdown).

percentile / quantile

: Valor por debajo del cual cae el N% de los datos. np.percentile(arr, 50) = mediana. np.percentile(arr, [25, 50, 75]) = cuartiles.

Dataset / recursos

Sintético (matriz aleatoria 100×10 simulando "10 features × 100 muestras") generado en el notebook. Sin descarga.

Ejercicios

1. Promedio por columna. Dada matriz 100×4 de ventas (filas=día, cols=tienda), calcula la media por tienda y por día.

2. Estadísticos completos. Para un array de 1000 normales, reporta mean, std, median, p25, p75, min, max.

3. Con NaN. Inserta 50 NaN aleatorios en el array anterior. Compara mean (propaga) vs nanmean.

4. Cumsum. Genera array de retornos diarios aleatorios. Calcula el precio acumulado con cumprod(1+r).

5. Mejor tienda. Con la matriz del ejercicio 1, usa argmax(axis=0) para encontrar el día de mayor venta de cada tienda.

Homework verificable

Notebook con matriz simulada 365 días × 5 tiendas de ventas, reportando: media/std por tienda, mejor y peor día de cada tienda, cumsum total anual, % de días con NaN simulados (20 aleatorios) usando variantes nan*.

Criterio de aceptación: Eje correcto en todas las agregaciones; valores reproducibles con seed.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
M.sum(axis=0) da resultado por columna y e	Confusión clásica. Regla: axis=0 reduce fi
arr.mean() da NaN aunque solo hay un par d	Cualquier NaN propaga. Fix: np.nanmean(arr
np.argmax(matriz) devuelve un solo número	Sin axis, aplana el array primero y devuel
np.percentile([1,2,3,4], 50) da 2.5 no 2	Interpolación lineal por default. Si quier
cumsum de floats acumula error de redondeo	Sumas múltiples introducen error numérico.

Preguntas frecuentes

¿arr.sum() o np.sum(arr)?

Equivalentes. Método del array (.sum()) es más legible en cadenas (arr.clip(0).sum()). Función (np.sum) acepta también listas (no solo ndarray).

¿Cómo recuerdo qué axis colapsa cuál?

El axis que pasas es el que desaparece. Si shape es (3, 4) y haces sum(axis=0), queda (4,) — desapareció dim 0. Si axis=1, queda (3,).

¿std usa N o N-1?

Default ddof=0 (divide por N — desviación poblacional). Para desviación muestral (N-1), arr.std(ddof=1). Pandas y scipy.stats usan N-1 por default — cuidado al comparar.

¿np.median ignora NaN?

No — usa np.nanmedian. Mismo patrón que mean/nanmean.

¿Hay una agregación 'top-3' built-in?

No directa. Usa np.partition(arr, -3)[-3:] para los 3 mayores (más rápido que sort completo). Si necesitas ordenados, .sort() después.

Referencias

- VanderPlas, cap. 2 § 2.4.
- NumPy statistics functions

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 017 — Clase 017 — NumPy: broadcasting

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 2 § 2.5 Computation on Arrays: Broadcasting.*

Duración estimada: 75 min.

Objetivo

Que el alumno internalice las reglas de broadcasting — el mecanismo por el que NumPy operó arrays de shapes distintos sin copiar datos. Es lo que hace que M - M.mean(axis=0) centrado por columna sea una línea, no un bucle anidado.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Recitar las 3 reglas de broadcasting (alinea por la derecha, dim 1 estira, falla si no es 1 ni igual).
2. Predecir la shape del resultado de una operación entre arrays de shapes distintos.
3. Centrar y escalar matrices por fila/columna sin loops.
4. Usar np.newaxis (o None) para promover un vector a matriz fila/columna.
5. Diagnosticar un ValueError: operands could not be broadcast together leyendo las shapes.

Temas

#	Tema	Por qué importa
1	Las 3 reglas	Padding a la derecha, dim 1 estira, error
2	Vector + matriz	Vector como fila o como columna.
3	np.newaxis / None	Insertar eje de tamaño 1.
4	Caso canónico: centrar/escalar	X - X.mean(axis=0) y (X - μ) / σ.
5	Outer product sin loop	a[:, None] * b[None, :].
6	ValueError común: "operands could not be b	Cómo leerlo.

Definiciones y características

Broadcasting

: Mecanismo por el que NumPy opera arrays de shapes distintos sin copiar memoria, estirando virtualmente las dimensiones de tamaño 1. Lo que hace posible X - X.mean(axis=0) (centrado por columna) en una línea sin loops.

Regla 1 — padding por la izquierda

: Si los arrays tienen distinta cantidad de dimensiones, la shape del menor se rellena con 1s a la izquierda. (4,) operado con (3, 4) se trata como (1, 4) vs (3, 4).

Regla 2 — estirar dim 1

: En cada dimensión donde los tamaños difieren, si uno es 1 se estira al otro. (3, 1) y (3, 4) → ambos (3, 4) (la primera se estira en eje 1).

Regla 3 — fallo

: Si en alguna dimensión los tamaños son distintos y ninguno es 1, lanza ValueError: operands could not be broadcast together. No hay forma de inferir qué hacer.

np.newaxis (alias None)

: Inserta una dimensión de tamaño 1 donde lo pongas. v[:, None] convierte vector (3,) en columna (3, 1). Crítico para forzar broadcasting en la dirección correcta.

Outer product vía broadcasting

: a[:, None] * b[None, :] produce matriz (len(a), len(b)) con todos los productos par a par — equivalente a np.outer(a, b) pero usando broadcasting puro.

Dataset / recursos

Sintético: matriz de features 100×5 para estandarización. Sin descarga.

Ejercicios

1. Predice antes de ejecutar. Para shapes (3,), (3,1), (1,3), (2,3,4) × (4,), predice la shape del resultado. Verifica.

2. Estandariza features. Matriz 100×5 aleatoria. Resta media por columna y divide por std por columna en una línea.

3. Outer product. Vectores a=[1,2,3], b=[10,20,30,40]. Calcula la matriz outer (3×4) sin np.outer, solo broadcasting.

4. Distance matrix. Dados 5 puntos 2D, construye matriz 5×5 de distancias euclídeas entre pares — sin cdist, solo broadcasting.

5. Diagnostica error. Intenta np.ones((3,4)) + np.ones((4,3)). Lee el ValueError y explica.

Homework verificable

Notebook que: (a) predice shapes de 4 operaciones broadcasting y verifica; (b) estandariza una matriz feature por columna en una línea; (c) construye distance matrix de 100 puntos sin loop; (d) provoca y explica un error de broadcasting.

Criterio de aceptación: Las predicciones coinciden. Estandarización: media≈0, std≈1 por columna.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
ValueError: operands could not be broadcas	Las shapes alineadas por la derecha no son
Resté M.mean(axis=0) y los promedios queda	mean(axis=0) devuelve shape (n_cols,) — se
a + b con shapes (3,) y (3,) da escalar (s	No — da array (3,) elementwise. Producto p
Memoria explota en una operación 'inocente	X[:, None, :] - X[None, :, :] produce arra
a[None] + b no se broadcastea como espero	a[None] añade dim al inicio. Quizás quería

Preguntas frecuentes

¿Cómo predigo la shape del resultado?

(1) Alinea las shapes por la derecha. (2) En cada columna alineada: si son iguales o uno es 1, OK; si no, error. (3) Resultado: por dimensión, toma el max de las dos.

¿Broadcasting copia memoria?

No — es virtual. NumPy itera con strides 0 en las dims estiradas. Por eso es tan eficiente: cero alloc extra (excepto el array resultado).

¿X - X.mean(0) o X - X.mean(0, keepdims=True)?

Para 2D ambos funcionan (broadcasting alinea). En 3D+, keepdims=True preserva la dimensión como 1 y evita confusiones. Recomendado en general.

¿np.newaxis o None?

Aliases — arr[:, None] y arr[:, np.newaxis] son idénticos. None es más conciso; muchos prefieren np.newaxis por explicitud.

¿Qué hago si broadcasting no me sirve?

Operaciones que no se ajustan a las reglas: usa np.einsum (más expresivo), np.tensordot, o reshape explícito. Como último recurso, loop Python — pero busca librería específica antes.

Referencias

- VanderPlas, cap. 2 § 2.5 Broadcasting.
- NumPy broadcasting docs

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 018 — Clase 018 — NumPy: boolean masks y fancy indexing

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 2, §§ 2.6–2.7 Comparisons, Masks, and Boolean Logic + Fancy Indexing.*

> *Duración estimada: 75 min.*

Objetivo

Que el alumno seleccione, filtre y modifique sub-arrays de tres formas: slicing (visto), máscaras booleanas (arr[arr > 0]) y fancy indexing (arr[[0, 3, 5]]). Saber cuál devuelve vista vs copia y cuándo cada uno es la herramienta correcta.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Filtrar elementos con máscaras booleanas: arr[arr > 0], arr[(a > 0) & (a < 10)].
2. Combinar máscaras con &, |, ~ — NO con and/or (no vectorizan).
3. Seleccionar por índices con fancy indexing: arr[[0, 3, 5]] o arr[idx_array].
4. Modificar in-place con máscara: arr[arr < 0] = 0 (clipping).
5. Diferenciar vista vs copia: slicing es vista; fancy indexing y máscara son copia.

Temas

#	Tema	Por qué importa	
1	Comparaciones elementwise → a	arr > 0 no devuelve un bool, devu	
2	np.count_nonzero, np.sum sobre l	Cuenta cuántos True.	
3	Combinar máscaras con &, `	, ~`	Operadores bitwise — no and/or.
4	Fancy indexing con array de índic	Selección no contigua.	
5	Vista vs copia	Slicing = vista; mask/fancy = copia	

6	np.where(cond) (sin alternativas)	Devuelve índices donde se cumpl	

Definiciones y características

Boolean mask

: Array de bools del mismo shape que el original. arr[mask] extrae solo los elementos donde mask es True. Devuelve un nuevo array (copia, no vista).

Fancy indexing

: Indexar con array de enteros (índices arbitrarios, posiblemente no contiguos). arr[[0, 3, 5]] selecciona esas 3 posiciones. Devuelve copia.

Vista (view) vs copia (copy)

: Slicing (arr[:5]) → vista (mismo storage, mutarla muta el original). Mask / fancy → copia (storage independiente). Fuente del 70% de los bugs sutiles.

Operadores bitwise vs lógicos

: Para combinar masks: &, |, ~ (bitwise, vectorizados, elementwise). NO uses and, or, not — son escalares Python y dan ValueError: truth value of an array is ambiguous.

np.where(cond) 1-arg

: Sin alternativas, devuelve tupla de arrays de índices donde se cumple la condición. Diferente a np.where(cond, a, b) (ternario).

Dataset / recursos

Sintético: array de precipitación diaria (365 valores). Sin descarga.

Ejercicios

1. Cuenta días lluviosos. Dado array de 365 días con precipitación (mm), cuenta cuántos tuvieron >5mm.

2. Estadísticos por máscara. Calcula precipitación media solo en días lluviosos (>0mm).

3. AND/OR combinados. Días entre 1 y 10 mm. Días <1 o >50 mm.

4. Clipping. Reemplaza valores negativos por 0 in-place (arr[arr < 0] = 0).

5. Vista vs copia. Demuestra con un experimento que arr[:5] modifica el original pero arr[arr > 0] no.

Homework verificable

Notebook con array sintético de 365 días de precipitación generado con seed. Calcula: (a) días lluviosos y su media, (b) días extremos (>50mm), (c) demo de vista vs copia, (d) clipping in-place, (e) índices del top 10 días más lluviosos con argsort.

Criterio de aceptación: Resultados reproducibles. Demo vista/copia muestra comportamiento opuesto.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar	
ValueError: The truth value of an array wi	Usaste and/or/not con arrays. Fix: &/`	/~ con paréntesis: (a > 0) & (a < 10)` (lo
Modifico arr[mask] y el original no cambia	Mask devuelve copia. Fix: para modificar i	
arr[idx1, idx2] con fancy indexing me da a	Si idx1 e idx2 son arrays del mismo length	
Slice modifica el original sin querer	subset = arr[:10]; subset[0] = 99 modifica	

Mask con shape distinto al array	Lanza IndexError: boolean index did not ma	

Preguntas frecuentes

¿Mask o fancy indexing?

Mask cuando la selección viene de una condición sobre los valores (arr[arr > 0]). Fancy cuando ya tienes los índices específicos (de un argsort, por ejemplo, o de business logic).

¿Cómo modifico múltiples elementos con mask?

arr[arr < 0] = 0 (clipping). Funciona para asignar escalar a todos los True, o array del mismo tamaño que la cantidad de Trues: arr[arr < 0] = -arr[arr < 0] (abs solo donde negativo).

¿arr[idx] con idx como booleano o entero?

NumPy distingue: bool array del mismo shape → mask; int array → fancy indexing. Lista Python de bools también funciona, pero ojo con [0, 1, 0, 1] que puede interpretarse como ints (índices 0 y 1) o bools — usa np.array(...) explícito si hay duda.

¿np.where(cond) o np.nonzero(cond)?

Idénticos cuando where se llama con un solo argumento. nonzero es más explícito del intent ("dónde NO es 0/False").

¿Cómo combino mask con np.where ternario?

np.where(cond, valor_si_true, valor_si_false) — ternario vectorizado. La diferencia con arr[cond] = X: where construye array nuevo; mask + asignación modifica in-place.

Referencias

- VanderPlas, cap. 2 §§ 2.6, 2.7.
- NumPy indexing user guide

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 019 — Clase 019 — NumPy: ordenamiento y búsqueda

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 2 § 2.8 Sorting Arrays · NumPy sorting reference.*

> *Duración estimada: 60 min.*

Objetivo

Que el alumno ordene arrays con criterio: sort vs argsort, ordenamiento por eje, partial sort con partition, y búsqueda binaria con searchsorted. Útil para top-K, rankings, alineación de series.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Ordenar con np.sort(arr) (devuelve copia) y arr.sort() (in-place).
2. Obtener índices del orden con argsort — base de top-K y rankings.
3. Ordenar por eje en matrices con axis=0 o axis=1.
4. Top-K eficiente con np.partition (no ordena completo, solo separa).
5. Búsqueda binaria con np.searchsorted en arrays ordenados (O(log n)).

Temas

#	Tema	Por qué importa
1	np.sort vs arr.sort()	Copia vs in-place.
2	argsort: el truco del top-K	Índices que ordenarían el array.
3	Ordenamiento por eje	Por fila o por columna.
4	np.partition para top-K	Más rápido que sort completo.
5	np.searchsorted — binaria O(log n)	Inserción en array ordenado.
6	np.unique	Únicos + opcionalmente cuentas.

Definiciones y características

np.sort vs arr.sort()

: np.sort(arr) devuelve copia ordenada (no muta). arr.sort() ordena in-place y retorna None. Mismo patrón que sorted(list) vs list.sort().

argsort

: Devuelve los índices que ordenarían el array. idx = arr.argsort(); ordenado = arr[idx]. Base de top-K, rankings, alineación entre arrays correlacionados.

np.partition

: Quick-select O(n): garantiza que los K menores quedan en las primeras K posiciones (no necesariamente ordenados entre sí), el resto después. Mucho más rápido que sort completo cuando solo necesitas top-K.

np.searchsorted

: Búsqueda binaria O(log n) en array ordenado. Devuelve el índice donde insertar un valor para mantener orden. Útil para calcular percentiles, bucketing.

np.unique

: Devuelve únicos ordenados. Con return_counts=True devuelve también las frecuencias — alternativa rápida a Counter para datos numéricos.

Dataset / recursos

Sintético: puntajes de 1M estudiantes. Sin descarga.

Ejercicios

1. Top-10. Dado array de 1M puntajes, obtén los 10 más altos. Compara np.sort()[-10:] vs np.partition.

2. Ranking. Con argsort, asigna a cada estudiante su ranking (1 = mejor).

3. Ordena matriz por columna. Matriz 10×5; ordena cada columna por su valor.

4. Mediana por bisect. Implementa una función que dado un valor v y un array ordenado, devuelve su posición percentil usando searchsorted.

5. np.unique con cuentas. Dado array de categorías, obtén valores únicos y sus frecuencias.

Homework verificable

Notebook con array de 100k puntajes: (a) top-100 con partition y benchmark vs sort completo; (b) ranking con argsort.argsort(); (c) percentil de un valor dado con searchsorted; (d) unique con return_counts y barplot top-10 categorías.

Criterio de aceptación: partition >10× más rápido que sort completo para N=100k y K=100.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Ordeno con arr.sort() y la variable queda	sort() es in-place — modifica arr y retorn
Top-K con sort()[-K:] es lento para N gran	Sort completo es O(N log N). Fix: np.parti
argsort da resultado raro con matriz	Sin axis, ordena cada fila/columna indepen
np.searchsorted da índice fuera del array	Si el valor es mayor que todos, devuelve l
np.unique(arr_2d) aplana el array	Por default, unique trabaja sobre array ap

Preguntas frecuentes

¿Cuándo argsort y cuándo sort?

Si solo necesitas los valores ordenados, sort. Si necesitas el orden para aplicarlo a otros arrays correlacionados (ej: ordenar nombres por puntajes), argsort te da los índices.

¿partition o heapq.nlargest?

partition para arrays NumPy (vectorizado, C). heapq.nlargest(K, lst) para listas Python. Para K muy pequeño (3-5) sobre N grande, similares; partition gana en arrays grandes.

¿Cómo ordeno por múltiples claves (lexicográfico)?

np.lexsort([clave_secundaria, clave_principal]) — devuelve índices. Atención: el orden es al revés (último argumento = key primaria).

¿np.unique preserva el orden de primera aparición?

No — siempre ordena. Para preservar orden de aparición: _, idx = np.unique(arr, return_index=True); arr[np.sort(idx)].

¿Existe equivalente a SQL ORDER BY x DESC?

arr[arr.argsort()][::-1] o arr[arr.argsort()[::-1]]. NumPy no tiene flag reverse= como sorted() Python — invierte tú.

Referencias

- VanderPlas, cap. 2 § 2.8.
- NumPy sorting reference

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 020 — Clase 020 — NumPy: álgebra lineal con numpy.linalg

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 2 § 2.9 Structured Arrays (referencia) · Numerical Linear Algebra (Trefethen & Bau).*

> *Duración estimada: 90 min.*

Objetivo

Que el alumno opere con vectores y matrices al nivel necesario para entender ML: producto punto, multiplicación matricial, inversa, sistema de ecuaciones (solve), descomposiciones (SVD, eigen). Saber cuándo no usar la inversa (lentitud + inestabilidad numérica).

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Multiplicar vectores y matrices con @ (operador moderno) y np.dot.
2. Resolver sistemas Ax = b con np.linalg.solve (NO con inv(A) @ b).
3. Calcular norma, determinante, rango, traza.
4. Computar SVD con np.linalg.svd y entender qué retorna.
5. Calcular eigenvalores/eigenvectores con np.linalg.eig / eigh (simétrica).

Temas

#	Tema	Por qué importa
1	@ operador (PEP 465): multiplicación matri	Reemplaza np.matmul.
2	Producto punto vs producto matricial	Vector·vector vs matriz·matriz.
3	Resolver sistemas: solve vs inv	Por qué NUNCA usar inv.
4	Norma, det, rank, trace	Diagnóstico estructural de matrices.
5	SVD — la factorización universal	Base de PCA, regresión lineal, recomendado
6	Eigen	Base de PCA conceptual.

Definiciones y características

Operador @

: Multiplicación matricial (PEP 465). A @ B ≡ np.matmul(A, B) ≡ A.dot(B). NO confundir con * (elementwise). Disponible Python 3.5+.

Producto punto vs producto matricial

: Punto (vector·vector → escalar): a @ b = sum(a*b). Matricial (matriz @ matriz → matriz): regla "fila por columna". Shapes: (m,n) @ (n,p) → (m,p).

np.linalg.solve(A, b)

: Resuelve Ax = b usando descomposición LU. Siempre preferible a inv(A) @ b: más rápido (no construye inversa), más estable numéricamente, menos memoria.

SVD (Singular Value Decomposition)

: Descomposición universal: $M = U \cdot \Sigma \cdot V^T$. U y V son ortogonales; Σ diagonal con valores singulares decrecientes ≥ 0. Base de PCA, recomendadores (matrix factorization), compresión, pseudo-inversa.

Eigen (eigenvalues/eigenvectors)

: Para A cuadrada, A v = λ v. eig general; eigh para matrices simétricas (más rápido, garantiza eigenvalores reales). Base de PCA conceptual.

Número de condición (cond)

: Ratio entre el valor singular más grande y el más pequeño. Mide sensibilidad de la solución a perturbaciones. cond > 1e10 matriz mal condicionada, solve perderá precisión.

Dataset / recursos

Sintético: matrices y vectores para los ejercicios. Sin descarga.

Ejercicios

1. Producto punto. Dados dos vectores 100-dim aleatorios, calcula np.dot(a, b) y verifica que coincide con sum(a*b).

2. Multiplicación matricial. (50, 30) @ (30, 20) → (50, 20). Verifica shapes y un elemento manualmente.

3. Resuelve sistema. Genera A = (5,5) aleatoria, b = (5,), resuelve Ax = b con solve. Verifica A @ x ≈ b.

4. Inv vs solve benchmark. Para A (1000,1000) y b (1000,), mide tiempo de inv(A) @ b vs solve(A, b). Reporta speedup.

5. SVD de matriz baja rank. Crea M = u @ v.T (rank 1). Calcula SVD y observa que solo el primer valor singular es no-cero.

Homework verificable

Notebook que: (a) compara inv(A) @ b vs solve(A, b) en tiempo Y precisión (np.allclose); (b) implementa regresión lineal cerrada $\beta = (X^TX)^{-1} X^Ty$ y luego con solve; (c) calcula SVD de una matriz y verifica M = U @ diag(s) @ Vt; (d) eigen de matriz de covarianza.

Criterio de aceptación: solve más rápido y más preciso que inv. SVD reconstruye la matriz dentro de tolerancia.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
A @ B falla con ValueError: shapes ... not	Las dimensiones internas no coinciden: (m,
Implementé inv(A) @ b y los resultados son	inv es inestable numéricamente para matric
np.linalg.solve lanza LinAlgError: Singula	det(A) ≈ 0 — el sistema no tiene solución
A * B da resultado raro y esperaba A @ B	Operador * es elementwise (Hadamard produc
SVD devuelve Vt no V	Por convención NumPy devuelve V^T (transpu

Preguntas frecuentes

¿@, np.matmul, o np.dot?

Para matriz×matriz son equivalentes — @ es el más legible. np.dot tiene comportamiento distinto para arrays >2D (no broadcasting); @ y matmul sí. Usa @ siempre que puedas.

¿eig o eigh?

eigh si la matriz es simétrica (covarianza, kernel matrices, métrica de distancias). Más rápido, garantiza eigenvalores reales. eig para matrices generales (no simétricas) — puede dar valores complejos.

¿Por qué np.linalg.det para chequear singularidad es mala idea?

El determinante es 0 o no-0 sin gradiente útil — para matrices grandes, det puede ser tan chico/grande que cause underflow/overflow numérico. Mejor: np.linalg.cond(A) — si > 1e10, problemática.

¿Cuándo necesito BLAS/LAPACK?

NumPy ya los usa por debajo (vía OpenBLAS o MKL). Si tu np.linalg.solve parece lento, instala MKL (pip install mkl) o usa la build de conda con mkl.

¿GPU para álgebra lineal?

CuPy (drop-in replacement de NumPy con CUDA), PyTorch tensors (.to('cuda')), o JAX. Para matrices >1000×1000 la GPU vale la pena.

Referencias

- VanderPlas cap. 2 (overview NumPy).
- numpy.linalg reference
- PEP 465 — @ operator
- Trefethen & Bau, Numerical Linear Algebra (1997) — fondo matemático.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 021 — Clase 021 — NumPy: aleatoriedad y semillas

> *Parte: 0 — Prerrequisitos · Fuente: Numerical Recipes cap. 7 (Random Numbers) · NumPy random.Generator docs.*

> *Duración estimada: 60 min.*

Objetivo

Que el alumno genere números aleatorios reproduciblemente con el API moderno (np.random.default_rng(seed)), use las distribuciones más comunes (uniforme, normal, Bernoulli, Poisson, exponencial), y entienda por qué la reproducibilidad es no-negociable en ciencia de datos.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Crear un Generator con np.random.default_rng(seed) y usarlo para reproducibilidad.
2. Generar muestras de uniforme, normal, integers, binomial, Poisson, exponencial.
3. Permutar y muestrear sin/con reemplazo con permutation y choice.
4. Reproducir un experimento exactamente con el mismo seed.
5. Saber por qué np.random.seed() (API legacy) es deprecated en favor de Generator.

Temas

#	Tema	Por qué importa
1	np.random.default_rng(seed)	El API moderno (Generator-based, PCG64).

2	Distribuciones continuas: uniform, normal,	Las más usadas en simulación.
3	Distribuciones discretas: integers, binomi	Conteos y procesos.
4	permutation y choice	Mezclar y muestrear.
5	Reproducibilidad: por qué importa	Misma cosa con mismo seed.
6	Múltiples generadores independientes	Evita interferencia entre experimentos.

Definiciones y características

Generador pseudoaleatorio (PRNG)

: Algoritmo determinístico que produce secuencia que parece aleatoria. Con la misma semilla (seed) produce la misma secuencia → reproducible. NumPy 2026 usa PCG64 por default.

Seed (semilla)

: Estado inicial del PRNG. Mismo seed → misma secuencia, siempre, en cualquier máquina. Es la base de la reproducibilidad científica.

np.random.default_rng(seed)

: API moderno (NumPy 1.17+). Devuelve Generator independiente — no toca estado global, múltiples generadores no se interfieren. Reemplaza a np.random.seed() + funciones globales (deprecated).

Distribuciones continuas comunes

: uniform(lo, hi), normal(μ, σ), exponential(scale), gamma(shape, scale), beta(a, b). Cada una con parámetros que controlan ubicación y dispersión.

Distribuciones discretas

: integers(lo, hi) (uniforme discreto), binomial(n, p) (éxitos en n trials), poisson(λ) (eventos en intervalo).

Bootstrap

: Técnica: resampleas con reemplazo del sample original B veces, recalculas el estadístico → obtienes distribución empírica del estimador. Sin asumir CLT ni normalidad.

Dataset / recursos

Sintético: simulación Monte Carlo. Sin descarga.

Ejercicios

1. Reproducibilidad. Crea 2 rngs con seed=42, genera 1000 normales con cada uno. Verifica que son idénticos.

2. Distribuciones. Genera 10000 muestras de: uniforme [0,1], normal(5,2), exponential(λ=1/3), poisson(λ=4). Calcula media y std empírica y compara con teórica.

3. Monte Carlo de π. Estima π lanzando puntos en un cuadrado 2×2 y contando cuántos caen dentro del círculo unitario. Compara con π real.

4. Bootstrap. Dado un sample de 30 valores, estima la distribución de la media por bootstrap (1000 resamples con reemplazo).

5. Permutación. Mezcla un array de 100 elementos con permutation. Verifica que es la misma cuando usas el mismo seed.

Homework verificable

Notebook con: (a) Monte Carlo de π con N=10k, 100k, 1M reportando error; (b) bootstrap de la media de un

sample (95% CI vs CLT); (c) demo de reproducibilidad con dos rngs; (d) tabla comparando momento empírico vs teórico para 4 distribuciones.

Criterio de aceptación: MC converge a π. Bootstrap CI similar al CLT. Reproducibilidad exacta.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Pongo seed pero el resultado cambia entre	Probablemente otra librería (sklearn, pyto
np.random.seed(42) no funciona como antes	Está deprecated en favor de default_rng. A
rng.choice(arr, size=N, replace=False) fal	Pediste más muestras únicas que elementos
Genero millones de números 'aleatorios' y	Estás materializando lista en RAM. Fix: si
Bootstrap CI parece muy estrecho	Pocos resamples (B). Fix: usa B ≥ 1000 par

Preguntas frecuentes

¿Por qué un generador independiente y no el global?

(1) No interfiere con libs que también usan random global. (2) Múltiples experimentos simultáneos sin conflicto. (3) API más limpia. (4) Algoritmo más rápido (PCG64). El legacy es solo por compatibilidad.

¿Mismo seed da mismos números en Linux/Windows/Mac?

Sí — PCG64 es determinístico cross-platform. La única diferencia podría venir de orden de operaciones float (paralelismo), pero default_rng es single-threaded.

¿Qué seed elegir?

Cualquier entero. 42 es convención (Hitchhiker's Guide). 0 funciona pero genera secuencias 'menos aleatorias' en los primeros bits con algunos PRNG (no PCG64). Lo importante es registrarlo para reproducir.

¿Bootstrap mejor que t-test?

Diferente caso. t-test: asume normalidad, da p-valor analítico. Bootstrap: sin asunción, da distribución empírica de cualquier estadístico (media, mediana, ratio). Para datos no-normales o estadísticos complejos, bootstrap gana.

¿Cuántas muestras necesito para Monte Carlo?

El error decrece como 1/√N. Para 1 decimal de precisión: N ≈ 100. Para 2 decimales: N ≈ 10k. Para 3: N ≈ 1M. Verifica convergencia haciendo N=100, 1k, 10k, 100k y mirando que el resultado se estabilice.

Referencias

- NumPy random.Generator
- NEP 19 — Random number generator policy
- Press et al., Numerical Recipes 3e — cap. 7 Random Numbers.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 022 — Clase 022 — Pandas: Series y DataFrame

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 3, §§ 3.1–3.2 Introducing Pandas Objects.*

> *Duración estimada: 90 min.*

Objetivo

Que el alumno entienda qué es una Series (ndarray + index) y un DataFrame (dict de Series alineadas por index), cómo se construyen desde 5 fuentes distintas, y por qué el index es el rasgo que distingue pandas de NumPy.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Crear Series y DataFrames desde dict, lista de tuplas, arrays NumPy, CSV y desde otro DataFrame.
2. Inspeccionar un DataFrame con head, tail, info, describe, dtypes, shape.
3. Acceder a columnas como atributo (df.col) y como key (df['col']) — y saber cuándo cada uno falla.
4. Modificar el index con set_index, reset_index, rename.
5. Convertir Series ↔ DataFrame ↔ ndarray cuando sea necesario.

Temas

#	Tema	Por qué importa
1	Series = ndarray + index	Index permite alineación automática.
2	DataFrame = dict de Series alineadas	Por eso df['col'] devuelve Series.
3	Construcción desde 5 fuentes	dict, lista de dicts, arrays, CSV, Series.
4	.loc vs .iloc vs []	Tres formas de acceso.
5	Index labels vs posición	El bug clásico cuando el index no es 0..N.
6	info y describe como first-look	Lo primero que mira un DS.

Versión profundizada — 2026

El tema moderno que vivía como complemento dentro de esta clase ahora tiene clase propia dedicada:

- Clase 032a — Polars: DataFrames modernos

Definiciones y características

Series
: Estructura 1D = ndarray + index labelled. s[label] accede por etiqueta, s.iloc[N] por posición. Característica clave: el index permite alineación automática entre Series.

DataFrame
: Estructura 2D = dict de Series que comparten el mismo index. Cada columna puede tener su propio dtype (a diferencia de un ndarray). df['col'] devuelve Series.

Index
: Estructura que etiqueta cada fila (o columna). Puede ser entero default (RangeIndex), strings, fechas (DatetimeIndex), o jerárquico (MultiIndex). Inmutable — para cambiarlo, set_index/reset_index.

Alineación automática

: Cuando operas dos Series/DataFrames, pandas alinea por index, NO por posición. Posiciones sin match → NaN. Esto es lo que distingue pandas de NumPy.

dtype en pandas

: Cada columna tiene su dtype. Tipos clásicos: int64, float64, bool, object (strings o mixto). Modernos (NA-aware): Int64, Float64, boolean, string, category.

Dataset / recursos

Palmer Penguins (descargable con seaborn/palmerpenguins) — 344 filas × 7 columnas, públicas, sin issues de licencia. Reemplaza al iris dataset.

Ejercicios

1. Series desde dict. Crea Series con población de 5 ciudades. Accede por label y por posición.

2. DataFrame desde dict de listas. Construye DataFrame de 5 estudiantes (nombre, edad, nota). Inspecciona con info() y describe().

3. Lee Palmer Penguins. pd.read_csv desde URL pública. Reporta shape, dtypes, % de NaN por columna.

4. Index labeled. Setea species como index. Compara df.loc['Adelie'] vs df.iloc[0].

5. Alineación automática. Crea 2 Series con index parcialmente solapado. Súmalas. Observa los NaN.

Homework verificable

Notebook que: (a) carga Palmer Penguins y reporta info(), describe(), missing por col; (b) muestra los 3 métodos de acceso a una columna (df.col, df['col'], df.loc[:, 'col']); (c) cambia el index a species, vuelve a default con reset_index; (d) demuestra alineación automática sumando dos Series.

Criterio de aceptación: Carga sin error, los 3 accesos producen la misma Series, alineación produce NaN donde corresponde.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
AttributeError al hacer df.column with spa	Acceso como atributo solo funciona con nom
KeyError: 'columna' aunque está en el Data	Espacios invisibles o capitalización disti
Sumar dos Series y aparecen NaN inesperado	Index no coincide en algunos labels. Fix:
pd.read_csv infiere mal los dtypes	Pandas adivina por las primeras filas; si
DataFrame is not callable: TypeError: 'Dat	Hiciste df() por error en vez de df (parén

Preguntas frecuentes

¿df['col'] o df.col?

df['col'] siempre — funciona con cualquier nombre, no choca con métodos. df.col falla con espacios, choca con .shape, .head, etc. Solo úsalo en exploración rápida.

¿Por qué read_csv me da Unnamed: 0?

El CSV se guardó con index (df.to_csv('x.csv')). Fix: al leer, pd.read_csv('x.csv', index_col=0). O al guardar, df.to_csv('x.csv', index=False).

¿Cuándo Series y cuándo DataFrame de 1 columna?

Series es más liviana y la mayoría de operaciones la prefieren. DataFrame de 1 col cuando vas a concatenar con otros DataFrames o necesitas mantener la columna nombrada. s.to_frame() convierte.

¿copy() cuándo?

Cuando vas a modificar un subset y NO quieres afectar el original. subset = df[cond].copy(). Si solo lees, no necesitas copy — y consumes el doble de memoria.

¿Qué pasa con NaN en una columna int?

Pandas la promueve a float64 automáticamente (porque NumPy int no soporta NaN). Para mantener int: usa dtype nullable Int64 (mayúscula) que sí permite pd.NA.

¿Tengo que aprender pandas Y polars?

Por ahora, no. Pandas primero — es la base del curso, lo vas a ver en clases, libros, Stack Overflow y en cualquier código que toques. Cuando lo domines y te topes con un dataset que no entra en RAM o un pipeline lento, ahí asomate a polars (Parte 5). La transición es directa porque los conceptos (DataFrame, columnas, group_by, joins) son los mismos; lo que cambia es la sintaxis y el motor.

Referencias

- VanderPlas, cap. 3 §§ 3.1, 3.2.
- pandas user guide — DataFrame
- Palmer Penguins
- Polars user guide

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 023 — Clase 023 — Pandas: indexación (loc, iloc, at, iat)

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 3 § 3.3 Data Indexing and Selection.*
>
> *Duración estimada: 75 min.*

Objetivo

Que el alumno domine los 4 indexers de pandas y elija el correcto según el caso. El bug "SettingWithCopyWarning" y el bug del slicing por label inclusivo nacen aquí — saber qué indexer usar evita ambos.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Usar .loc[row_label, col_label] para acceso por etiqueta (inclusivo en slicing).
2. Usar .iloc[row_pos, col_pos] para acceso por posición entera (exclusivo, como Python).
3. Usar .at / .iat para acceso a un único valor (más rápido que loc/iloc).
4. Evitar SettingWithCopyWarning usando .loc para asignar en una vista.

5. Filtrar filas con boolean mask dentro de .loc: df.loc[df['edad'] > 30, 'nombre'].

Temas

#	Tema	Por qué importa
1	[] directo: shortcut con quirks	Columnas → Series; filas → KeyError.
2	.loc: por label, slicing inclusivo	El indexer principal del 80% del tiempo.
3	.iloc: por posición, slicing exclusivo (co	Cuando no te importa el label.
4	.at / .iat: single value	Optimizado para 1 celda — útil en loops.
5	Mask + loc para filtros con asignación	df.loc[mask, 'col'] = valor.
6	SettingWithCopyWarning: qué es y cómo evit	Usar .loc para asignar.

Definiciones y características

.loc[fila, col]

: Acceso por etiqueta. Slicing es inclusivo en ambos extremos (df.loc['a':'c'] incluye 'c'). Acepta booleano: df.loc[df['x'] > 0, 'col'].

.iloc[fila, col]

: Acceso por posición entera. Slicing es exclusivo del extremo (estilo Python). df.iloc[0:5] da 5 filas (índices 0..4).

.at[fila, col] / .iat[fila, col]

: Versiones para acceder/asignar un único valor. ~10× más rápidos que .loc/.iloc cuando estás en loops. Mismo patrón label vs posición.

Indexer chaining (df[cond][col] = x)

: Encadenar dos [] operaciones de acceso. Crea ambigüedad: ¿es vista o copia? Origen del SettingWithCopyWarning. Evítalo siempre.

SettingWithCopyWarning

: Aviso de pandas: "estoy haciendo algo ambiguo, puede que tu asignación se pierda". Causado por chaining o asignación sobre subset no-explícito. Solución universal: .loc[cond, col] = x en una sola operación.

Dataset / recursos

Palmer Penguins desde URL (mismo de clase 022) o el sintético si no hay internet.

Ejercicios

1. Acceso simple. Carga penguins. Obtén la columna species con los 3 métodos: df.species, df['species'], df.loc[:, 'species'].

2. loc inclusivo vs iloc exclusivo. Con index 0..N por default, compara df.loc[0:5] vs df.iloc[0:5]. ¿Cuántas filas devuelve cada uno?

3. Filtro + columnas seleccionadas. Pingüinos Adelie machos con bill_length > 40: df.loc[(df.species=='Adelie') & (df.sex=='male') & (df.bill_length_mm > 40), ['species', 'island', 'bill_length_mm']].

4. Asignación segura. Crea una columna is_big que sea True si body_mass_g > 4500, usando .loc.

5. Provoca y arregla SettingWithCopyWarning. Slicea con df[df.x > 0] y modifica → ve warning. Hazlo con .loc → sin warning.

Homework verificable

Notebook que: (a) muestra los 3 métodos de acceso a columna; (b) compara loc vs iloc en slicing con tabla; (c) filtra Adelie machos con bill_length>40 mostrando 3 columnas; (d) reproduce y arregla SettingWithCopyWarning con explicación.

Criterio de aceptación: Los filtros producen el subset correcto; la versión con .loc no emite warning.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
SettingWithCopyWarning aparece y no sé por	Estás asignando sobre un resultado de slic
df.loc[0:5] da 6 filas no 5	loc es inclusive end. Para 5 filas: df.ilo
KeyError con .loc[5] cuando hice set_index	El index ya no es 0..N — es la columna id.
Asignación a vista no modifica el original	Pandas 3+ va a ser estricto: la vista no s
df.col1 = x no actualiza la columna	Como atributo, asignar no agrega columna n

Preguntas frecuentes

¿loc o iloc?

loc cuando el index es semántico (nombres, fechas, ids). iloc cuando solo importa la posición (top-K por orden, primeros 10, último). Mezclarlos en el mismo código causa confusión.

¿Por qué loc slicing es inclusivo?

Decisión histórica: para labels (strings, fechas), incluir el end es lo intuitivo ('enero':'marzo' debe incluir marzo). Para enteros default queda raro — usa iloc ahí.

¿at vale la pena vs loc para single value?

Solo en hot loops (>10k iteraciones). Para uso interactivo, loc es lo suficientemente rápido y más legible.

¿Cómo selecciono múltiples columnas?

Lista entre brackets: df[['a', 'b', 'c']] (devuelve DataFrame). Notar el [[]]. Un solo [] con string devuelve Series.

¿.loc[mask, cols] o .query() + [cols]?

Ambos válidos. .loc[mask, cols] para máscaras computadas; .query() para filtros declarativos largos. Misma velocidad para datasets <100k.

Referencias

- VanderPlas, cap. 3 § 3.3.
- pandas Indexing user guide
- SettingWithCopyWarning explained

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 024 — Clase 024 — Pandas: operaciones y alineación

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 3 § 3.4 Operating on Data in Pandas.*

> *Duración estimada: 60 min.*

Objetivo

Que el alumno entienda cómo pandas alinea automáticamente por index en operaciones entre Series/DataFrames, cómo manejar NaN resultantes, y use apply/map para transformaciones custom (con consciencia de cuándo es lento).

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Predecir el resultado de operar dos Series/DataFrames con indexes parcialmente distintos.
2. Usar fill_value en operaciones para no propagar NaN: s1.add(s2, fill_value=0).
3. Aplicar funciones con apply (lento, flexible), map (Series), applymap / df.map (elementwise).
4. Vectorizar transformaciones cuando se puede en vez de apply (10–100× más rápido).
5. Usar ufuncs NumPy sobre Series — pandas las soporta directamente y preserva el index.

Temas

#	Tema	Por qué importa
1	Alineación automática por index	Producto, suma, todo — pandas alinea, no a
2	fill_value para operaciones	Reemplaza el NaN antes de calcular.
3	apply axis=0 vs axis=1	Por columna vs por fila — costoso en filas
4	map para Series con dict	s.map({'A': 1, 'B': 2}).
5	df.map (era applymap) — elementwise	Cell-by-cell, lento.
6	Vectorización > apply	Si puedes hacerlo con ufunc, hazlo.

Definiciones y características

Alineación por index

: Operación matemática entre dos pandas (Series + Series, DF + Series, etc.) alinea por etiqueta de index. Labels que no estén en ambos → NaN.

fill_value en operaciones

: Parámetro que reemplaza NaN durante la operación: s1.add(s2, fill_value=0) trata índices ausentes como 0 en vez de propagar NaN.

apply

: Aplica función a cada fila (axis=1) o columna (axis=0) de un DataFrame, o cada elemento de una Series. Lento porque itera en Python. Usa solo si vectorización no aplica.

map (Series)

: Aplica función o dict a cada elemento. Útil para recodificación rápida: s.map({'A': 1, 'B': 2}). Más rápido que apply por su API más restringida.

df.map (antes applymap)

: Aplica función a cada celda del DataFrame (elementwise). Lento; usa solo cuando vectorización no aplica.

applymap está deprecated en pandas 2.1+.

Ufunc-aware

: Pandas Series respeta ufuncs NumPy: np.log(s), np.sqrt(s) funcionan y preservan el index. Ventaja sobre convertir a array y perder labels.

Dataset / recursos

Sintético + Palmer Penguins. Sin descarga adicional.

Ejercicios

1. Suma con alineación. Dos Series con index parcialmente solapado. Súmalas (default) y con fill_value=0.

2. apply por fila. Define una función que reciba una fila de penguins y devuelva BMI = body_mass / bill_length². Aplica con axis=1.

3. Mismo cálculo vectorizado. Implementa BMI con operaciones vectorizadas. Mide ambos con %timeit.

4. map con dict. Mapea species a códigos: {'Adelie': 0, 'Chinstrap': 1, 'Gentoo': 2}.

5. ufunc NumPy preserva index. Aplica np.log a una columna; verifica que el index sigue intacto.

Homework verificable

Notebook con penguins: (a) BMI por fila con apply vs vectorizado (tabla %timeit); (b) species → código numérico con map; (c) demo de alineación con fill_value; (d) np.log sobre body_mass preservando index.

Criterio de aceptación: Vectorizado >50× más rápido que apply. Mapping y alineación correctos.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
apply toma 10× más que vectorización equiv	Iteración Python por fila. Fix: piensa si
s1 + s2 produce NaN aunque los datos están	Index distinto (incluso por orden diferent
df.apply(func, axis=1) rompe con KeyError	Tu función accede row['col_x'] pero esa co
df.map(lambda x: ...) falla "object has no	Es método de DataFrame solo en pandas ≥2.1
np.log(df['x']) da error con NaN	Si NaN está presente, log da NaN. Si negat

Preguntas frecuentes

¿Cuándo apply y cuándo NO?

NO: si lo puedes hacer con df['a'] * df['b'] u operación pandas built-in (groupby, transform, etc.). SÍ: lógica compleja por fila que no se descompone.

¿apply(axis=1) con tipos mixtos da problemas?

Sí — pandas convierte cada row a Series con dtype común. Si tienes int y str, queda object y los operadores <, > rompen. Mejor extrae columnas y opera directo.

¿Cómo paralelizo apply?

pandarallel, swifter, modin — drop-in replacements. Pero antes de paralelizar, asegúrate que tu apply no es vectorizable (el speedup es mayor).

¿s.map(dict) o s.replace(dict)?

map: mapea uno-a-uno; valores no encontrados en dict → NaN. replace: reemplaza solo los que aparecen; el resto queda igual. Elige según comportamiento deseado en valores faltantes.

¿Por qué pandas a veces es más lento que NumPy?

Overhead del index, manejo de NaN, dtype-aware. Para operaciones puramente numéricas sobre datos limpios y rectangulares, NumPy puede ser 2-5× más rápido. Pandas vale por la API, no por velocidad raw.

Referencias

- VanderPlas, cap. 3 § 3.4.
- pandas — apply, map

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 025 — Clase 025 — Pandas: datos faltantes

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 3 § 3.5 Handling Missing Data.*

> *Duración estimada: 75 min.*

Objetivo

Que el alumno detecte, cuantifique y maneje datos faltantes con criterio. Eliminar es la opción fácil pero suele ser incorrecta: cuándo eliminar, cuándo imputar (media, mediana, forward-fill), y cuándo el faltante es señal que merece su propia columna.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Detectar NaN con isna(), notna() y cuantificar por columna/fila.
2. Eliminar filas/columnas con NaN usando dropna con how/thresh/subset.
3. Imputar con fillna: valor escalar, media/mediana, forward/backward fill, interpolación.
4. Distinguir NaN vs None vs pd.NA y por qué importan los dtypes nullable (Int64, boolean).
5. Decidir entre eliminar/imputar/dejar — y crear columna was_missing cuando el faltante es informativo.

Temas

#	Tema	Por qué importa
1	Tipos de missing en pandas: NaN, None, NaT	Cada uno tiene caso de uso.
2	Detección: isna, notna, isna().sum()	First-look obligatorio.
3	dropna: how='any'/'all', thresh, subset	Eliminar con precisión.
4	fillna: escalar, dict, ffill, bfill, inter	Imputar según contexto.
5	Dtypes nullable: Int64, Float64, boolean	El nuevo missing nativo.
6	was_missing como feature	Cuando el missing es señal.

Definiciones y características

NaN (Not a Number)

: Float especial IEEE 754 que representa missing en columnas numéricas. Característica: NO es igual a sí mismo (np.nan == np.nan → False). Usa pd.isna() para detectarlo.

None vs NaN

: None es objeto Python (en columnas object). NaN es float (en columnas numéricas). Pandas trata ambos como missing pero internamente son distintos.

pd.NA

: Sentinel "missing universal" (pandas 1.0+) compatible con dtypes nullable (Int64, boolean, string). Comportamiento más consistente que NaN en operaciones.

MCAR / MAR / MNAR

: MCAR (Missing Completely At Random): missing es aleatorio. MAR (Missing At Random): missing depende de variables observadas. MNAR (Missing Not At Random): depende del propio valor missing (peor caso).

Imputación

: Reemplazar missing con un valor estimado. Estrategias: media/mediana/moda global, por grupo (groupby.transform), forward-fill (series temporales), KNN, regresión, MICE.

was_missing flag

: Columna booleana adicional que registra qué filas tenían missing antes de imputar. Permite al modelo usar el hecho de que faltaba como feature (a veces es informativo).

Dataset / recursos

Palmer Penguins (tiene NaN reales en sex y mediciones). Sin descarga adicional si ya está en clases anteriores.

Ejercicios

1. Cuantifica. Carga penguins, reporta % de NaN por columna y por fila.

2. Eliminar filas con cualquier NaN. df.dropna(how='any'). Compara shape antes/después.

3. Eliminar solo filas con NaN en sex. df.dropna(subset=['sex']). Más selectivo.

4. Imputar. Rellena bill_length_mm con la mediana por especie (groupby + transform). Justifica por qué la mediana es mejor que la media aquí.

5. Forward fill en series temporales. Crea una Series con NaN intercalados. Aplica ffill, bfill, interpolate. Compara.

Homework verificable

Notebook con penguins: (a) reporte completo de missing (% por col, % por fila, filas más incompletas); (b) 3 estrategias: drop all, drop subset, imputar por grupo; (c) columna bill_was_missing y demuestra que el flag puede mejorar un modelo simple; (d) demo de dtypes nullable Int64.

Criterio de aceptación: Imputación por grupo no introduce sesgo; el flag was_missing añade información.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
if df['col'] == np.nan no funciona	NaN no es igual a nada (incluso a sí mismo

df.dropna() borra todo	Sin parámetros, borra fila con CUALQUIER N
Imputar con media global introduce sesgo	Si los grupos tienen medias muy distintas,
Una columna int se volvió float tras leer	Tiene NaN → promoción automática (NumPy in
fillna(0) mata el flag de "missing"	Pierdes la información de que faltaba. Fix

Preguntas frecuentes

¿Eliminar, imputar o flag?

Depende: <1% missing aleatorio → drop. 5-30% MAR → imputar (mediana por grupo). >50% → eliminar columna o tratar como categoría 'missing'. Si missing es informativo → flag + imputar.

¿Mediana o media para imputar?

Mediana es más robusta a outliers (recomendada por default). Media si la distribución es ~normal y sin outliers. Para categóricas: moda.

¿ffill siempre bueno para series temporales?

Bueno cuando los valores cambian poco entre observaciones (precios, temperaturas). Malo si los gaps son largos o el dato es volátil. Considera interpolate(method='time') para mejor resultado.

¿Cómo sé si la imputación afecta mi modelo?

Compara métricas: (a) baseline drop, (b) imputación X, (c) imputación + flag. Si el flag mejora el modelo, el missing era informativo (caso MNAR).

¿KNN/MICE para imputación en pandas?

No nativo. Usa sklearn.impute.KNNImputer o IterativeImputer (= MICE). Más caro pero mejor que mediana en datasets con correlaciones fuertes.

Referencias

- VanderPlas, cap. 3 § 3.5.
- pandas — Missing data user guide
- pandas nullable Integer dtypes

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 026 — Clase 026 — Pandas: MultiIndex

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 3 § 3.6 Hierarchical Indexing.*

> *Duración estimada: 75 min.*

Objetivo

Que el alumno use índices jerárquicos (MultiIndex) cuando hay estructura natural en los datos (país × ciudad,

año × mes, sector × empresa). Saber cuándo aporta vs cuándo complica — el 80% del tiempo en data science aplanado es mejor.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Crear MultiIndex desde tuplas, arrays, producto cartesiano (from_product).
2. Indexar con .loc[(nivel1, nivel2)] y .loc[:, ('grupo', 'col')].
3. Aplanar y reconstruir con unstack(), stack(), reset_index().
4. Decidir cuándo MultiIndex aporta (groupby con múltiples claves devuelve uno automáticamente) y cuándo es más legible aplanar.
5. Renombrar niveles con rename(level=...) y reordenarlos con swaplevel.

Temas

#	Tema	Por qué importa
1	MultiIndex: motivación	Datos con jerarquía natural.
2	Construcción: tuples, arrays, from_product	3 formas comunes.
3	Indexación: tuple selector	.loc[('A', 2024)].
4	stack / unstack — pivot rápido	Mover niveles entre filas y columnas.
5	groupby + multiindex resultado	groupby con 2+ claves devuelve MultiIndex.
6	Cuándo aplanar	Para CSV de salida, plot, scikit-learn.

Definiciones y características

MultiIndex

: Índice jerárquico con N niveles. Cada fila identificada por tupla de N labels (('España', 2024)). Útil cuando los datos tienen estructura natural (país→ciudad, año→mes).

Nivel (level)

: Cada "capa" del MultiIndex. Se referencia por nombre (level='año') o posición (level=0). Útil en operaciones como unstack(level=...).

stack / unstack

: Mueven niveles entre filas y columnas. unstack sube un nivel del index a columnas (long→wide). stack baja un nivel de columnas al index (wide→long). Reversibles.

xs (cross-section)

: Slice por un valor en un nivel: df.xs(2024, level='año'). Más limpio que indexar con tuplas parciales.

Aplanar (flatten)

: Convertir MultiIndex a Index plano: df.reset_index() (vuelve a default 0..N) o df.index = ['_'.join(map(str, t)) for t in df.index] (concatena niveles).

Dataset / recursos

Sintético: ventas por país/año.

Ejercicios

1. Construye desde tuplas. Crea DataFrame con index [(España, 2023), (España, 2024), (Chile, 2023), (Chile, 2024)] y 2 cols ventas/clientes.

2. from_product. Mismo con pd.MultiIndex.from_product([paises, años]).

3. Acceso jerárquico. df.loc['España'], df.loc[('España', 2024)]. Compara con df.xs(2024, level=1) para slice por nivel.

4. unstack y stack. Convierte tu MultiIndex en wide (años como columnas) y de vuelta.

5. groupby produce MultiIndex. Carga penguins, agrupa por (species, sex) y agrega mean(). Aplana con reset_index().

Homework verificable

Notebook con ventas trimestre×región sintéticas (4 trimestres × 3 regiones × 2 años): (a) construir con from_product; (b) acceso a un trimestre específico; (c) total por región (unstack); (d) groupby penguins por (species, sex) → MultiIndex → aplanar.

Criterio de aceptación: MultiIndex con shape correcto; unstack/stack reversibles.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
KeyError al acceder df.loc['España', 2024]	loc con MultiIndex requiere tupla: df.loc[
unstack() lanza ValueError: Index contains	Tienes filas duplicadas en (index, columns
groupby([a, b]).sum() devuelve cosa extrañ	Es correcto: groupby con N keys devuelve M
Plot ignora niveles del MultiIndex	matplotlib/seaborn esperan columnas planas
sort_index() ordena raro con MultiIndex	Default ordena por todos los niveles. Para

Preguntas frecuentes

¿Cuándo MultiIndex aporta vs cuándo complica?

Aporta en análisis interactivo con slicing por nivel frecuente. Complica para export a CSV, plot, sklearn — aplana ahí.

¿set_index([a, b]) vs groupby([a, b])?

set_index solo mueve cols al index (sin agregar). groupby colapsa filas por las cols (con sum/mean/agg). Diferentes operaciones.

¿Cómo evito MultiIndex en groupby?

groupby([a, b], as_index=False) devuelve DataFrame plano directamente. O .reset_index() después.

¿stack(future_stack=True) qué significa?

Es el comportamiento del nuevo stack (default en pandas 3+). Maneja NaN distinto al legacy. Mejor pasarlo siempre explícito para suprimir warnings.

¿Performance MultiIndex vs Index plano?

MultiIndex tiene overhead. Para datasets grandes (>1M filas) con acceso intenso, aplana al final del pipeline.

Referencias

- VanderPlas, cap. 3 § 3.6.
- pandas MultiIndex user guide

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 027 — Clase 027 — Pandas: concat, merge, join

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 3 §§ 3.7–3.8 Combining Datasets: Concat/Merge.*
>
> *Duración estimada: 90 min.*

Objetivo

Que el alumno junte datasets correctamente: concat (apilado simple), merge (SQL-style joins) y join (atajo por index). El error más común es usar el join equivocado y obtener duplicados o filas perdidas — saber qué tipo (inner/left/right/outer) evita semanas de bugs.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Apilar DataFrames con pd.concat por filas (axis=0) o columnas (axis=1).
2. Hacer joins SQL-style con pd.merge: inner, left, right, outer, cross.
3. Diagnosticar duplicados generados por merge con validate='one_to_one' | 'many_to_one' | ….
4. Joinear por index con df1.join(df2) (atajo para merge por index).
5. Usar indicator=True para saber qué filas vienen de cada lado del merge.

Temas

#	Tema	Por qué importa
1	concat axis=0 (filas) vs axis=1 (columnas)	Apilado simple con alineación de index.
2	merge how='inner'/'left'/'right'/'outer'	Los 4 tipos de join SQL.
3	on vs left_on/right_on	Cuando los nombres de columna difieren.
4	validate para evitar duplicación	1:1, 1:m, m:1, m:m.
5	indicator=True para auditar	Columna _merge con left_only/right_only/bo
6	df.join por index	Atajo idiomático.

Definiciones y características

concat

: Apila DataFrames por filas (axis=0, default) o columnas (axis=1). Alinea por el otro eje. No requiere key; es apilamiento puro.

merge (SQL-style join)

: Combina dos DataFrames por una key común. how='inner'/'left'/'right'/'outer'/'cross' controla qué filas se conservan.

INNER JOIN

: Solo filas presentes en AMBOS lados de la key. Si la key no matchea, se descarta. Default de merge.

LEFT JOIN

: Todas las filas del lado izquierdo (df1). Si no hay match en el derecho, columnas derechas quedan NaN. Útil para enriquecer datos sin perder ninguno.

OUTER JOIN

: Todas las filas de ambos lados; NaN donde no hay match. Vista "unión". Útil para auditoría.

validate

: Parámetro de merge que valida la cardinalidad esperada: 'one_to_one', 'one_to_many', 'many_to_one', 'many_to_many'. Si los datos no la cumplen, lanza error → evita duplicados ocultos.

indicator=True

: Añade columna _merge con 'left_only'/'right_only'/'both'. Útil para auditar qué tipo de match tuvo cada fila.

Dataset / recursos

Sintético: tabla de clientes + tabla de órdenes (relación 1:N).

Ejercicios

1. Concat por filas. 3 DataFrames mensuales con mismas columnas → uno anual. ignore_index=True.

2. Inner join. Clientes + órdenes por cliente_id. Verifica que solo aparecen clientes con al menos 1 orden.

3. Left join. Clientes + órdenes, conservando clientes sin órdenes (NaN en cols de orden).

4. Detectar duplicados. Provoca un merge muchos-a-muchos no intencional. Usa validate='one_to_many' para que falle si hay duplicación oculta.

5. indicator=True. Auditar cuántas filas son left_only / right_only / both.

Homework verificable

Notebook con clientes (10) + órdenes (25): (a) 4 tipos de join con _merge indicator; (b) tabla con conteo de cada tipo; (c) detección de relación con validate; (d) join por index con df.join.

Criterio de aceptación: Counts de cada join coherentes (inner ≤ left ≤ outer). validate lanza excepción si la relación esperada falla.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Tras merge tengo más filas que el DataFram	Relación uno-a-muchos no esperada. Fix: va
merge produce KeyError en la key	Tipos distintos: int vs str aunque el valo
Columnas se renombran con _x/_y tras merge	Ambos DataFrames tenían cols con el mismo
pd.concat([df1, df2]) da columnas extra co	Los dos tenían columnas distintas (pandas
concat ignora mi ignore_index=True y queda	Si tus DFs tienen index distintos, sin ign

Preguntas frecuentes

¿merge o join?

merge es la API rica (por columnas, control total). df1.join(df2) es atajo cuando ambos tienen index alineado a la key. Mismo motor por dentro.

¿Cuándo concat vs merge?

concat: apilas datos con la misma estructura (mes 1, mes 2, mes 3 → año). merge: combinas datasets

diferentes que comparten una key (clientes + órdenes).

¿validate siempre?

Sí — cuesta nada y atrapa el bug "silenciosamente generé el doble de filas". Recomendado en todo merge de producción.

¿Cómo merge por múltiples columnas?

merge(df1, df2, on=['a', 'b']) o left_on=['a','b'], right_on=['x','y']. La key compuesta es lista de strings.

¿Merge es lento con datasets grandes?

Con N=1M ya empieza a notarse. Acelera: setea index a la key antes (set_index('key').join(...)) o usa DuckDB (SELECT ... JOIN) — frecuentemente más rápido.

Referencias

- VanderPlas, cap. 3 §§ 3.7-3.8.
- pandas Merge user guide

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 028 — Clase 028 — Pandas: groupby (split-apply-combine)

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 3 § 3.9 Aggregation and Grouping.*

> *Duración estimada: 90 min.*

Objetivo

Que el alumno aplique el patrón split-apply-combine que es el patrón fundamental de análisis tabular: dividir por grupo, aplicar función, recombinar. Saber elegir entre agg, transform, filter y apply — cada uno tiene su rol.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Agrupar por una o más columnas con groupby y aplicar agregaciones (sum, mean, count).
2. Usar agg con dict para distintas funciones por columna: agg({'a': 'sum', 'b': 'mean'}).
3. transform para preservar la shape original (broadcasting del estadístico de grupo).
4. filter para filtrar grupos enteros según condición.
5. Diferenciar los 4 métodos del groupby y elegir el correcto.

Temas

#	Tema	Por qué importa
1	Split-apply-combine: el patrón	El más común en análisis tabular.
2	agg (= aggregate)	Reduce a una fila por grupo.

3	transform	Misma shape — útil para imputar/normalizar
4	filter	Conserva grupos completos según condición.
5	apply: el más flexible, el más lento	Cuando los 3 anteriores no alcanzan.
6	Múltiples columnas de agrupación	groupby(['a','b']) → MultiIndex.

Definiciones y características

Split-apply-combine

: Patrón: (1) split divide datos por valor de columna(s) → grupos; (2) apply ejecuta función en cada grupo; (3) combine junta resultados. El más usado en análisis tabular.

agg (= aggregate)

: Reduce cada grupo a una fila (sum, mean, count, std). Acepta función nombrada, lista de funciones, o dict por columna: agg({'a': 'sum', 'b': ['min','max']}).

transform

: Aplica función por grupo PERO mantiene la shape original (broadcastea resultado a cada fila). Ideal para z-score por grupo, imputación por grupo, ratios.

filter (groupby)

: Filtra grupos completos (no filas) según una condición. g.filter(lambda x: len(x) > 100) mantiene solo grupos con >100 filas.

apply (groupby)

: El más flexible y el más lento. Cualquier función custom por grupo (puede devolver Series, DataFrame, escalar). Úsalo solo cuando agg/transform/filter no alcanzan.

Named aggregation

: Sintaxis pandas 0.25+: g.agg(total=('monto', 'sum'), n=('id', 'count')). Más legible que el dict tradicional, permite renombrar en el mismo paso.

Dataset / recursos

Palmer Penguins (groupby por species y/o sex).

Ejercicios

1. Agg básico. Penguins agrupado por species: media de cada feature numérica.

2. Agg con dict. Por species: mean de bill_length, max de body_mass, count de filas.

3. Transform: z-score por grupo. Crea columna mass_z = z-score de body_mass dentro de su species.

4. Filter: solo grupos grandes. Conserva solo species con >100 individuos.

5. Apply custom. Por species, devuelve el pingüino con mayor body_mass (un DataFrame por grupo).

Homework verificable

Notebook con penguins: (a) agg múltiple por (species, sex); (b) transform z-score por species; (c) filter species con n>50; (d) apply que devuelva el top-3 más pesado por species; (e) tabla groupby.size() por sex × island.

Criterio de aceptación: z-score por grupo tiene media ≈ 0 y std ≈ 1 dentro de cada species.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
g.apply(...) lanza FutureWarning sobre inc	Pandas 2.2+ cambia comportamiento. Fix: g.
g.mean() solo muestra cols numéricas	Comportamiento intencional (pandas 2+). Fi
g['col'].transform(...) da error "function	Tu función devolvió shape distinta a la en
Resultado de g.agg(...) tiene MultiIndex e	Lista de funciones por columna → MultiInde
groupby(col).size() vs count() dan resulta	size: número de filas por grupo (incluye N

Preguntas frecuentes

¿agg, transform, filter o apply?

agg: reduces a 1 fila por grupo (resumen). transform: mantienes shape original (z-score). filter: excluyes grupos enteros. apply: lo demás, asumiendo overhead.

¿Cómo agrego columnas usando agg + named?

g.agg(total=('monto', 'sum'), avg=('monto', 'mean'), n=('id', 'count')).reset_index(). Tres columnas nombradas en una sola operación.

¿groupby([a, b]).agg(...).reset_index() o as_index=False?

Equivalentes. as_index=False evita el reset_index() posterior. Para encadenar con merge/concat, as_index=False es más limpio.

¿Cómo agrupo por una expresión derivada?

Pasa Series directa: df.groupby(df['fecha'].dt.year). O crea columna temporal: df.groupby(df['fecha'].dt.year.rename('año')).

¿groupby es lento con miles de grupos?

Con N=1M filas y K=1000 grupos, debería ser <1s. Si es más lento, posibles causas: agg con función custom Python (no built-in), keys con dtype object (string), o falta de sort. Usa sort=False si no necesitas orden.

Referencias

- VanderPlas, cap. 3 § 3.9.
- pandas groupby user guide
- Wickham, "The split-apply-combine strategy for data analysis" (J Stat Software, 2011).

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 029 — Clase 029 — Pandas: pivot tables y crosstab

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 3 § 3.10 Pivot Tables.*

> *Duración estimada: 60 min.*

Objetivo

Que el alumno construya tablas pivot (estilo Excel) con pivot_table y tablas de contingencia con crosstab. Son atajos sobre groupby pensados para resumen×visualización rápida.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Usar pivot_table con index, columns, values, aggfunc.
2. Añadir totales con margins=True.
3. Construir tablas de contingencia con pd.crosstab y normalizar (normalize='all'/'index'/'columns').
4. Diferenciar pivot (sin agregar) vs pivot_table (con aggfunc, agrega duplicados).
5. Visualizar una pivot como heatmap básico para confirmar patrones.

Temas

#	Tema	Por qué importa
1	pivot vs pivot_table	pivot no acepta duplicados; pivot_table sí
2	Parámetros: index, columns, values, aggfun	Análogos a Excel.
3	margins=True: totales	Útil para verificar.
4	crosstab: tabla de contingencia	Counts entre dos categóricas.
5	normalize en crosstab	Proporciones por fila/col/total.
6	Pivot → heatmap	Detectar patrones visualmente.

Definiciones y características

pivot_table

: Resumen tabular estilo Excel: defines index, columns, values y aggfunc. Acepta duplicados (agrega). El atajo más usado para reportes.

pivot (sin _table)

: Variante que NO agrega — falla si hay duplicados en (index, columns). Más estricta; úsala solo cuando garantizas unicidad.

crosstab

: Tabla de contingencia entre 2 categóricas: counts cruzados. Con normalize='index'/'columns'/'all' muestra proporciones.

margins=True

: Añade fila/columna "Total" al pivot. Útil para verificar manualmente y para reportes ejecutivos.

Heatmap de pivot

: Renderizar el pivot como matriz coloreada (plt.imshow o seaborn.heatmap) — patrones visuales saltan a la vista.

Dataset / recursos

Palmer Penguins. Sin descarga adicional.

Ejercicios

1. Pivot básico. Penguins: índice species, columnas sex, valores body_mass mean.

2. Pivot con totales. Mismo con margins=True.

3. Crosstab counts. Counts species × island.

4. Crosstab normalizado. Mismo con normalize='index' (% por fila).

5. Pivot → heatmap. Toma un pivot table y plotéala con matplotlib imshow.

Homework verificable

Notebook con penguins: (a) pivot_table (species × island, mean body_mass); (b) crosstab species × island, count y normalizado; (c) verificación de totales con margins; (d) heatmap simple del pivot.

Criterio de aceptación: Pivot con shape correcto; sum de normalize='index' = 1.0 por fila.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
pivot() lanza ValueError: Index contains d	Hay duplicados en (index, columns). Fix: u
pivot_table da NaN donde no hay datos	Combinaciones (index × columns) sin filas.
crosstab cuenta cosas raras con muchos NaN	Crosstab cuenta filas no-NaN por default.
Pivot con cols numéricas float queda feo	Sin aggfunc explícito, pandas usa mean. Si
Plot del pivot rompe por MultiIndex	Pivot con múltiples niveles de columnas →

Preguntas frecuentes

¿pivot_table o groupby + unstack?

Equivalentes en resultado. pivot_table es más declarativo, mejor para reportes. groupby + unstack más componible, mejor en pipelines.

¿crosstab o pivot_table con aggfunc='count'?

Equivalentes para counts. crosstab tiene API más simple para 2 categóricas. pivot_table más flexible (varias values, varias funcs).

¿Cómo ordeno el pivot?

Por valores: pivot.sort_values('col_x', ascending=False). Por suma: pivot.loc[pivot.sum(axis=1).sort_values(ascending=False).index].

¿Reportes Excel-like exportables?

pivot.to_excel('reporte.xlsx') directo. O to_csv para CSV. Para formato fino (colores, formulas), usa openpyxl o xlsxwriter.

¿Cuándo no usar pivot?

Cuando los datos ya están en formato wide y solo necesitas plot/agregaciones — usar groupby directo. Pivot es para transformar long → wide.

Referencias

- VanderPlas, cap. 3 § 3.10.
- pandas Pivot guide

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 030 — Clase 030 — Pandas: operaciones vectorizadas sobre strings

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 3 § 3.11 Vectorized String Operations.*

> *Duración estimada: 60 min.*

Objetivo

Que el alumno limpie y transforme columnas de texto sin caer en apply(lambda x: ...), usando el accessor .str de pandas — vectorizado, NaN-aware, con métodos análogos a los de Python (lower, strip, replace, split, contains, regex).

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Usar .str para aplicar operaciones de string vectorizadamente a una Series.
2. Manejar NaN automáticamente (los métodos .str propagan NaN sin error).
3. Aplicar regex con .str.contains(patron), .str.extract(...), .str.replace(...).
4. Dividir y unir con .str.split(sep, expand=True) que produce un DataFrame.
5. Trabajar con categorical cuando el cardinalidad es baja (memoria y speedup).

Temas

#	Tema	Por qué importa
1	Accessor .str	Métodos vectorizados que respetan NaN.
2	Casos típicos: lower, strip, replace, cont	El 80% del trabajo.
3	Regex con .str.extract y grupos nombrados	Extracción estructurada.
4	.str.split(expand=True) → DataFrame	Desnormalizar columnas combinadas.
5	dtype='string' (nullable) vs object	El moderno y NA-aware.
6	Categorical para baja cardinalidad	Menos memoria, groupby más rápido.

Complemento previo: Regex con el módulo re

Antes de meternos con .str.extract / .str.contains, conviene tener una intro mínima a expresiones regulares. Pandas usa regex por debajo en casi todos los métodos de texto, y en scraping (BeautifulSoup, parsing de HTML/logs) son prerrequisito invisible. Sin entender regex, los patrones se vuelven magia negra que se copypastea de Stack Overflow.

Metacaracteres esenciales

Símbolo	Significado	Ejemplo		
.	Cualquier carácter (excepto	a.c matchea abc, a c, a3c		
*	0 o más repeticiones del ar	ab* matchea a, ab, abbb		
+	1 o más repeticiones	ab+ matchea ab, abbb (no		
?	0 o 1 (opcional)	colou?r matchea color y co		
\d	Dígito [0-9]	\d\d\d matchea 123		
\w	Word char [a-zA-Z0-9_]	\w+ matchea hola_123		
\s	Whitespace (espacio, tab, r	\s+ matchea uno o más esp		
[]	Set de caracteres	[aeiou] matchea una vocal		

()	Grupo de captura	(\d+)-(\d+) captura ambos r		
^	Inicio de string	^Hola matchea solo si arra		
$	Fin de string	\.com$ matchea solo si terr		
`\	`	OR	`gato\	perro` matchea cualquiera
{n,m}	Entre n y m repeticiones	\d{2,4} matchea 2 a 4 dígit		

Funciones clave de re

Función	Para qué sirve
re.search(pat, s)	Busca el primer match en cualquier parte d
re.match(pat, s)	Igual que search pero solo desde el inicio
re.findall(pat, s)	Devuelve lista con todos los matches.
re.sub(pat, repl, s)	Reemplaza todos los matches por repl. Equi
re.compile(pat)	Pre-compila el patrón. Útil si lo vas a us
re.IGNORECASE (flag)	Match case-insensitive. Se pasa como flags

Mini-ejemplo — extraer dominio de email con grupos nombrados:

```
import re

email = "vladimir.acuna@gmail.com"
pat = re.compile(r"(?P<usuario>[\w.]+)@(?P<dominio>[\w.]+)")
m = pat.search(email)
print(m.group("usuario"))  # vladimir.acuna
print(m.group("dominio"))  # gmail.com
```

Raw strings (r"...")

Siempre se usan raw strings con regex. Sin la r, Python interpreta \d, \w, \s como secuencias de escape y muchas veces te las come o te tira DeprecationWarning. Con r"\d+" le decís a Python "esto es literal, no lo toques, pasáselo crudo a re". Regla: toda regex va en raw string, sin excepciones.

> *Herramienta recomendada para iterar patrones: regex101.com — testa en vivo, explica cada token y soporta flavor Python.*

Definiciones y características

Accessor .str

: Espacio de nombres en Series con métodos string vectorizados. Análogos a los de Python (.lower(), .split(), .replace()) pero aplicados elementwise y NaN-aware (propagan NaN sin error).

.str.extract(pattern)

: Aplica regex con grupos () y devuelve DataFrame con una columna por grupo. Soporta grupos nombrados ((?P<dominio>...)).

.str.split(sep, expand=True)

: Divide cada string y opcionalmente expande a DataFrame de columnas. Útil para denormalizar 'Apellido, Nombre' → 2 cols.

dtype 'string' (nullable)

: Versión moderna del dtype para texto. Diferencias con object: NA-aware (usa pd.NA), futuras optimizaciones. Recomendado en pandas 2+.

Categorical

: Dtype para columnas con cardinalidad baja (pocos valores únicos). Almacena cada valor como entero + diccionario. Ahorra ~10× memoria y acelera groupby/sort.

Dataset / recursos

Sintético: emails, nombres con espacios, fechas como string.

Ejercicios

1. Lower + strip. Lista de emails con mayúsculas y espacios. Normaliza con .str.lower().str.strip().

2. Extract dominio. De una columna de emails, extrae el dominio con regex (@(.+)$).

3. Split nombre completo. Columna 'Ana García' → nombre, apellido en columnas separadas.

4. Filtro por contains. Filas donde la columna descripcion contiene la palabra (case-insensitive) 'urgente'.

5. Categorical. Convierte una columna con 5 valores únicos en 100k filas a Categorical. Compara memoria.

Homework verificable

Notebook con CSV sintético de contactos (nombre, email, teléfono): (a) normalizar email (lower+strip); (b) extraer dominio; (c) separar nombre/apellido; (d) flag de email corporativo (no gmail/yahoo/hotmail); (e) convertir país a Categorical y reportar memoria.

Criterio de aceptación: Operaciones manejan NaN sin error. Categorical reduce memoria al menos 5×.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
'NoneType' has no attribute 'lower' al hac	Hay NaN/None en la Series. Fix: usa .str.l
Regex no captura nada con .str.extract	Falta () para definir grupo, o el pattern
.str.contains('foo') lanza error con NaN	Por default, na=NaN propaga. Fix: s.str.co
Convertí a Categorical y el sort sale alfa	Categorical por default es no-ordenado. Fi
.str.split(',') da listas, no columnas	Sin expand=True. Fix: s.str.split(',', exp
Regex con "\d+" no matchea o tira Deprecat	Olvidaste la r del raw string — Python int

Preguntas frecuentes

¿.str.lower() o .apply(str.lower)?

.str.lower() siempre — vectorizado, maneja NaN, mucho más rápido en N grande. apply es loop Python disfrazado.

¿Cuándo convertir a Categorical?

Cuando la cardinalidad es baja (~<5% de N filas) y vas a hacer groupby/sort. Para 100k filas de 5 países: enorme ganancia. Para 100k filas de 80k strings únicos: no ayuda.

¿'string' o object dtype?

'string' para código nuevo (NA-aware). object sigue siendo default por compat. Conviértelo explícito: df['col'] = df['col'].astype('string').

¿Regex case-insensitive?

s.str.contains('foo', case=False) o flags=re.IGNORECASE. También s.str.lower().str.contains('foo') (más explícito).

¿Cómo elimino acentos?

Pandas no trae nativo. Usa unidecode o s.str.normalize('NFKD').str.encode('ascii', 'ignore').str.decode('ascii').

¿Por qué \d a veces falla?

Porque te olvidaste de la r del raw string. "\d" en Python plano no es un escape válido y según la versión te lo come o te tira DeprecationWarning (y en Python 3.12+ va directo a SyntaxWarning). Con r"\d" el backslash se pasa literal a re, que es quien lo interpreta como "dígito". Regla mecánica: toda regex en raw string, sin pensarlo.

Referencias

- VanderPlas, cap. 3 § 3.11.
- pandas Text data user guide
- pandas Categorical
- Python re HOWTO

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 031 — Clase 031 — Pandas: series de tiempo, resampling, rolling

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 3 § 3.12 Working with Time Series.*
>
> *Duración estimada: 90 min.*

Objetivo

Que el alumno trabaje con datos temporales correctamente: parsear fechas, indexar por DatetimeIndex, hacer resampling (cambiar la frecuencia) y rolling (ventanas móviles para tendencias).

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Parsear strings de fecha con pd.to_datetime(..., format=..., errors=...).
2. Indexar por DatetimeIndex y slicear con strings de fecha (df.loc['2024-01':'2024-03']).
3. Resamplear a otra frecuencia: df.resample('M').sum(), 'W', 'D', 'H'.
4. Aplicar ventanas móviles con rolling(window).mean() para suavizar tendencias.
5. Manejar zonas horarias con tz_localize y tz_convert.

Temas

#	Tema	Por qué importa
1	pd.to_datetime con errors='coerce'	Parseo robusto.
2	DatetimeIndex y slicing por fecha	Sintaxis natural: '2024-01':'2024-03'.

3	Resampling: 'D', 'W', 'M', 'Q', 'Y', 'H'	Cambiar frecuencia + agregar.
4	Rolling windows	Suavizado, medias móviles.
5	shift y diff	Diferencias entre periodos, lag features.
6	Timezones: localize → convert	Cuando los datos tienen TZ.

Definiciones y características

Timestamp / DatetimeIndex

: Tipo nativo de pandas para fechas-horas. Vectorizado. Permite indexar con strings: df.loc['2024-01':'2024-03'].

pd.to_datetime

: Conversor robusto string→Timestamp. errors='coerce' convierte fallos a NaT (Not a Time) en vez de excepción.

Resampling

: Cambiar la frecuencia de una serie: diaria→mensual, horaria→semanal. Requiere agregación (sum, mean, last, ohlc). Códigos: 'D', 'W', 'ME' (month-end), 'h', 'min'.

Rolling window

: Ventana móvil: para cada punto, aplicar función a los últimos N (.rolling(N).mean()). Suaviza tendencias, calcula medias móviles.

shift / diff / pct_change

: shift(N): desplaza N posiciones (NaN al inicio). diff(N): s - s.shift(N) (cambio absoluto). pct_change(): cambio relativo.

tz_localize vs tz_convert

: localize asigna TZ a fecha naive (no convierte). convert convierte de una TZ a otra (mantiene el instante). Patrón: localize primero, convert después.

Dataset / recursos

Sintético: serie diaria de 2 años de ventas. Sin descarga.

Ejercicios

1. Parseo robusto. Lista de fechas con formatos mixtos ('2024-01-15', '15/02/2024', 'foo'). Parsea con errors='coerce'. Reporta NaT.

2. Slice por fecha. Con índice datetime, selecciona Q1 2024 con df.loc['2024-01':'2024-03'].

3. Resample diaria → mensual. Suma ventas por mes con df.resample('M').sum().

4. Rolling 7-day mean. Calcula media móvil de 7 días sobre ventas diarias. Plotea junto a la serie original.

5. shift para lag feature. Crea columna ventas_lag_1 con shift(1). Útil para features de ML.

Homework verificable

Notebook con serie sintética de 2 años: (a) parseo robusto; (b) slice por trimestre; (c) resample a mensual con sum y mean; (d) rolling 7/30 días con plot; (e) diff y pct_change para variación.

Criterio de aceptación: Plots muestran tendencia clara. Resample correcto (#meses esperado).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
pd.to_datetime falla con ValueError	Formato no estándar. Fix: format='%d/%m/%Y
df.loc['2024-13'] lanza KeyError	Mes inválido. Fix: verifica formato — pand
resample('M') warning sobre deprecation	Pandas 2.2+ prefiere 'ME' (month-end) o 'M
Resta de fechas da Timedelta en lugar de n	Es correcto — usa .dt.days, .dt.seconds pa
Operaciones con TZ mezcladas: Cannot compa	Una fecha tiene timezone, la otra no. Fix:

Preguntas frecuentes

¿Cuándo DatetimeIndex?

Cuando vas a hacer resample, slicing por fechas, o cualquier operación temporal. set_index('fecha') después de parsear.

¿resample o groupby(date.dt.month)?

resample es nativo y maneja gaps + zonas horarias mejor. groupby para agrupaciones no temporales ("por mes ignorando año").

¿Rolling cuánto Nuevoiniciar?

Default: NaN en los primeros N-1 (no hay datos suficientes para la ventana). Si quieres usar lo que haya: .rolling(N, min_periods=1).

¿Cómo manejo zonas horarias en producción?

Regla: guarda todo en UTC, convierte solo para mostrar. df['fecha'] = pd.to_datetime(...).dt.tz_localize('UTC') al ingerir, tz_convert('America/Santiago') al exhibir.

¿pd.date_range o pd.timestamp_range?

date_range — timestamp_range no existe. pd.date_range('2024-01-01', '2024-12-31', freq='D') para 365 fechas diarias.

Referencias

- VanderPlas, cap. 3 § 3.12.
- pandas Time Series user guide

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 032 — Clase 032 — Pandas: eval y query

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 3 § 3.13 High-Performance Pandas: eval and query.*

> *Duración estimada: 45 min.*

Objetivo

Que el alumno conozca df.eval y df.query — herramientas para expresar operaciones y filtros con sintaxis tipo SQL en strings. Útiles para legibilidad en cadenas largas y, en datasets muy grandes, también más rápidos (usan numexpr).

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Filtrar con df.query("col > 10 and other == 'X'").
2. Calcular columnas nuevas con df.eval('z = x + y') o df.eval('x * 2').
3. Referenciar variables locales en query/eval con prefijo @: df.query('x > @threshold').
4. Decidir cuándo usar query (legibilidad en cadenas largas) vs filtro tradicional (mejor autocompletado IDE).
5. Saber que el speedup real solo aparece con datasets >10k filas y expresiones complejas.

Temas

#	Tema	Por qué importa
1	df.query — sintaxis tipo SQL	Una sola string en vez de máscara compuest
2	df.eval — expresiones aritméticas	Calcula columnas sin temporales.
3	Variables locales con @	Pasar valores del scope.
4	numexpr para speedup	Solo en datasets grandes.
5	Trade-off: legibilidad vs introspección ID	Query strings no tienen autocompletado.

Definiciones y características

df.query()

: Filtro como string tipo SQL: df.query('precio > 100 and categoria == "A"'). Más legible que máscara booleana compuesta cuando hay >2 condiciones.

df.eval()

: Evalúa expresiones aritméticas: df.eval('total = precio * cantidad'). Con inplace=True añade la columna al DF.

Variables locales (@var)

: Dentro de query/eval, prefijo @ referencia variables del scope Python: df.query('x > @threshold').

numexpr

: Motor de cómputo que vectoriza expresiones en C/SIMD. Usado por eval/query bajo el capó cuando los datasets son grandes. Acelera operaciones aritméticas complejas.

Trade-off legibilidad vs IDE

: Query strings: más legibles para humanos. Filtros tradicionales: autocomplete del IDE, type checking. Elige según contexto.

Dataset / recursos

Sintético: DataFrame grande para benchmark. Sin descarga.

Ejercicios

1. Filter tradicional vs query. df[(df.a > 10) & (df.b < 5) & (df.c == 'x')] vs df.query('a > 10 and b < 5 and c == "x"'). Compara legibilidad.

2. Variable local. threshold = 100; filtra con df.query('precio > @threshold').

3. eval para nueva columna. df.eval('total = precio * cantidad', inplace=True).

4. Benchmark. Genera df 1M filas. Compara filter tradicional vs query con %timeit.

5. eval con inplace=False vs cálculo tradicional df['total'] = df['precio'] * df['cantidad'] — verifica resultados idénticos.

Homework verificable

Notebook con df 100k filas: (a) 3 filtros equivalentes (mask, query, query con @var); (b) eval para crear 2 columnas derivadas; (c) benchmark tradicional vs query en N=100k y N=1M; (d) reporte: cuándo conviene cada uno.

Criterio de aceptación: Resultados idénticos entre métodos. Speedup de query aparece en N≥100k.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
UndefinedVariableError: name 'X' is not de	Variable Python no prefijada con @. Fix: d
Strings dentro de query con comillas doble	Mezcla de quotes. Fix: usa quotes opuestas
df.eval('col_nueva = ...') no añade la col	Sin inplace=True. Fix: df.eval('col = xy',
query/eval más lento que máscara tradicion	Para datasets pequeños (<10k filas), el ov
Funciones custom no funcionan en query	Solo aritmética + operadores. Fix: df.quer

Preguntas frecuentes

¿query o máscara tradicional?

Máscara para filtros simples (1-2 condiciones) y autocomplete del IDE. query para filtros largos (4+ condiciones), parametrizables (@var), o cuando el lector lo encuentra más claro.

¿eval y query siempre son más rápidos?

No — solo en datasets grandes (>100k) con expresiones complejas. Para pequeños son iguales o ligeramente más lentos.

¿Qué operadores acepta query?

Aritméticos (+ - / *), comparación (==, <, >, <=, >=, !=), boolean (and, or, not o &, |, ~), in, not in. No funciones.

¿query reemplaza a SQL en pandas?

Para filtros, sí (mismo nivel expresivo). Para joins y aggregations, no — usa merge y groupby clásicos. Para datasets >1GB, considera DuckDB directo.

¿Hay eval peligroso (security)?

pd.eval no usa exec() Python — es parser propio limitado. No ejecuta arbitrary code. Pero: nunca pases strings de usuario sin sanitizar a query/eval.

Referencias

- VanderPlas, cap. 3 § 3.13.
- pandas eval/query docs

- numexpr project

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 033 — Clase 033 — Polars: DataFrames modernos

> *Parte: 0 — Prerrequisitos · Fuente: Polars User Guide + Vink (2020+) · Duración estimada: 75 min.*

Objetivo

Conocer Polars — la librería de DataFrames moderna (Rust + Arrow) que está reemplazando a pandas en proyectos donde performance importa. Aprender su API (similar a pandas pero con expresiones lazy y paralelismo automático) y entender cuándo conviene Polars sobre pandas (datasets > 1 GB, pipelines con muchas transformaciones, multi-core).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Instalar Polars (pip install polars) y leer datos con pl.read_csv, pl.read_parquet, pl.scan_csv (lazy).
- Aplicar la API de expresiones: df.select(pl.col('precio').sum()), pl.col('x').filter(pl.col('y') > 0).mean().
- Diferenciar eager (DataFrame) de lazy (LazyFrame) — y por qué lazy permite optimizaciones del query planner.
- Hacer groupby, join, pivot, unpivot con sintaxis Polars y comparar con pandas.
- Reconocer el speedup típico: 5-30× sobre pandas en operaciones comunes (single-machine, multi-core).

Temas

- Polars vs pandas vs DuckDB: el panorama 2026.
- Arrow como formato columnar in-memory.
- Eager (DataFrame) vs Lazy (LazyFrame).
- Expresiones encadenables: pl.col(...).operation().
- Query optimization automática: predicate pushdown, projection pushdown.
- Multi-threading automático.

Definiciones y características

- Polars: librería de DataFrames escrita en Rust, ABI estable en Python.
- pl.DataFrame: estructura eager — los datos están en RAM.
- pl.LazyFrame: descripción de pipeline; ejecución diferida hasta .collect().
- Expresión (pl.col, pl.lit): describe operación sobre columna; se compone y optimiza.
- pl.scan_csv / scan_parquet: leer lazy — solo se materializa lo que necesitás.
- Query optimization: el planner reordena filtros y projections para minimizar trabajo.
- streaming: para datasets mayores que RAM (collect(streaming=True)).

Dataset / recursos

- NYC Taxi (~100 MB Parquet) o cualquier CSV > 100 MB.

- Librerías: polars, pandas (para comparar), pyarrow.

Ejercicios

1. Eager básico: df = pl.read_csv('archivo.csv'); df.head(); df.describe(). Comparar con pandas.
2. Expresiones: df.filter(pl.col('precio') > 100).group_by('categoria').agg(pl.col('precio').mean().alias('precio_medio')).
3. Lazy + collect: lf = pl.scan_csv('big.csv').filter(...).group_by(...).agg(...); result = lf.collect(). Comparar tiempo vs eager.
4. Query plan: lf.explain() muestra el plan optimizado. Identificar predicate pushdown.
5. Pandas ↔ Polars: df.to_pandas() y pl.from_pandas(pd_df). Útil para mantener compatibilidad gradual.

Homework verificable

Sobre un dataset > 500 MB (NYC Taxi recomendado):

1. Pipeline en pandas y en Polars (lazy): filtrar viajes con tip_amount > 5, agrupar por hora del día, calcular media y desviación.
2. Medir wall time de cada uno.
3. Reportar speedup.

Criterio de aceptación: Polars debe ser ≥ 3× más rápido que pandas; los resultados numéricos deben coincidir (±0.001).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Mezclar APIs (df['col'].sum() estilo panda	Funciona pero no usa optimizaciones. Fix:
LazyFrame no devuelve datos	Es perezoso. Fix: .collect() al final del
Sin streaming=True en datasets enormes	OOM. Fix: lf.collect(streaming=True).
Columnas con tipos string mezclados → sche	Polars es estricto con tipos. Fix: pl.read
Esperar índices como en pandas	Polars NO tiene índice. La columna que qui

Preguntas frecuentes

¿Polars reemplaza a pandas?

Para data engineering / pipelines: cada vez más sí. Para ad-hoc analysis en Jupyter y compatibilidad con todo el ecosistema (sklearn, statsmodels, plotting): pandas sigue siendo el default. Ambos coexisten.

¿Polars o DuckDB?

Polars: API DataFrame Python-first. DuckDB: SQL embebido, mejor para joins masivos y queries SQL. Se complementan: muchos pipelines usan DuckDB para reads grandes + Polars para transformaciones.

¿pl.col vs df['col']?

pl.col es una expresión (reusable, optimizable). df['col'] materializa una Series — útil para inspección pero no para pipelines.

¿Polars en producción?

Sí. Empresas como Goldman Sachs, Citadel, Bank of America lo usan. Estable, mantenimiento activo, sponsoreado por la compañía Polars Cloud.

¿Polars soporta lazy streaming desde S3?

Sí: pl.scan_parquet('s3://bucket/*.parquet') con credentials configuradas. Lee solo lo que necesita.

Referencias

- Polars User Guide
- Polars Cookbook
- McKinney (2022 blog), Apache Arrow and the "10 Things I Hate About pandas".
- Comparativa: <https://h2oai.github.io/db-benchmark/> — Polars consistentemente top.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 034 — Clase 034 — Parquet, Arrow, PyArrow, DuckDB

> *Parte: 0 — Prerrequisitos · Fuente: docs Apache Arrow + DuckDB + Wes McKinney blogs. Duración estimada: 75 min.*

Objetivo

Conocer el stack columnar moderno que reemplaza al CSV para datos serios: Parquet (formato en disco), Arrow (formato in-memory), PyArrow (la implementación Python), y DuckDB (SQL embebido sobre Parquet/Arrow). Saber por qué el ecosistema entero (Polars, pandas 2.x, Spark, BigQuery, DataFusion) convergió a este stack.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Leer y escribir Parquet con pandas, polars y pyarrow.
- Aplicar column pruning (leer solo columnas necesarias) y predicate pushdown (leer solo filas necesarias).
- Manejar particionado por columna (year=2024/month=03/) para queries eficientes.
- Hacer queries SQL con DuckDB directamente sobre Parquet sin cargarlo a RAM.
- Reconocer ventajas de Arrow: zero-copy entre librerías (Polars ↔ pandas ↔ Spark).

Temas

- CSV: limitaciones (sin tipos, fila por fila, sin compresión).
- Parquet: columnar, comprimido (snappy/zstd), tipos preservados, metadata por chunk.
- Arrow: formato in-memory zero-copy.
- PyArrow: API Python para ambos.
- DuckDB: "SQLite para analytics", consulta Parquet directo.
- Particionado tipo Hive.

Definiciones y características

- Parquet: formato columnar en disco. Standard de facto para data engineering.
- Arrow: formato columnar in-memory standardizado. Zero-copy entre lenguajes.
- PyArrow: implementación Python de Arrow (pa.Table, pa.parquet.read_table).
- Column pruning: leer solo las columnas requeridas → ahorra I/O y RAM.

- Predicate pushdown: aplicar filtros en el read level → no carga lo que descartás.
- DuckDB: in-process SQL OLAP DB. pip install duckdb y ya.
- Hive partitioning: directorios clave=valor/; el lector deduce particiones.

Dataset / recursos

- NYC Taxi Parquet (Cloudfront público).
- Librerías: pyarrow, duckdb, polars, pandas.

Ejercicios

1. CSV → Parquet: leer un CSV grande con pandas, escribir Parquet con df.to_parquet('file.parquet', compression='zstd'). Comparar tamaño en disco (típicamente 3-10× menor).
2. Column pruning: leer SOLO 2 columnas de un Parquet de 50 columnas con pq.read_table('f.parquet', columns=['a', 'b']). Comparar tiempo vs leer todo.
3. DuckDB sobre Parquet: duckdb.sql("SELECT date, AVG(amount) FROM 'taxi/*.parquet' WHERE amount > 10 GROUP BY date").df(). Sin cargar nada explícitamente.
4. Particionado: escribir df.to_parquet('out/', partition_cols=['year', 'month']). Inspeccionar estructura de directorios.
5. Arrow zero-copy: arrow_table = polars_df.to_arrow(); pandas_df = arrow_table.to_pandas(). Verificar que es rápido (no copia memoria).

Homework verificable

Sobre NYC Taxi (varios meses Parquet, ~5 GB total):

1. Query con DuckDB: total tip_amount por mes para passenger_count >= 2. Sin descargar todos los archivos a memoria.
2. Reportar tiempo y RAM peak (psutil o resource).
3. Comparar contra pandas leyendo todo y filtrando.

Criterio de aceptación: DuckDB completa el query usando < 2 GB RAM mientras pandas explota (OOM o muy lento).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Parquet sin compresión → archivos enormes	Fix: compression='zstd' (default moderno)
read_parquet carga todo aunque solo necesi	Sin column pruning. Fix: columns=['a', 'b'
DuckDB no encuentra archivos	Path glob mal. Fix: 'data/.parquet', 'data
Particionado no detectado	Estructura no Hive. Fix: directorios key=v
Mixing PyArrow Table con pandas DataFrame	Confusión. Fix: decidir un formato; conver

Preguntas frecuentes

¿Parquet o CSV en 2026?

Parquet, salvo para interchange manual con herramientas no técnicas (Excel). 3-10× más chico, 10-100× más rápido de leer, tipos preservados.

¿DuckDB o Spark?

DuckDB para single-machine, datasets < TB. Spark para cluster, > TB. La frontera se mueve: DuckDB maneja hasta cientos de GB cómodamente.

¿Arrow es solo formato o también compute?

Ambos. Arrow Compute (pyarrow.compute) tiene operaciones vectorizadas. Polars usa el motor de Arrow + propio (rust).

¿Por qué zstd?

Compresión mejor que snappy a costo similar de CPU. Es el default en formats modernos (Polars write_parquet default).

¿pandas 2.x soporta Arrow nativo?

Sí — pd.read_csv(..., dtype_backend='pyarrow') usa Arrow types internamente. Más rápido, menos RAM.

Referencias

- Apache Arrow project
- Apache Parquet docs
- DuckDB docs
- McKinney (2017 blog), Apache Arrow and the "10 Things I Hate About pandas".

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 035 — Clase 035 — Matplotlib: anatomía figura/axes

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 4 § 4.1 Visualization with Matplotlib.*
>
> *Duración estimada: 60 min.*

Objetivo

Que el alumno entienda la jerarquía de objetos de matplotlib (Figure → Axes → Artist) y use la API orientada a objetos (fig, ax = plt.subplots()) en vez del interfaz pyplot estilo MATLAB. Esto es lo que separa gráficos publicables de notebooks de cualquier curso introductorio.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Explicar la jerarquía Figure → Axes → Artist y por qué la API OO es preferible.
2. Crear una figura con fig, ax = plt.subplots(figsize=(8, 4)) y configurar título, ejes, leyenda.
3. Guardar una figura a PNG/SVG/PDF con DPI controlado.
4. Cerrar figuras explícitamente para liberar memoria en notebooks que generan muchas.
5. Configurar defaults con plt.rcParams (font, line width, colors).

Temas

#	Tema	Por qué importa
1	Figure (canvas) → Axes (gráfico) → Artist	Modelo mental.

2	pyplot vs OO API	plt.plot (state-based) vs ax.plot (explíci
3	fig, ax = plt.subplots()	El patrón canónico.
4	fig.savefig y formatos	PNG raster vs SVG/PDF vector.
5	Liberar memoria: plt.close(fig)	Importante en loops.
6	plt.rcParams y stylesheets	Defaults globales.

Definiciones y características

Figure

: El canvas/lienzo completo (ventana, archivo PNG). Una Figure contiene N Axes.

Axes

: Un gráfico individual con sus ejes X/Y, ticks, labels, leyenda. No es plural de axis — es la palabra técnica de matplotlib para 'el rectángulo donde se dibuja'.

Artist

: Todo lo dibujado: lines (Line2D), puntos (PathCollection), texto, leyenda. Cada element es un Artist con propiedades modificables.

API OO vs pyplot

: OO: fig, ax = plt.subplots(); ax.plot(...) — explícito, escalable. pyplot: plt.plot(...) — estilo MATLAB, estado global, menos predecible. Usa OO siempre.

rcParams

: Dict global con defaults de matplotlib: plt.rcParams['figure.figsize'] = (10, 5). Modificar afecta todos los plots subsiguientes hasta reset.

Dataset / recursos

Sintético: serie temporal corta + scatter. Sin descarga.

Ejercicios

1. Hello world. Crea figura 8×4, plot de y = sin(x) para x [0, 2π]. Título, xlabel, ylabel.

2. Dos líneas en un axes. Misma figura: sin(x) y cos(x) con colores distintos y leyenda.

3. Guarda 3 formatos. Mismo plot a PNG (100 DPI), PNG (300 DPI), SVG. Compara tamaños.

4. Loop sin leak. Genera 20 plots en loop. Cierra cada uno con plt.close(fig). Verifica que len(plt.get_fignums()) queda en 0.

5. rcParams. Cambia font.size y lines.linewidth para tu sesión. Verifica el efecto.

Homework verificable

Notebook: (a) figura sin/cos con todos los elementos (título, labels, leyenda, grid); (b) guardar PNG@300dpi y SVG; (c) generar 50 plots en loop sin memory leak; (d) demo de rcParams modificados.

Criterio de aceptación: Plot publicable (labels, leyenda, tamaño razonable). Loop deja 0 figuras abiertas.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Mi notebook se llena de memoria al hacer m	Cada plt.subplots() deja Figure abierta. F

savefig('out.png') da PNG con fondo transp	Default es transparent. Fix: fig.savefig('
Labels cortados al guardar	Bounding box no incluye text fuera del axe
plt.plot(x, y) no muestra nada en notebook	Falta %matplotlib inline (default suele es
Texto en español sale mal	Default font no tiene tildes/ñ. Fix: plt.r

Preguntas frecuentes

¿fig, ax = plt.subplots() o fig = plt.figure(); ax = fig.add_subplot()?

subplots() es el shortcut más usado. Da figure + axes en una línea, devuelve grid si pasas (n, m). El segundo es más explícito y útil para layouts custom (GridSpec).

¿PNG, SVG o PDF?

PNG para web/presentaciones (raster, DPI fijo). SVG para documentos editables / publicación (vector, escala infinita). PDF para LaTeX (también vector).

¿Cuándo constrained_layout=True vs tight_layout()?

constrained_layout=True (al crear figure): más nuevo, más confiable, maneja colorbars y leyendas externas mejor. tight_layout() (después): legacy, falla con elementos no estándar.

¿Por qué mi plot se ve diferente entre script y notebook?

Backend distinto: notebook usa inline, script usa Qt5Agg/Tk. DPI y tamaño cambian. Para consistencia: plt.rcParams['figure.dpi'] = 100 explícito.

¿Está bien usar plt.plot() directo?

Para 1 plot rápido en notebook, OK. Para cualquier cosa más compleja (subplots, savefig, reutilizable), siempre API OO.

Referencias

- VanderPlas, cap. 4 § 4.1.
- matplotlib quick start
- Anatomy of a figure (matplotlib gallery)

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 036 — Clase 036 — Matplotlib: line, scatter, bar, histogram, boxplot

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 4 §§ 4.2–4.5 Simple Line/Scatter/Bar/Histogram Plots.*

> *Duración estimada: 75 min.*

Objetivo

Que el alumno conozca los 5 plots básicos que cubren el 80% del trabajo de EDA, y sepa cuándo cada uno: line (tendencia temporal), scatter (relación dos variables), bar (categóricas), histogram (distribución), boxplot (5 estadísticos + outliers).

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Elegir el plot correcto según el tipo de variables (continua/categórica) y el objetivo.
2. Ajustar marker, color, linestyle, alpha para legibilidad.
3. Construir histogramas con bins adecuados (regla de Freedman-Diaconis o 'auto').
4. Interpretar boxplot: mediana, Q1/Q3, whiskers, outliers.
5. Combinar bar + error bars para mostrar incertidumbre.

Temas

#	Tema	Por qué importa
1	Line: tendencias y series temporales	El más fácil de leer mal.
2	Scatter: relación entre dos variables	Con c= y s= para 3ª/4ª dimensión.
3	Bar y barh: categóricas	Vertical vs horizontal.
4	Histogram: distribución de una continua	Bins importan.
5	Boxplot: distribución resumida + outliers	Cuando hay muchos grupos.
6	Errorbar y fill_between	Mostrar incertidumbre.

Definiciones y características

Line plot

: Une puntos con líneas. Implica continuidad/orden en X — solo úsalo cuando X tiene orden natural (tiempo, espacio, secuencia).

Scatter

: Puntos no conectados. Muestra relación entre 2 continuas. Con c= codificas 3ª dim (color), con s= 4ª (tamaño). Más de 4 dims sobrecarga.

Bar / barh

: Barras para categóricas. Vertical (bar) si etiquetas son cortas, horizontal (barh) si son largas o muchas.

Histogram

: Distribución de UNA continua. Bins importan: pocos esconden estructura, muchos generan ruido. bins='auto' usa Freedman-Diaconis (buen default).

Boxplot

: Resumen de distribución: mediana (línea), Q1-Q3 (caja), whiskers (1.5×IQR), outliers (puntos). Útil para comparar muchos grupos rápido.

Errorbar

: Barra + línea vertical/horizontal indicando incertidumbre (std, IC95%). Sin esto, las barras mienten visualmente.

Dataset / recursos

Palmer Penguins. Sin descarga adicional.

Ejercicios

1. Line. Serie temporal de ventas mensuales (sintética). Anota máximo con flecha.

2. Scatter. body_mass vs bill_length, color por species. Adicionalmente: s= con flipper_length para tamaño.

3. Bar. Count por species, ordenado descendente. Vertical y horizontal — compara legibilidad.

4. Histogram. Distribución de body_mass con bins='auto' y bins=10. Compara.

5. Boxplot. body_mass por species: 3 cajas lado a lado. Identifica outliers.

Homework verificable

Notebook con penguins: (a) 5 plots básicos cada uno bien etiquetado; (b) scatter decorado con color y tamaño codificando 3 dimensiones; (c) bar con errorbars de std; (d) boxplot agrupado con interpretación de outliers.

Criterio de aceptación: Cada plot tiene título, labels, leyenda donde aplica. Bins justificados.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Line plot entre 2 categorías "A", "B"	Conectar categorías con línea engaña (sugi
Histograma con escala Y rara	Por default density=False (counts). Si qui
Boxplot todos iguales por outliers extremo	Outliers dominan visualmente; cajas quedan
Bar chart con colores random distrae	Sin codificación significativa, color = ru
Scatter con miles de puntos = blob negro	Overplotting. Fix: alpha=0.3, hexbin (plt.

Preguntas frecuentes

¿Pie chart cuándo?

Casi nunca. El ojo humano compara mal ángulos. Para proporciones: bar o stacked bar. Pie tolerable solo con 2-3 categorías y proporciones muy distintas.

¿Cuántos bins en un histograma?

bins='auto' (Freedman-Diaconis) es buen default. Si es estudios académicos: regla de Sturges (bins=int(np.log2(n)+1)). Experimenta con 10/30/50 si dudas.

¿Boxplot o violinplot?

Boxplot: rápido, 5 estadísticos, outliers claros. Violinplot: muestra distribución completa (multimodalidad). Para comparar 3-10 grupos, ambos OK. Para >10, boxplot gana en densidad.

¿Errorbars con std o con IC?

Std: dispersión natural de los datos. IC95% de la media: incertidumbre del estimador (más pequeño con N grande). Para inferencia, IC. Para describir, std.

¿Plot 3D buen idea?

Casi nunca. Oclusión + perspectiva engañan. 2D con color/tamaño suele comunicar mejor. Excepción: superficies analíticas z = f(x, y).

Referencias

- VanderPlas, cap. 4 §§ 4.2-4.5.
- matplotlib gallery

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 037 — Clase 037 — Matplotlib: subplots y gridspec

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 4 § 4.6 Multiple Subplots.*

> *Duración estimada: 60 min.*

Objetivo

Que el alumno organice múltiples plots en una sola figura — con plt.subplots(n, m) para grillas regulares y con GridSpec para layouts irregulares (un plot grande + varios pequeños). Crítico para informes y dashboards.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Crear grillas regulares con fig, axes = plt.subplots(2, 3, figsize=...).
2. Iterar sobre axes.flat para llenar la grilla con loops.
3. Compartir ejes con sharex=True, sharey=True para comparar.
4. Usar GridSpec para layouts irregulares (1 grande + 3 pequeños).
5. Usar constrained_layout=True en vez de tight_layout() (más confiable).

Temas

#	Tema	Por qué importa
1	plt.subplots(nrows, ncols)	Grilla regular.
2	Iterar con .flat	Llenar muchos plots en loop.
3	sharex/sharey	Comparar con misma escala.
4	GridSpec para layouts irregulares	1 grande + N pequeños.
5	constrained_layout vs tight_layout	El primero es mejor.
6	add_subplot con posiciones custom	Cuando necesitas full control.

Definiciones y características

plt.subplots(n, m)

: Crea figure + grilla regular de n filas × m columnas de axes. Devuelve (fig, axes) donde axes es array 2D — itera con .flat para llenar.

sharex / sharey

: Comparten escala entre axes del grid. Ideal para comparar distribuciones de la misma magnitud sin distorsión visual.

GridSpec

: Layout irregular avanzado: define grilla y asigna spans manualmente ("plot grande arriba + 3 pequeños abajo"). Mucho más flexible que subplots(n, m).

add_subplot(spec)

: Añade un axes a una figure según una posición específica (índice 121, o GridSpec). Útil cuando subplots no alcanza.

constrained_layout=True

: Algoritmo moderno (default en pandas 3+) que ajusta spacing automáticamente. Maneja colorbars, leyendas externas, suptitle sin overlap. Recomendado sobre tight_layout().

Dataset / recursos

Palmer Penguins. Sin descarga.

Ejercicios

1. Grilla 2×2. 4 histogramas de las 4 features numéricas de penguins en una figura.

2. Grilla con loop. Itera axes.flat para plot consistente.

3. sharey=True. 3 boxplots por species lado a lado con misma escala Y.

4. GridSpec irregular. Un scatter grande (2×2) + 1 hist arriba (1×2) + 1 hist a la derecha (2×1) — marginal histograms.

5. constrained_layout. Compara una figura compleja con tight_layout() vs constrained_layout=True — observa diferencia.

Homework verificable

Notebook con penguins: (a) grilla 2×2 hists; (b) 3 boxplots con sharey; (c) layout GridSpec con scatter central + marginales arriba/derecha; (d) misma figura comparando tight_layout vs constrained_layout.

Criterio de aceptación: Sin superposición de labels. Layouts limpios. Marginales alineadas al scatter.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
axes es array 1D cuando esperaba 2D	Si pides subplots(1, 3) o subplots(3, 1),
subplots(1, 1) devuelve axes escalar	No es array. Fix: si quieres array siempre
Plots se superponen / labels cortados	Sin constrained_layout=True ni tight_layou
sharey=True pero un plot tiene escala dist	Quizás esa subplot necesita un eje secunda
GridSpec con índices confusos	El orden es gs[fila, col] igual que NumPy.

Preguntas frecuentes

¿Cuándo subplots(n, m) vs GridSpec?

Regular: subplots. Irregular (un grande + varios pequeños, joint plot, dashboard): GridSpec.

¿fig.suptitle o ax.set_title con subplots?

suptitle = título de toda la figura (encima de todos). ax.set_title = título por subplot. Combinables.

¿Cómo controlo espacio entre subplots?

subplots(..., gridspec_kw={'hspace': 0.3, 'wspace': 0.3}) (ratios relativos al tamaño del axes). Con constrained_layout suele ser automático.

¿figsize cómo elegir?

Para artículos: ancho de columna (típicamente 3.5" para 1 col, 7" para ancho página). Para presentaciones: aspect ratio 16:9 → (12, 6.75).

¿axes.flat o axes.flatten()?

flat es iterador (no copia). flatten() devuelve nuevo array. Para iterar, flat ahorra memoria.

Referencias

- VanderPlas, cap. 4 § 4.6.
- GridSpec tutorial

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 038 — Clase 038 — Matplotlib: legends, colorbars, ticks, anotaciones

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 4 §§ 4.7–4.9 Customizing Legends, Colorbars, Ticks.*

> *Duración estimada: 60 min.*

Objetivo

Que el alumno controle los detalles que distinguen un plot ad-hoc de uno publicable: leyenda fuera del gráfico, colorbar discreto, ticks personalizados, y anotaciones (flechas, texto) para guiar la atención del lector.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Posicionar leyenda fuera del axes con bbox_to_anchor.
2. Configurar colorbar con label, ticks discretos, y categoría.
3. Personalizar ticks: rotación, formato (FuncFormatter, PercentFormatter), scale log.
4. Anotar puntos con ax.annotate(..., xy=..., xytext=..., arrowprops=...).
5. Añadir líneas de referencia con axhline/axvline (umbrales, medias).

Temas

#	Tema	Por qué importa
1	Legend con bbox_to_anchor	Sacarla del axes.
2	Colorbar con label y ticks discretos	Cuando hay codificación por color.
3	Tick formatters: percent, scientific, cust	Legibilidad.
4	ax.annotate con flecha	Resaltar un punto específico.

5	axhline / axvline / axhspan	Líneas y bandas de referencia.
6	Log scale: ax.set_yscale('log')	Cuando hay rango grande.

Definiciones y características

Leyenda (ax.legend)

: Mapeo labels↔elementos del plot. Auto-detecta si pones label= en plot/scatter. Posición con loc= o bbox_to_anchor= para colocarla fuera del axes.

Colorbar

: Escala de color para plots con codificación continua (scatter(c=valor), imshow). Indica cómo se mapean valores a colores. Siempre etiqueta con cbar.set_label(...).

Tick / TickFormatter

: Marcas en los ejes y sus etiquetas. Formatter define el formato: PercentFormatter, FuncFormatter(lambda x,p: f'${x:,.0f}'), LogFormatter.

ax.annotate

: Texto con flecha apuntando a un punto. Parámetros: xy (punto a apuntar), xytext (posición del texto), arrowprops (estilo flecha).

Líneas/bandas de referencia

: axhline(y) horizontal, axvline(x) vertical, axhspan(y1, y2) banda horizontal. Útiles para umbrales, medias, eventos.

Dataset / recursos

Sintético: serie con outliers, scatter con categorías.

Ejercicios

1. Leyenda fuera. Plot con 5 líneas, leyenda a la derecha fuera del axes.

2. Colorbar. Scatter con c= continuo (ej: density), colorbar con label.

3. PercentFormatter. Bar chart con eje Y formateado como porcentaje.

4. Anotar outlier. Scatter con un punto extremo; flecha + texto identificándolo.

5. Log scale. Plot de valores con rango grande (1, 10, 100, 1000); compara linear vs log.

Homework verificable

Notebook con: (a) plot multi-línea con leyenda externa; (b) scatter con colorbar etiquetado; (c) bar % usando PercentFormatter; (d) plot con anotación de máximo via flecha; (e) comparativa lineal vs log en datos exponenciales.

Criterio de aceptación: Cada elemento visual tiene propósito. Anotaciones legibles, no superpuestas.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Leyenda tapa parte del plot	Default es 'best' que a veces no encuentra
Colorbar es enorme/desproporcionada	Default fill toda la altura. Fix: fig.colo
Ticks rotados se cortan	Texto rotado no entra en el bounding box.

annotate con flecha que apunta mal	xy y xytext están en coordenadas de datos
axhline(media) invisible	Pintada gris claro encima de fondo similar

Preguntas frecuentes

¿Leyenda dentro o fuera?

Dentro si hay espacio (1-3 series, plot no saturado). Fuera (bbox_to_anchor) si son 5+ series o el plot está denso.

¿Colorbar discreta o continua?

Continua si los datos son continuos (densidad, precio). Discreta (con N boundaries) si son categóricas/cuantiles — BoundaryNorm + ListedColormap.

¿set_xticks o set_xticklabels?

set_xticks define dónde van las marcas. set_xticklabels define qué texto se muestra. Usa ambos juntos para control fino; nunca uses solo el segundo (deprecation warning).

¿Cómo añado emojis/Unicode a texts?

Default font (DejaVu Sans) trae básicos. Para emoji color, fuente especial ('Segoe UI Emoji' en Windows). Mejor: evita emojis dentro del plot, úsalos en título/labels markdown.

¿Log scale para presentaciones?

Si los datos cubren >2 órdenes de magnitud (1, 100, 10000), sí. Pero siempre indícalo en el título o axis label ("escala log") — no es obvio.

Referencias

- VanderPlas, cap. 4 §§ 4.7-4.9.
- matplotlib text and annotations

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 039 — Clase 039 — Matplotlib: stylesheets

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 4 § 4.11 Customizing Matplotlib: Configurations and Stylesheets.*

> *Duración estimada: 30 min.*

Objetivo

Que el alumno aproveche stylesheets built-in y propios para mantener consistencia visual entre plots y proyectos — y deje de configurar manualmente rcParams en cada notebook.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Listar stylesheets disponibles con plt.style.available.
2. Aplicar un style globalmente (plt.style.use(...)) o solo a un bloque (with plt.style.context(...)).
3. Crear style propio en un archivo .mplstyle y usarlo.
4. Combinar styles (uno + ajustes manuales).
5. Elegir style según contexto (informe, presentación, B&N para impresión).

Temas

#	Tema	Por qué importa
1	plt.style.available	Catálogo built-in.
2	plt.style.use(...) global	Afecta todos los plots subsiguientes.
3	with plt.style.context(...)	Temporal, ideal para un bloque.
4	Archivo .mplstyle propio	Reusar entre proyectos.
5	Stylesheets comunes	default, seaborn-v0_8-whitegrid, ggplot, f
6	rcParams override puntual	Style + ajuste fino.

Definiciones y características

Stylesheet

: Conjunto de rcParams predefinidos en un archivo .mplstyle. Activa con plt.style.use('nombre'). Cambia colors, fonts, grids, spines, etc., consistentemente.

plt.style.available

: Lista de stylesheets built-in: default, ggplot, seaborn-v0_8-whitegrid, bmh, grayscale, dark_background, etc.

plt.style.context(name)

: Aplica style solo dentro de un with block; al salir, vuelve al anterior. Útil cuando quieres style distinto para 1 plot sin afectar el resto.

.mplstyle propio

: Archivo de texto con clave: valor (igual sintaxis que rcParams). Ubícalo donde sea y carga con plt.style.use('/ruta/al.mplstyle').

rcParams override en context

: with plt.rc_context({'figure.figsize': (12, 6)}): aplica overrides temporales sobre el style activo.

Dataset / recursos

Sintético: mismo plot en varios styles. Sin descarga.

Ejercicios

1. Catalogo. Imprime plt.style.available. Identifica 5 que suenen útiles.

2. Galería visual. Mismo scatter plot bajo 4 styles distintos (default, ggplot, seaborn-whitegrid, grayscale).

3. Bloque temporal. Con with plt.style.context('seaborn-v0_8-darkgrid'): aplica style solo a 1 figura.

4. Style propio. Crea mi_style.mplstyle con tus defaults preferidos. Úsalo.

5. Style + override. Aplica ggplot y luego cambia figure.figsize para un plot específico.

Homework verificable

Notebook: (a) galería de 4 styles sobre un mismo dataset; (b) crear informe.mplstyle con paleta corporativa simulada (3 colores principales); (c) demo de uso temporal con plt.style.context; (d) comparativa B&N (grayscale) vs color para una figura que podría imprimirse.

Criterio de aceptación: Galería con plots reconocibles; style propio aplica colores definidos.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
plt.style.use('seaborn') da error	Renombraron a seaborn-v0_8-whitegrid (y va
Style activado pero un plot no lo respeta	Aplicado después de crear el axes. Style a
Colors definidos en .mplstyle no aparecen	Sintaxis cycler incorrecta. Fix: axes.prop
dark_background rompe scatter color	Background oscuro, color de marker default
Style se mantiene tras cerrar notebook	style.use afecta rcParams globales del ker

Preguntas frecuentes

¿Style propio o usar built-in?

Built-in para empezar. Propio cuando tu organización/proyecto tiene paleta corporativa o convenciones específicas.

¿Style afecta seaborn?

Seaborn maneja su propio system con sns.set_theme(). Coordinar ambos puede colisionar. Recomendado: elige uno (style de matplotlib O sns.set_theme).

¿Cómo veo todos los styles?

plt.style.available lista todos. Para galería visual: matplotlib style sheets reference.

¿bmh qué es?

Style del libro Bayesian Methods for Hackers — popular para gráficos estadísticos limpios.

¿Style para impresión B&N?

grayscale convierte automáticamente. Pero verifica: cmaps con poca variación de luminancia ('jet') quedan ilegibles. Mejor 'viridis' o linestyles distintos.

Referencias

- VanderPlas, cap. 4 § 4.11.
- matplotlib stylesheets gallery

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 040 — Clase 040 — Matplotlib: 3D plotting

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 4 § 4.12 Three-Dimensional Plotting in Matplotlib.*

Duración estimada: 45 min.

Objetivo

Que el alumno sepa cuándo (raramente) usar 3D y cómo hacerlo bien: scatter 3D, superficies (plot_surface), wireframes y contornos. Spoiler: la mayoría de las veces un buen 2D + color comunica mejor.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Crear axes 3D con projection='3d'.
2. Scatter, line, surface, wireframe, contour en 3D.
3. Controlar ángulo de vista con ax.view_init(elev, azim).
4. Reconocer cuándo NO usar 3D: la mayoría de las veces hay una alternativa 2D mejor.

Temas

#	Tema	Por qué importa
1	projection='3d'	Habilita el 3D toolkit.
2	Scatter 3D con codificación por color	3 dims + 4 (color).
3	plot_surface para z = f(x, y)	Funciones bivariadas.
4	plot_wireframe y contour3D	Alternativas más simples.
5	view_init: rotar interactivo	En notebooks con %matplotlib widget.
6	Cuándo NO usar 3D	Casi siempre.

Definiciones y características

projection='3d'
: Parámetro al crear axes (fig.add_subplot(111, projection='3d')) que habilita el toolkit 3D (mplot3d). Sin esto, las llamadas 3D fallan.

plot_surface(X, Y, Z)
: Superficie 3D para z = f(x, y). Requiere mesh: X, Y = np.meshgrid(x, y); Z = f(X, Y). Soporta cmap para color por valor de Z.

plot_wireframe(X, Y, Z)
: Como surface pero solo líneas — más liviano, mejor para densidad alta, peor para forma.

view_init(elev, azim)
: Define ángulo de cámara. elev elevación (0=horizontal, 90=vista superior). azim rotación horizontal en grados. Animar varia azim para 360°.

Oclusión
: Limitación fundamental del 3D: objetos al frente tapan los del fondo, sin forma confiable de elegir qué ver. La razón principal por la que 3D es problemático.

Dataset / recursos

Sintético: superficie analítica + nube de puntos. Sin descarga.

Ejercicios

1. Scatter 3D. 200 puntos con coords (x, y, z) y color por una 4ª variable.

2. Superficie. z = sin(sqrt(x² + y²)) en mesh 50×50. plot_surface con colormap.

3. Wireframe + contour. Misma función con plot_wireframe. Compara legibilidad con superficie llena.

4. view_init. Cambia (elev, azim) a 4 ángulos y graba una grilla 2×2.

5. Reto: 2D que vence al 3D. Para tu scatter 3D del ejercicio 1, propón un 2D + color/tamaño que comunique igual o mejor.

Homework verificable

Notebook: (a) scatter 3D con 4 dimensiones (xyz + color); (b) superficie z=f(x,y); (c) wireframe del mismo z; (d) grilla 2×2 con 4 view_init distintos; (e) ejercicio de "2D vence al 3D": versión 2D del scatter.

Criterio de aceptación: Plots 3D legibles (no espagueti). Versión 2D del scatter comparable.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
projection='3d' da error "unknown projecti	No importaste el toolkit. Fix: from mpl_to
Surface plot tarda mucho con mesh grande	100×100 mesh = 10k vertices. Fix: reduce r
3D scatter con miles de puntos = mancha	Oclusión. Fix: reduce N (sampling), o usa
view_init no se aplica si llamado después	El orden importa: configura ángulo ANTES d
Labels Z se cortan o solapan	3D tiene limitaciones de layout. Fix: ajus

Preguntas frecuentes

¿Realmente necesito 3D?

Pregúntate: ¿hay una versión 2D con color/tamaño que comunique igual? Casi siempre sí. 3D vale para superficies analíticas (z = f(x,y)), datos físicos 3D, o cuando es interactivo (rotable).

¿%matplotlib widget para rotar interactivo?

Sí — en JupyterLab/Notebook permite rotar con el mouse. Requiere pip install ipympl. Default inline da imagen estática.

¿plotly 3D mejor?

Sí para uso interactivo en web/dashboard. No para reportes estáticos (mismo problema de oclusión, peso del HTML).

¿Animaciones 3D?

matplotlib.animation.FuncAnimation + variando view_init por frame. Bonito pero costoso de generar. Considera plotly que es interactivo nativo.

¿Axes3D deprecated?

En matplotlib moderno, basta subplot_kw={'projection': '3d'} — no necesitas importar Axes3D explícitamente.

Referencias

- VanderPlas, cap. 4 § 4.12.
- matplotlib mplot3d tutorial

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 041 — Clase 041 — Seaborn: distribuciones, relaciones, categóricas, facetas

> *Parte: 0 — Prerrequisitos · Fuente: VanderPlas, cap. 4 § 4.13 Visualization with Seaborn · seaborn docs.*
>
> *Duración estimada: 75 min.*

Objetivo

Que el alumno use seaborn cuando aporta sobre matplotlib puro: defaults estéticos, API tipada para DataFrames (x=, y=, hue=, col=), distribuciones (histplot, kdeplot, displot), relaciones (scatterplot, lmplot), categóricas (boxplot, violinplot, swarmplot), y facetas (grilla automática por categoría).

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Usar la API moderna (figure-level vs axes-level) y elegir la correcta.
2. Construir un pairplot para EDA rápido de un DataFrame.
3. Codificar 3 dimensiones con hue, style, size.
4. Hacer facetas con col= y row= para grillas automáticas.
5. Personalizar themes con sns.set_theme(style=..., palette=...).

Temas

#	Tema	Por qué importa
1	seaborn vs matplotlib	Seaborn es matplotlib + defaults + API tip
2	Figure-level (displot, relplot, catplot) v	Cuándo cada uno.
3	hue, style, size	Codificar dimensiones extra.
4	Facetas con col, row	Grilla automática por categoría.
5	pairplot para EDA	Matriz de scatters.
6	Themes y paletas	Defaults consistentes.

Definiciones y características

Seaborn

: Wrapper sobre matplotlib con (1) defaults estéticos mejores, (2) API tipada para DataFrames (x=, y=, hue=), (3) plots estadísticos directos (regresión, distribuciones, facetas).

Figure-level (displot, relplot, catplot)

: Funciones que crean su propia figura, soportan facetas (col=, row= para grilla automática). Más alto nivel, menos control fino.

Axes-level (histplot, scatterplot, boxplot)

: Funciones que dibujan en un ax que tú pasas. Más bajo nivel, integran con layouts custom de matplotlib.

hue, style, size

: Codifican dimensiones extra: hue (color por categoría), style (marker por categoría), size (tamaño por valor continuo).

pairplot

: Matriz de scatterplots de todas las parejas de variables numéricas, diagonal con KDE/histograma. EDA visual en una línea.

Faceta (col/row)

: Una sub-figura por cada valor de una categórica. relplot(..., col='species', row='sex') produce grilla n_species × n_sex de plots automática.

Dataset / recursos

Palmer Penguins (seaborn lo trae built-in via sns.load_dataset('penguins')).

Ejercicios

1. Pairplot. Penguins, color por species. EDA en 1 línea.

2. Scatter con hue + size. body_mass vs flipper, hue por species, size por bill_length.

3. KDE distribución. body_mass por species (3 KDE en mismo plot).

4. Boxplot + swarm. Combinar boxplot con swarm para ver puntos individuales.

5. Facetas. sns.relplot(...col='species', row='sex') para 3×2 = 6 subplots automáticos.

Homework verificable

Notebook con penguins: (a) pairplot completo; (b) violin + swarm de body_mass por (species, sex); (c) faceta 2×3 de scatter; (d) tema custom + paleta; (e) decisión documentada: cuándo usar figure-level vs axes-level.

Criterio de aceptación: Plots de EDA legibles. Decisiones de hue/style/col justificadas.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Mezclo sns.set_theme() con plt.style.use(.	Compiten por los rcParams. Fix: elige uno.
Figure-level no me deja agregar ax.set_tit	Devuelve FacetGrid, no axes. Fix: g.set_ti
pairplot con muchas columnas tarda eternid	N² scatters; con 20 cols son 400 plots. Fi
hue con muchas categorías → leyenda enorme	Default muestra todas las categorías. Fix:
sns.scatterplot(...) no aparece en noteboo	Falta capturar return o plt.show(). Fix: a

Preguntas frecuentes

¿seaborn o matplotlib puro?

Seaborn para plots estadísticos rápidos con DataFrame (hue, facetas, defaults). Matplotlib para control fino, customización extrema, o plots que no son estadísticos.

¿Cuándo figure-level vs axes-level?

Figure-level si quieres facetas o el plot es el output completo. Axes-level si necesitas integrar con layout custom (subplots manuales).

¿pairplot o corrmatrix?

pairplot muestra relaciones visualmente (no lineales, outliers, clusters). Heatmap de corr() muestra coeficiente Pearson (solo lineal). Complementarios.

¿Tema corporativo en seaborn?

sns.set_theme(palette=['#0F766E', '#D9A441', '#7C3AED']) o palette custom: sns.color_palette('husl', n_colors=5). Combina con style='whitegrid'.

¿Seaborn maneja datasets grandes (>1M)?

Plots scatter/hist sí. Pairplot con muchas cols se vuelve lento. Para datasets enormes, considera datashader o downsample antes.

Referencias

- VanderPlas, cap. 4 § 4.13.
- seaborn user guide
- Waskom, seaborn paper (JOSS, 2021)

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 042 — Clase 042 — Visualización geográfica (Plotly / folium)

> *Parte: 0 — Prerrequisitos · Fuente: Plotly Choropleth docs · folium docs · Cartographies of the Mind (background).*

> *Duración estimada: 60 min.*

Objetivo

Que el alumno construya mapas básicos cuando los datos tienen componente geográfico: folium (mapas Leaflet interactivos, markers, choropleth), plotly (choropleth, scatter geo). Sin entrar a GIS profundo (eso es geopandas, fuera del scope de Parte 0).

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Crear mapa folium centrado, con tile layer básico.
2. Añadir markers con popup, tooltip, color según valor.
3. Construir choropleth (mapa de calor por región) con folium o plotly.
4. Decidir entre folium y plotly geo según destino (HTML standalone vs dashboard).
5. Citar fuentes de tiles y GeoJSON públicos.

Temas

#	Tema	Por qué importa
1	Sistemas de coordenadas: lat/lng	Convención: lat primero en folium, lng pri
2	folium: mapa + markers + popups	Mapas Leaflet en notebook.

3	folium choropleth con GeoJSON	Mapas de calor por país/región.
4	plotly choropleth y scatter_geo	Cuando ya usas plotly.
5	Tile providers (OSM, CartoDB)	Estética y licencia.
6	Cuándo geopandas	Análisis geoespacial real.

Definiciones y características

Coordenadas geográficas (lat, lng)

: Latitud (-90 a 90): N/S del ecuador. Longitud (-180 a 180): E/W de Greenwich. Convención y orden varía: folium [lat, lng], GeoJSON [lng, lat] — fuente clásica de bugs.

folium

: Wrapper Python de Leaflet.js. Mapas interactivos (zoom, pan, popups) embebidos en notebook como HTML. Ideal para exploración.

plotly geo

: Mapas en plotly (scatter_geo, choropleth). Bonitos, interactivos, integran con dashboards Plotly/Dash. Para choropleth de países usa códigos ISO-3.

Choropleth

: Mapa de calor por región: color de cada polígono representa un valor (PIB, población, casos). Requiere GeoJSON con las shapes.

Tile provider

: Servidor de "baldosas" (imágenes 256×256) que componen el mapa de fondo. OpenStreetMap (default), CartoDB (limpio), Mapbox (premium).

geopandas

: Extensión de pandas con geometrías (Shapely). Para análisis espacial (intersection, buffer, dissolve), no solo visualizar. Fuera del scope Parte 0.

Dataset / recursos

Sintético: lista de ciudades con coords + métrica simulada. GeoJSON de países público desde un CDN para choropleth.

Ejercicios

1. Mapa con markers. 5 ciudades españolas con marker y popup mostrando nombre + población.

2. Markers coloreados. Mismo, pero color verde si pop>1M, rojo si <500k.

3. Choropleth folium. Mapa mundial con un valor sintético por país (ej: PIB).

4. Choropleth plotly. Lo mismo con plotly.express.choropleth.

5. Comparar. ¿Cuándo folium (mapa físico explorable) vs plotly (integra con dashboard)?

Homework verificable

Notebook: (a) mapa folium con 10+ markers + popups + tooltips; (b) choropleth folium de un dataset por país; (c) mismo choropleth con plotly express; (d) reporte 1-párrafo comparando ambos.

Criterio de aceptación: Mapas funcionales en notebook; popups muestran info correcta; choropleth con leyenda.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Marcadores aparecen en medio del océano	Invertiste lat/lng. Fix: folium espera [la
Mapa folium no se renderiza en VS Code	Algunas extensiones de notebook no muestra
choropleth plotly sale gris	El código ISO no matchea con el shape. Fix
Mapa pesa MB y carga lento	Demasiados markers (>1000). Fix: usa Marke
Tiles fallan a cargar	Provider con rate limit (Mapbox sin token,

Preguntas frecuentes

¿folium o plotly?

folium para exploración interactiva en notebook (mapa real con zoom/pan/popups). plotly para integrar en dashboard o cuando ya usas plotly. Ambos producen HTML standalone.

¿Cómo encuentro lat/lng de un lugar?

Click derecho en Google Maps → "¿Qué hay aquí?". O geocodifica con geopy (from geopy.geocoders import Nominatim).

¿GeoJSON dónde lo consigo?

Para países: Natural Earth (público). Para regiones: portal gov del país. Para custom: dibuja en geojson.io.

¿Cuándo necesito geopandas?

Análisis espacial real: "¿cuántos puntos caen en este polígono?", "buffer de 1km alrededor", re-proyección entre CRS. Para solo visualizar puntos en mapa: folium basta.

¿Mapas offline?

Tiles son online por default. Para offline: descarga tiles con mbtiles, sirve local con folium.raster_layers.TileLayer(url='file://...'). Setup complejo, considera si vale.

Referencias

- folium docs
- plotly choropleth docs
- Natural Earth GeoJSON

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 043 — Clase 043 — SQL fundamental: SELECT, WHERE, JOIN, GROUP BY, HAVING

> *Parte: 0 — Prerrequisitos · Fuente: SQL for Data Scientists (Tanimura) caps. 1-3 · SQLite docs · DuckDB docs.*

> *Duración estimada: 120 min.*

Objetivo

Que el alumno escriba consultas SQL no triviales — SELECT con filtros, JOINs (inner/left), agregaciones con GROUP BY y filtros sobre agregados con HAVING. Y entienda el orden de ejecución lógico (FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY → LIMIT), que es lo que confunde a todo el mundo al principio.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Escribir SELECT con filtros WHERE y operadores (=, <>, IN, BETWEEN, LIKE, IS NULL).
2. Hacer JOIN (INNER, LEFT, RIGHT, FULL) y reconocer cuándo cada uno.
3. Agrupar y agregar con GROUP BY + COUNT, SUM, AVG, MAX, MIN.
4. Filtrar agregados con HAVING (no se puede con WHERE).
5. Recitar el orden lógico: FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY → LIMIT.

Temas

#	Tema	Por qué importa
1	SELECT, FROM, WHERE	Lo básico, sin trampa.
2	Operadores WHERE	=, <>, IN, BETWEEN, LIKE, IS NULL.
3	JOINs (inner/left/right/full)	Análogos a pandas merge.
4	GROUP BY + agregadas	COUNT/SUM/AVG/MAX/MIN.
5	HAVING vs WHERE	HAVING filtra después de GROUP BY.
6	ORDER BY, LIMIT, OFFSET	Final del pipeline.
7	Orden lógico ≠ orden escrito	El gran malentendido.

Definiciones y características

SQL (Structured Query Language)

: Lenguaje declarativo para bases de datos relacionales. Describes qué quieres (no cómo) y el motor lo ejecuta. Estandarizado pero con dialectos (SQLite, PostgreSQL, MySQL, BigQuery).

Orden lógico vs escrito

: Escribes: SELECT-FROM-WHERE-GROUP-HAVING-ORDER. Ejecuta: FROM-WHERE-GROUP-HAVING-SELECT-ORDER-LIMIT. Por eso WHERE SUM(...) falla (aún no agrupado) — usa HAVING.

JOIN

: Combina filas de 2+ tablas por una key. Tipos: INNER (intersección), LEFT (todo left + match right), RIGHT (espejo), FULL OUTER (todo unión), CROSS (producto cartesiano).

GROUP BY + HAVING

: GROUP BY: agrupa filas por valor(es). HAVING: filtra los grupos después de agregar (no se puede con WHERE).

Funciones de agregación

: Operan sobre grupos: COUNT(*), COUNT(DISTINCT x), SUM, AVG, MIN, MAX, STDDEV. Devuelven UN valor por grupo.

DuckDB

: Motor SQL embebido (como SQLite) pero columnar y optimizado para analytics. Lee CSV/Parquet directo (FROM 'file.csv'). Drop-in para queries analíticas, mucho más rápido que SQLite en agregados.

Dataset / recursos

SQLite en memoria con 2 tablas sintéticas: clientes (10 filas) y ordenes (30 filas). Generado en el notebook con sqlite3 stdlib. Sin descarga.

Ejercicios

1. SELECT básico. Lista de clientes con país = 'ES'.

2. JOIN. Cada orden con el nombre del cliente.

3. LEFT JOIN. Todos los clientes, sumando órdenes (NaN si no tienen).

4. GROUP BY + HAVING. Clientes con más de 3 órdenes y monto total > 200.

5. Orden lógico. Explica con tus palabras por qué WHERE total > 100 no funciona si total es SUM(monto) — necesitas HAVING.

Homework verificable

Notebook con SQLite en memoria: (a) crea 2 tablas y carga datos sintéticos; (b) 5 consultas progresivas (filter, join, group, having, top-N); (c) explica el orden lógico con un ejemplo; (d) mismo ejercicio con DuckDB (sustituye sqlite3.connect(':memory:')).

Criterio de aceptación: Las 5 consultas producen el resultado esperado; DuckDB devuelve igual.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
column "x" must appear in GROUP BY clause	Seleccionaste col no agregada ni en GROUP
WHERE SUM(monto) > 100 falla	WHERE corre ANTES de GROUP. Fix: usa HAVIN
INNER JOIN pierde filas que esperaba ver	La key no matchea (NULL, tipos, espacios).
COUNT(col) devuelve menos que COUNT(*)	COUNT(col) ignora NULL en esa columna. Fix
SELECT * después de JOIN trae cols duplica	Ambas tablas tienen id. Fix: aliasea: SELE

Preguntas frecuentes

¿COUNT(*) o COUNT(1)?

Equivalentes en motores modernos (parser optimiza). COUNT(*) es más legible — úsalo.

¿Cuándo DISTINCT?

Cuando hay filas duplicadas que no deberían contarse. Cuidado: en SELECT con muchas cols puede ser caro. Mejor agrupa con GROUP BY si vas a agregar después.

¿UNION o UNION ALL?

UNION quita duplicados (más caro). UNION ALL mantiene todo (más rápido). Usa ALL si sabes que no hay duplicados (más común).

¿SQLite o PostgreSQL para aprender?

SQLite para arrancar (sin servidor, 1 archivo). El SQL es 90% igual. Migras a PostgreSQL cuando necesites: tipos avanzados, concurrencia, escala, JSON nativo.

¿Cómo trato fechas en SQL?

Cada motor su dialecto. SQLite: strings ISO '2024-01-15' + date(), strftime(). PostgreSQL: tipo DATE/TIMESTAMP nativo. Estándar: DATE '2024-01-15'.

Referencias

- Tanimura, SQL for Data Scientists, caps. 1-3.
- SQLite SELECT docs
- DuckDB docs

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 044 — Clase 044 — SQL avanzado: CTEs, window functions, subqueries correlacionadas

> *Parte: 0 — Prerrequisitos · Fuente: Tanimura, SQL for Data Scientists caps. 4-5 · PostgreSQL docs (window functions).*

> *Duración estimada: 120 min.*

Objetivo

Que el alumno escriba SQL legible y potente: CTEs (WITH) para descomponer queries complejas, window functions (OVER) para rankings/totales corridos/lag/lead sin perder filas, y subqueries correlacionadas cuando aportan.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Escribir CTEs con WITH name AS (...) para mejorar legibilidad.
2. Encadenar múltiples CTEs: WITH a AS (...), b AS (...) SELECT ....
3. Aplicar window functions: ROW_NUMBER(), RANK(), LAG(), LEAD(), SUM() OVER (PARTITION BY ... ORDER BY ...).
4. Calcular ranking por grupo con ROW_NUMBER() OVER (PARTITION BY ...).
5. Diferenciar subquery (independiente) vs correlacionada (depende de la outer).

Temas

#	Tema	Por qué importa
1	CTEs: WITH name AS (...)	Descomponer queries largas.
2	Múltiples CTEs encadenadas	Pipeline legible.
3	Recursive CTEs	Jerarquías, grafos.

4	Window functions: OVER (PARTITION BY ...	Agregar sin colapsar filas.
5	ROW_NUMBER, RANK, DENSE_RANK	Diferencias sutiles.
6	LAG, LEAD: comparar con fila anterior/sigu	Series temporales.
7	Subqueries correlacionadas	Cuando la subquery depende de la outer.

Definiciones y características

CTE (Common Table Expression)

: Vista temporal dentro de una query con WITH nombre AS (...). Descompone queries complejas en pasos legibles. Puedes encadenar múltiples: WITH a AS (...), b AS (...) SELECT ....

Recursive CTE

: CTE que se referencia a sí misma. Útil para jerarquías (árbol organizacional), grafos, generar series (calendario diario). Sintaxis: WITH RECURSIVE t AS (caso_base UNION ALL caso_recursivo).

Window function

: Agregación que no colapsa filas — añade el resultado por fila. Sintaxis: FUNC() OVER (PARTITION BY col ORDER BY col2). Ejemplos: ROW_NUMBER, RANK, LAG, LEAD, SUM() OVER (...).

PARTITION BY vs GROUP BY

: PARTITION BY (en window): subgrupos para la función, pero mantiene cada fila. GROUP BY: reduce a una fila por grupo.

LAG / LEAD

: Acceden a la fila anterior/siguiente dentro de la partition. LAG(monto, 1) OVER (PARTITION BY cliente ORDER BY fecha). Útil para diffs, growth rates.

Subquery correlacionada

: Subquery que depende de la outer query (referencia sus columnas). Se ejecuta una vez por cada fila de la outer. Más lenta que JOIN equivalente.

Dataset / recursos

SQLite con ordenes (cliente_id, fecha, monto) de clase 041 — extendido. Sin descarga.

Ejercicios

1. CTE básica. Reescribe una query con subquery anidada usando WITH.

2. ROW_NUMBER por grupo. Top-1 orden por cliente (mayor monto).

3. Total corrido. SUM(monto) OVER (PARTITION BY cliente_id ORDER BY fecha) — total acumulado por cliente.

4. LAG. Por cliente, diferencia entre el monto actual y el anterior.

5. Recursive CTE. Genera serie de fechas día a día desde 2024-01-01 a 2024-01-31.

Homework verificable

Notebook: (a) 3 versiones de la misma query (anidada → CTE → CTEs múltiples) comparando legibilidad; (b) top-3 órdenes por cliente con ROW_NUMBER; (c) total corrido y delta vs orden anterior; (d) recursive CTE para calendario diario.

Criterio de aceptación: Las 3 versiones devuelven exactamente el mismo resultado. Window functions sin

error.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
syntax error at or near "OVER"	Motor sin soporte de window functions (SQL
ROW_NUMBER() da números repetidos	Olvidaste OVER (...). Sin él, no es window
CTE recursiva nunca termina	Caso base falta o caso recursivo no conver
LAG(x) OVER (ORDER BY fecha) da NULL en la	Comportamiento esperado — no hay fila ante
CTE da mismo resultado pero más lento que	Algunos motores no inlineaban CTEs (Postgr

Preguntas frecuentes

¿CTE o subquery?

CTE si el lector necesita entender qué hace cada paso (legibilidad). Subquery si es trivial y de un solo uso. Para queries >10 líneas, CTE casi siempre gana.

¿ROW_NUMBER, RANK o DENSE_RANK?

Para valores [10, 20, 20, 30]: ROW_NUMBER [1,2,3,4] (siempre único). RANK [1,2,2,4] (huecos). DENSE_RANK [1,2,2,3] (sin huecos). Elige según semántica.

¿Window function es lo mismo que groupby+merge en pandas?

Conceptualmente sí — g.transform(...) en pandas hace lo equivalente. Window functions son la versión SQL más eficiente.

¿Cuándo subquery correlacionada vs JOIN?

Casi siempre JOIN o window function es más rápido. Correlacionada solo cuando no tiene equivalente JOIN (raro) o el optimizador del motor la maneja bien (motores modernos).

¿Top-N por grupo?

Patrón estándar: WITH ranked AS (SELECT , ROW_NUMBER() OVER (PARTITION BY grupo ORDER BY metric DESC) rn FROM tabla) SELECT FROM ranked WHERE rn <= N.

Referencias

- Tanimura, SQL for Data Scientists, caps. 4-5.
- PostgreSQL window functions tutorial
- Modern SQL — CTEs

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 045 — Clase 045 — SQL desde Python: sqlite3, SQLAlchemy, DuckDB

> *Parte: 0 — Prerrequisitos · Fuente: Python stdlib sqlite3 · SQLAlchemy docs · DuckDB Python docs.*

> *Duración estimada: 75 min.*

Objetivo

Que el alumno conecte Python con SQL de las 3 formas que va a encontrar en producción: sqlite3 (stdlib, demo local), SQLAlchemy (ORM/engine genérico para PostgreSQL/MySQL), y DuckDB (columnar embebido para análisis sobre CSV/Parquet sin servidor).

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Conectar y consultar con sqlite3 stdlib, usando placeholders ? (NUNCA concatenar SQL).
2. Usar SQLAlchemy create_engine(URL) + pd.read_sql para queries a cualquier RDBMS.
3. Usar DuckDB para hacer SQL sobre DataFrames y CSV/Parquet directamente.
4. Prevenir SQL injection con queries parametrizadas.
5. Decidir entre sqlite/SQLAlchemy/DuckDB según el caso.

Temas

#	Tema	Por qué importa
1	sqlite3 stdlib: connect, cursor, fetchall	Para demos y BDs ligeras.
2	Placeholders ? y :nombre	NUNCA concatenar strings.
3	SQLAlchemy create_engine('postgresql://...	Soporta todos los RDBMS.
4	pd.read_sql y df.to_sql	Pasarela pandas ↔ BD.
5	DuckDB: SQL sobre DataFrames y archivos	duckdb.query('SELECT ... FROM df').
6	Cuándo cada uno	Trade-offs.

Definiciones y características

sqlite3 (stdlib)

: Driver Python para SQLite. Sin dependencias externas. Patrón: connect(...) → cursor() → execute(sql, params) → fetchall().

Parameterized query (? o :name)

: Placeholder donde el driver substituye valores escapados. Única forma segura de pasar datos de usuario — previene SQL injection.

SQLAlchemy

: Toolkit ORM + Core para Python. Backend-agnostic: cambias el URL del engine y migras entre SQLite/Postgres/MySQL sin tocar queries. create_engine('postgresql://...').

DuckDB

: OLAP DB embebida. SQL sobre DataFrames pandas (duckdb.query('SELECT ... FROM df')) y archivos CSV/Parquet directos (FROM 'data.csv'). Mucho más rápido que sqlite3 para analytics.

SQL injection

: Inyección de SQL malicioso via concatenación de strings con input de usuario. Prevención: SIEMPRE parameterized queries, nunca f-string con valores externos.

pd.read_sql / df.to_sql

: Pasarela pandas↔BD. Acepta connection o engine. read_sql_query para queries complejas; read_sql_table para tablas completas.

Dataset / recursos

Penguins descargado a CSV local para DuckDB; datos sintéticos para sqlite/SQLAlchemy.

Ejercicios

1. sqlite3 con placeholders. Crea tabla, inserta 5 filas usando executemany con tuples, consulta con ? placeholder. Demuestra el bug si concatenas.

2. df.to_sql y pd.read_sql. Carga un DataFrame a SQLite y consulta de vuelta.

3. SQLAlchemy engine. Crea engine SQLite. Usa pd.read_sql con engine.

4. DuckDB sobre DataFrame. Carga penguins en df. duckdb.query('SELECT species, AVG(body_mass_g) FROM df GROUP BY species').df().

5. DuckDB sobre CSV. Mismo query pero FROM 'penguins.csv' directo, sin cargar a pandas.

Homework verificable

Notebook con 3 backends del mismo análisis: (a) sqlite3 stdlib + cursor; (b) SQLAlchemy engine + pd.read_sql; (c) DuckDB sobre CSV. Documenta cuándo elegirías cada uno. Demuestra explícitamente el peligro de SQL injection con concatenación vs placeholders.

Criterio de aceptación: Las 3 versiones devuelven el mismo resultado. Demo de injection sin daño real.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
OperationalError: no such table: X después	Falta con.commit(). sqlite3 no auto-commit
Concatené input de usuario en query y func	Hasta que el usuario malicioso prueba '; D
SQLAlchemy 2.0 — Engine.execute no existe	API cambió: ahora with engine.connect() as
DuckDB lee CSV pero pierde tipos	Pandas adivina mejor. Fix: duckdb.read_csv
pd.read_sql lento con N grande	Driver Python carga todo a Python. Fix: ch

Preguntas frecuentes

¿sqlite3, SQLAlchemy o DuckDB?

sqlite3 para demos/tests/scripts locales. SQLAlchemy para producción con Postgres/MySQL. DuckDB para EDA sobre CSV/Parquet sin servidor — el más rápido para analytics.

¿pd.read_sql es seguro contra injection?

Sí si pasas params: read_sql('SELECT * FROM t WHERE x=:val', con, params={'val': user_input}). No si concatenas strings.

¿ORM (SQLAlchemy declarative) o queries directas?

ORM cuando el modelo se usa en muchas partes (web app con N modelos). Queries directas para análisis ad-hoc. Pueden coexistir.

¿DuckDB sobre Parquet vs sobre pandas?

Parquet directo es más rápido (no carga a Python). Pandas cuando ya tienes el DataFrame en memoria. DuckDB es smart: optimiza ambos casos.

¿Cerrar conexión manualmente?

Usa context manager: with sqlite3.connect(...) as con: ... o con.close() en finally. Conexiones dejadas abiertas

consumen handles del OS.

Referencias

- Python sqlite3 docs
- SQLAlchemy tutorial
- DuckDB Python API
- OWASP — SQL injection prevention

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 046 — Clase 046 — NoSQL: MongoDB con pymongo

> *Parte: 0 — Prerrequisitos · Fuente: MongoDB docs · pymongo docs · MongoDB: The Definitive Guide (Bradshaw et al.) cap. 1.*

> *Duración estimada: 75 min.*

Objetivo

Que el alumno entienda el modelo NoSQL documento (collections de JSON-like), cuándo conviene sobre SQL, y use pymongo para CRUD básico + queries con operadores típicos. Sin pretender competir con un curso entero de MongoDB.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Diferenciar modelo relacional (tablas + filas) vs documento (collections + docs JSON).
2. Reconocer cuándo NoSQL aporta (schema flexible, datos jerárquicos, escala horizontal).
3. Conectar con pymongo, hacer insert/find/update/delete.
4. Filtrar con operadores: $gt, $lt, $in, $regex, $and, $or.
5. Hacer agregaciones con el pipeline ($match, $group, $sort).

Temas

#	Tema	Por qué importa
1	SQL vs NoSQL — cuándo cada uno	No "NoSQL es mejor" — distinto.
2	Modelo documento: collections + docs JSON	Schema flexible.
3	pymongo: connect, insert_one, find, update	CRUD básico.
4	Operadores de query: $gt/$lt/$in/$regex	Equivalentes a WHERE.
5	Aggregation pipeline	$match/$group/$sort — análogo a SQL.
6	Cuándo NO usar Mongo	Cuando relacional es claramente mejor.

Definiciones y características

NoSQL documento

: Familia de DBs que almacena documentos JSON-like (BSON en Mongo) en lugar de filas en tablas. Schema flexible — cada documento puede tener campos distintos.

Collection

: Equivalente a una tabla en SQL, pero sin schema fijo. Contiene documentos del mismo "tipo" lógico (productos, usuarios, eventos).

Documento (dict BSON)

: Unidad de almacenamiento. JSON con tipos extra (Date, ObjectId, Decimal128). Puede contener arrays y sub-documentos anidados.

Operador ($gt, $in, $elemMatch)

: Prefijo $ en las queries Mongo: {'precio': {'$gt': 100}} ≈ WHERE precio > 100. La query es un dict JSON, no string.

Aggregation pipeline

: Equivalente Mongo a CTEs encadenadas: lista de etapas ($match, $group, $sort, $project, $lookup) que procesan documentos secuencialmente.

$elemMatch

: Operador para filtrar por condiciones sobre elementos de un array dentro del documento. Útil con arrays de sub-docs (reviews dentro de producto).

Dataset / recursos

MongoDB local (Docker o Atlas free tier) — o usar mongomock para tests. Datos sintéticos: collection de productos.

Ejercicios

1. CRUD básico. Conecta a Mongo (o mongomock), inserta 5 productos, lee todos, actualiza uno, borra uno.

2. Find con operadores. Productos con precio > 100 y categoría en ['libros', 'musica'].

3. Update con $set y $inc. Incrementa stock de un producto en 10 unidades.

4. Aggregation pipeline. Promedio de precio por categoría con $group.

5. Documento jerárquico. Inserta un producto con array de reviews (sub-documentos). Consulta los que tienen alguna review con rating < 3 usando $elemMatch.

Homework verificable

Notebook con mongomock (no requiere Mongo real): (a) collection productos con 20 docs sintéticos; (b) 5 queries demostrando operadores; (c) aggregation pipeline con $match → $group → $sort; (d) reporte: 3 casos donde Mongo es mejor que SQL y 3 donde no.

Criterio de aceptación: Las queries funcionan; el reporte tiene casos justificados.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
pymongo.errors.ServerSelectionTimeoutError	No conecta al servidor. Fix: verifica que
Update sin $set reemplaza el documento ent	update_one(filter, {'precio': 100}) reempl
Aggregation con $group sin _id falla	$group requiere _id (la key de agrupación,

Query con {} devuelve todo (no None)	{} es "sin filtro" en Mongo. Si querías ma
Cuento con count_documents({}) y va lento	Sin filtro, recorre toda la collection. Fi

Preguntas frecuentes

¿SQL o NoSQL?

SQL para datos tabulares con relaciones, integridad referencial, reporting/BI. NoSQL documento para schema variable, datos jerárquicos naturales, escala write masiva. No es "mejor" — distinto.

¿find_one o find?

find_one devuelve dict (o None). find devuelve cursor iterable. Para 1 doc: find_one. Para muchos: list(coll.find(query)) o iterar el cursor.

¿pymongo vs Motor (async)?

pymongo síncrono, default. Motor asíncrono (asyncio) — para web apps con muchas concurrent connections.

¿Cómo tipo los documentos en Python?

Usa pydantic con BaseModel. Convierte dict ↔ tipo validado. Combinado con FastAPI, casi gratis (verás en MLOps).

¿Mongo para data science?

Como fuente sí (extraes datos con aggregation, los pasas a pandas). Para análisis ya no — pandas/DuckDB son mejores. Mongo brilla en operaciones (logs, eventos).

Referencias

- pymongo docs
- MongoDB query operators
- mongomock
- Bradshaw, MongoDB: The Definitive Guide 3e, cap. 1.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 047 — Clase 047 — APIs REST con requests

> *Parte: 0 — Prerrequisitos · Fuente: HTTP: The Definitive Guide caps. 1-2 · requests docs.*
>
> *Duración estimada: 90 min.*

Objetivo

Que el alumno consuma APIs REST públicas con requests: GET con parámetros, manejo de status codes, autenticación (header, bearer token), paginación, rate limiting con Retry, y carga eficiente con Session. Lo mínimo para no romper la API del proveedor ni tu pipeline.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Hacer GET/POST con requests, manejar params, headers, body JSON.
2. Verificar status code (200 vs 4xx vs 5xx) y usar raise_for_status().
3. Autenticarse con header Authorization: Bearer ... o API key en header/query.
4. Paginar correctamente cuando la API devuelve resultados en páginas.
5. Rate-limiting con urllib3.util.retry.Retry para reintentos exponenciales.
6. Reusar conexión con requests.Session para múltiples requests.

Temas

#	Tema	Por qué importa
1	Métodos HTTP: GET, POST, PUT, DELETE	Verbos REST.
2	Status codes: 2xx/3xx/4xx/5xx	Cómo reaccionar a cada uno.
3	Params, headers, body	Las 3 formas de mandar datos.
4	Autenticación: Bearer token, API key	Header Authorization.
5	Paginación: offset/limit, cursor, link hea	3 patrones comunes.
6	Rate limiting + retry exponencial	No tirar la API ajena.
7	requests.Session para reuso	Más rápido + cookies persistentes.

Versión profundizada — 2026

El tema moderno que vivía como complemento dentro de esta clase ahora tiene clase propia dedicada:

- Clase 046a — async / httpx / aiohttp para data scientists

Definiciones y características

REST (REpresentational State Transfer)

: Estilo arquitectónico para APIs web sobre HTTP. Recursos identificados por URLs, operaciones por verbos HTTP (GET=leer, POST=crear, PUT=update, DELETE=borrar).

requests

: Librería Python de facto para hacer HTTP. API simple: requests.get(url, params=..., headers=..., timeout=...). Soporta auth, cookies, sessions, retry.

Status code

: Número HTTP que indica resultado: 2xx éxito, 3xx redirect, 4xx error cliente (404 no encontrado, 401 no auth, 403 prohibido, 429 rate limited), 5xx error servidor.

Paginación

: API que devuelve resultados en bloques (no todo de golpe). Patrones: offset/limit (?page=2), cursor (?after=<id>), Link header (<url>; rel="next").

Rate limiting

: Política del servidor: máximo N requests/seg/usuario. Excederlo → 429. Respétalo con delays y retries exponenciales.

Session

: Reuso de conexión TCP/TLS entre requests. Mantiene cookies. ~10× más rápido para múltiples requests al mismo host vs requests.get repetido.

Bearer token

: Esquema de auth común: Authorization: Bearer <token> header. Token suele ser JWT o opaque string emitido por OAuth/login.

Dataset / recursos

API pública sin auth: https://api.coingecko.com (precios cripto). Sin API key necesaria.

Ejercicios

1. GET básico. requests.get('https://api.github.com'). Inspecciona status_code, headers, .json().

2. Con params. GitHub search: https://api.github.com/search/repositories?q=python+ml&sort=stars. Imprime top 5.

3. raise_for_status + try. Pega a una URL que devuelve 404 (/notfound) y maneja la excepción.

4. Paginación. GitHub events API. Itera 3 páginas con page=1,2,3.

5. Session + Retry. Configura una Session con HTTPAdapter + Retry (3 intentos, backoff 1s). Verifica que reintenta en 5xx simulado.

Homework verificable

Notebook que: (a) consulta una API pública (CoinGecko, GitHub, JSONPlaceholder) con GET; (b) maneja status codes con try/except; (c) pagina 3+ páginas; (d) configura Session con Retry exponencial; (e) reporta cuánto se tardó vs un loop sin Session.

Criterio de aceptación: Maneja al menos un error sin crash. Pagination devuelve datos esperados.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
requests.exceptions.ConnectTimeout o Timeo	API lenta o sin red. Fix: siempre pasa tim
API responde 200 pero .json() lanza error	Body no es JSON (HTML de error, vacío). Fi
Hardcodeé el token en el código y subí a G	Catástrofe de seguridad — el token es públ
Mi script tira la API ajena (HTTP 429)	Sin rate limiting. Fix: time.sleep() entre
HTTPError no se lanza con status 4xx	requests NO lanza por default. Fix: r.rais
RuntimeError: asyncio.run() cannot be call	El notebook ya tiene un event loop corrien

Preguntas frecuentes

¿requests o httpx?

requests sigue siendo la default (estable, ubícua). httpx drop-in con async support (async with httpx.AsyncClient() as client:). Para async/HTTP2, httpx; para todo lo demás, requests.

¿Cuándo Session?

Más de 2-3 requests al mismo host. La primera request hace handshake TCP/TLS (~100ms); Session lo reusa. Para single request, no aporta.

¿json= o data= en POST?

json=dict: serializa a JSON y setea Content-Type: application/json. data=dict: form-encoded (application/x-www-form-urlencoded). Para APIs REST modernas, casi siempre json=.

¿Cómo paginar genéricamente?

Loop hasta que la API diga "no más": while True: r = requests.get(url, params=...); items.extend(r.json()['data']); if not r.json().get('next'): break.

¿Auth OAuth desde Python?

Para casos simples (Bearer fijo): pasa el header. Para OAuth flow completo: authlib, requests-oauthlib. Para producción: librería oficial del proveedor (google-auth, pyOpenSSL, etc.).

¿Cuándo paso de requests a httpx async?

Regla práctica: cuando tenés >20 requests simultáneos al mismo proveedor y el cuello de botella es latencia de red (no CPU). Si tu script está 90% del tiempo esperando respuestas HTTP, async te da un speedup de 10-100×. Si en cambio estás parseando JSONs gigantes o haciendo cálculo pesado, async no ayuda — ahí lo tuyo es multiprocessing. Pocas requests (<10): seguí con requests, no vale la complejidad.

Referencias

- requests docs
- urllib3 Retry
- GitHub REST API
- HTTP status codes (MDN)
- httpx docs
- asyncio + Jupyter (autoawait)

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 048 — Clase 048 — Web scraping con BeautifulSoup

> *Parte: 0 — Prerrequisitos · Fuente: Web Scraping with Python (Mitchell, 2ª ed.) caps. 1-3 · BeautifulSoup docs.*

> *Duración estimada: 75 min.*

Objetivo

Que el alumno extraiga datos de páginas HTML cuando no hay API disponible, usando requests + BeautifulSoup. Y entienda los límites éticos y legales: robots.txt, rate limiting humano, ToS, datos personales, copyright. Lo último que debe hacer al scrapear es tirar abajo el sitio o meterse en problemas.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Parsear HTML con BeautifulSoup(html, 'html.parser').
2. Encontrar elementos con find, find_all, select (CSS selectors).
3. Extraer texto y atributos (.text, ['href']).
4. Respetar robots.txt y rate limit (delay entre requests).

5. Identificar cuándo scraping es buena idea vs cuándo buscar otra fuente (API, dataset público).

Temas

#	Tema	Por qué importa
1	HTTP → HTML → parser tree	Cómo funciona scraping.
2	BeautifulSoup: find vs select	Selectores CSS son más potentes.
3	Extracción de texto y atributos	.text, .get_text(strip=True), ['href'].
4	Páginas dinámicas (JS) — requests no las r	Para eso: Playwright/Selenium.
5	robots.txt — qué dice y por qué respetar	User-agent, Disallow, Crawl-delay.
6	Ética: ToS, rate limiting, datos personale	Lo que sí, lo que no.

Definiciones y características

Web scraping

: Extracción programática de datos de páginas HTML cuando no hay API. Pipeline típico: descargar HTML con requests, parsear con BeautifulSoup, extraer con selectores.

BeautifulSoup

: Parser HTML tolerante a errores. soup = BeautifulSoup(html, 'html.parser'). Permite navegar el árbol y buscar por tag, atributo o selector CSS.

CSS selector

: Sintaxis para localizar elementos: '.class', '#id', 'tag', 'parent > child', 'tag[attr=val]'. Usados con soup.select(...). Más expresivos que find_all.

DOM (Document Object Model)

: Representación en árbol del HTML. Cada tag es un nodo; tiene padre, hermanos, hijos. BeautifulSoup navega este árbol con .parent, .next_sibling, .find_all, etc.

robots.txt

: Archivo en raíz del dominio (/robots.txt) que declara qué paths pueden crawlear los bots. Estándar de facto; respetarlo es legalmente importante (variable por jurisdicción) y éticamente siempre.

JS rendering

: Páginas SPA (React, Vue) cargan contenido vía JavaScript tras el HTML inicial. requests solo trae el HTML inicial — JS no se ejecuta. Solución: Playwright o Selenium (navegador headless).

Dataset / recursos

Página HTML simple servida desde un string en el notebook (sin tocar internet). Ejercicios opcionales con https://quotes.toscrape.com (sitio diseñado para practicar).

Ejercicios

1. Parsea HTML local. Crea un HTML con 3 productos ( ). Extrae nombres y precios con find_all.

2. Selectores CSS. Lo mismo con soup.select('.product .price').

3. Tabla a DataFrame. pd.read_html(url) para una tabla HTML — bonus: requests + BeautifulSoup para tablas custom.

4. Scrape ético. Scrapea quotes.toscrape.com (público, diseñado para esto). Respeta Crawl-delay. 3 páginas

con time.sleep(1) entre cada una.

5. Inspeccionar robots.txt. Lee https://quotes.toscrape.com/robots.txt con requests. Identifica qué paths están Disallow.

Homework verificable

Notebook: (a) HTML local con 5 productos, extrae nombre/precio/url; (b) scrape quotes.toscrape.com (3 páginas, con delay); (c) consulta robots.txt y razona; (d) listado de 3 escenarios cuando scrapear es buena idea y 3 cuando no.

Criterio de aceptación: Scraping respeta delays. Análisis de robots.txt correcto.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
soup.find('div') devuelve None aunque hay	Buscas un atributo específico que no match
Scrapeo y recibo HTML distinto al que veo	El sitio renderiza con JavaScript. request
HTTP 403 Forbidden	El sitio detecta tu bot (User-Agent vacío
Site funciona en navegador pero scraper de	Anti-bot agresivo (Cloudflare, reCAPTCHA).
Encoding raro (acentos Ã¡)	Pandas/requests dedujo encoding mal. Fix:

Preguntas frecuentes

¿Scraping es legal?

Depende: jurisdicción, ToS del sitio, naturaleza del dato. Datos personales = casi siempre regulado (GDPR/LGPD). Datos públicos sin ToS prohibitivo = generalmente OK. Consulta abogado para casos serios.

¿find_all o select?

select (CSS selectors) — más potente, sintaxis estándar web, más legible. find_all para casos simples o cuando integras con código heredado.

¿Cómo descargo imágenes?

r = requests.get(url_imagen); open('img.jpg', 'wb').write(r.content). Para muchas, usa Session + thread pool.

¿Scrapy vs BeautifulSoup?

BeautifulSoup: librería de parsing. Una página, una request. Scrapy: framework completo (crawler, pipelines, throttling). Para proyectos serios (miles de páginas), Scrapy.

¿Y si el sitio me bloquea?

Respeta. Aumentar agresividad (proxies, rotating User-Agents) puede ser ilegal en algunas jurisdicciones (CFAA en USA). Considera: ¿realmente vale la pena? ¿hay otra fuente?

Referencias

- Mitchell, Web Scraping with Python 2e, caps. 1-3.
- BeautifulSoup docs
- quotes.toscrape.com (sitio para practicar)
- Google — robots.txt

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 049 — Clase 049 — async / httpx / aiohttp para data scientists

> *Parte: 0 — Prerrequisitos · Fuente: docs asyncio + httpx + Beazley Python Concurrency. Duración estimada: 80 min.*

Objetivo

Aprender asyncio y httpx —el HTTP client moderno con soporte sync + async + HTTP/2— para hacer scraping y consumo de APIs en paralelo sin bloquear. Pasar de "1 request por segundo" con requests a "100+ concurrentes" con httpx.AsyncClient. Comparar con aiohttp (alternativa popular) y concurrent.futures (parallelism con threads).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Definir async def y await; ejecutar con asyncio.run.
- Usar httpx.AsyncClient para fetches concurrentes con asyncio.gather.
- Limitar concurrencia con asyncio.Semaphore para no DOS-ear a la API.
- Implementar rate limiting + retries con backoff exponencial.
- Decidir entre asyncio, threading, multiprocessing según I/O-bound vs CPU-bound.

Temas

- Event loop, coroutines, await.
- httpx: API unificada sync/async, HTTP/2, timeouts.
- asyncio.gather, asyncio.as_completed.
- Semaphore para limitar concurrencia.
- Backoff exponencial con tenacity o backoff lib.
- aiohttp como alternativa (más antigua, más utilities).
- Cuándo NO async: tareas CPU-bound (usar multiprocessing).

Definiciones y características

- Coroutine: función async def, devuelve un objeto coroutine que el event loop ejecuta.
- await: cede control al event loop hasta que la operación termina.
- Event loop: scheduler que ejecuta coroutines cooperativamente.
- asyncio.gather(*coros): ejecuta varias coroutines concurrentemente, espera todas.
- asyncio.as_completed(coros): itera resultados a medida que llegan.
- asyncio.Semaphore(n): limita a n coroutines concurrentes.
- httpx: HTTP client moderno (Tom Christie, dev de Django REST Framework).
- aiohttp: client + server async. Más viejo, más maduro, sin sync API.

Dataset / recursos

- Una API pública lenta: https://httpbin.org/delay/2 (espera 2 seg).
- Lista de 100 URLs para fetcher.
- Librerías: httpx, aiohttp, opcional tenacity.

Ejercicios

1. Sync baseline: fetcher de 50 URLs con requests.get en loop. Medir tiempo.
2. Async con httpx: async with httpx.AsyncClient() as c: results = await asyncio.gather(*[c.get(url) for url in urls]). Comparar tiempo (debería ser ~20-50× más rápido).
3. Semaphore: limitar a 10 concurrent. Útil para no ser bloqueado por rate limits.
4. Retry exponencial: usar tenacity.retry(stop=stop_after_attempt(5), wait=wait_exponential(min=1, max=30)) sobre un fetcher.
5. httpx vs aiohttp: hacer el mismo benchmark con ambos. Similar performance; httpx tiene mejor DX.

Homework verificable

Scraper concurrente de 200 URLs (una página de Wikipedia o API pública):

1. Implementar con httpx.AsyncClient + asyncio.gather.
2. Limitar a 20 concurrent con Semaphore.
3. Retry con backoff sobre errores 5xx.
4. Reportar tiempo total y % de éxito.

Criterio de aceptación: tiempo total ≤ 1/10 de la versión sync; success rate ≥ 95 %.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
RuntimeError: This event loop is already r	Jupyter ya tiene un loop. Fix: nest_asynci
Olvido await → coroutine never awaited war	Fix: await coro() o asyncio.run(coro()).
Mezclar sync requests adentro de coroutine	Bloquea el event loop. Fix: usar httpx.Asy
Sin limit → API te banea	Demasiada concurrencia. Fix: Semaphore + d
AsyncClient no cerrado	Resource leak. Fix: async with httpx.Async

Preguntas frecuentes

async o threading?

Async para I/O-bound (HTTP, DB) — más eficiente, menos overhead. Threading para I/O-bound legacy code. Multiprocessing para CPU-bound (cálculo numérico, ML training).

httpx o aiohttp?

httpx es default moderno: API unificada sync/async, mejor DX, HTTP/2. aiohttp tiene más utilities (sessions, file uploads) pero solo async.

async con pandas / numpy?

No tiene sentido — son CPU-bound. Async brilla cuando esperás de I/O externo.

Test de async code?

pytest-asyncio con @pytest.mark.asyncio.

async en Django / Flask?

Django 4+ soporta async views. Flask 2+ también. FastAPI es async-native (default moderno).

Referencias

- httpx docs

- asyncio docs
- aiohttp docs
- Beazley, D. Python Concurrency from the Ground Up (PyCon 2015).
- tenacity para retries.

Siguiente parte

Clase 050 — Panorama del ML: tipos, batch vs online, instance vs model-based

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Cierre de la parte

Fin del bundle consolidado de Parte 0 — Prerrequisitos: Python + NumPy + pandas + visualización + SQL + APIs · 49 clases.