

# Herramientas para Consultoría con IA

Git, VSCode, Python y Jupyter – HE2

Santiago Neira Hernández / Catalina Bernal

2025

# Contenido

¿Por qué estas herramientas?

Git: Control de versiones

VSCode: Tu entorno de desarrollo

Python y ambientes virtuales

Jupyter Notebooks

Flujo de trabajo integrado

Recursos y próximos pasos

# El desafío de la consultoría moderna

## Contexto

En consultoría económica con IA, necesitamos:

- ▶ **Reproducibilidad:** que otros puedan replicar nuestro análisis
- ▶ **Colaboración:** trabajar en equipo sin conflictos
- ▶ **Trazabilidad:** saber qué cambió, cuándo y por qué
- ▶ **Eficiencia:** automatizar y reutilizar código

# El desafío de la consultoría moderna

## Contexto

En consultoría económica con IA, necesitamos:

- ▶ **Reproducibilidad:** que otros puedan replicar nuestro análisis
- ▶ **Colaboración:** trabajar en equipo sin conflictos
- ▶ **Trazabilidad:** saber qué cambió, cuándo y por qué
- ▶ **Eficiencia:** automatizar y reutilizar código

## Solución

Un flujo de trabajo profesional con herramientas estándar de la industria.

# Las 4 herramientas esenciales

1. **Git**: Control de versiones (el "track changes" del código)
2. **VSCode**: Editor de código moderno y extensible
3. **Python**: Lenguaje de programación versátil
4. **Jupyter**: Notebooks interactivos para análisis exploratorio

# Las 4 herramientas esenciales

1. **Git**: Control de versiones (el "track changes" del código)
2. **VSCode**: Editor de código moderno y extensible
3. **Python**: Lenguaje de programación versátil
4. **Jupyter**: Notebooks interactivos para análisis exploratorio

## Objetivo de hoy

Entender el **flujo completo**: desde clonar un repositorio hasta ejecutar análisis en notebooks.

# ¿Qué es Git?

## Definición

Git es un sistema de **control de versiones distribuido** que registra cambios en archivos a lo largo del tiempo.

# ¿Qué es Git?

## Definición

Git es un sistema de **control de versiones distribuido** que registra cambios en archivos a lo largo del tiempo.

### Sin Git:

- ▶ `analisis_final.py`
- ▶ `analisis_final_v2.py`
- ▶ `analisis_final_AHORA_SI_FINAL.py`
- ▶ `analisis_final_revision_profesor.py`

# ¿Qué es Git?

## Definición

Git es un sistema de **control de versiones distribuido** que registra cambios en archivos a lo largo del tiempo.

### Sin Git:

- ▶ `analisis_final.py`
- ▶ `analisis_final_v2.py`
- ▶ `analisis_final_AHORA_SI_FINAL.py`
- ▶ `analisis_final_revision_profesor.py`

### Con Git:

- ▶ Un solo archivo: `analisis.py`
- ▶ Historial completo de cambios
- ▶ Mensajes descriptivos de cada modificación
- ▶ Posibilidad de volver atrás cuando sea necesario

# Conceptos clave de Git

**Repositorio (repo)** Carpeta de proyecto con historial de cambios

**Commit** "Fotografía" del proyecto en un momento dado

**Branch** Línea independiente de desarrollo

**Remote** Versión del repo en la nube (ej: GitHub)

**Clone** Copiar un repositorio remoto a tu computadora

**Pull** Traer cambios del remote a tu repo local

**Push** Enviar tus commits al remote

# Comandos básicos de Git

```
# Clonar un repositorio existente  
git clone https://github.com/usuario/proyecto.  
git
```

```
# Ver estado de cambios  
git status
```

```
# Agregar archivos al staging area  
git add archivo.py  
git add . # agregar todos los cambios
```

```
# Hacer commit (guardar cambios)  
git commit -m "Descripcion clara del cambio"
```

```
# Enviar cambios al repositorio remoto  
git push
```

```
# Traer cambios del repositorio remoto  
git pull
```

# Flujo típico con Git

1. **Clone:** Descargar el repositorio del proyecto
2. **Editar:** Hacer cambios en archivos localmente
3. **Add:** Preparar archivos modificados
4. **Commit:** Guardar cambios con mensaje descriptivo
5. **Pull:** Asegurarse de tener la última versión
6. **Push:** Subir cambios al repositorio compartido

## Buena práctica

Hacer commits frecuentes y con mensajes claros: *“Agregué análisis de correlación”* en vez de *“cambios varios”*.

# GitHub: Git en la nube

**GitHub** es una plataforma que:

- ▶ Aloja repositorios Git en la nube
- ▶ Facilita colaboración en equipo
- ▶ Permite revisión de código (pull requests)
- ▶ Ofrece gestión de proyectos (issues, proyectos)

Para este curso

- ▶ Proyectos en repositorios de equipo
- ▶ Documentación en README.md
- ▶ Código y análisis versionados

**Alternativas:** GitLab, Bitbucket

Documentación GitHub (español) — GitHub Student Pack (beneficios)

# ¿Qué es Visual Studio Code?

## Definición

VSCoDe es un **editor de código** gratuito, ligero y altamente extensible desarrollado por Microsoft.

# ¿Qué es Visual Studio Code?

## Definición

VSCoide es un **editor de código** gratuito, ligero y altamente extensible desarrollado por Microsoft.

## ¿Por qué VSCoide?

- ▶ Soporte nativo para Git
- ▶ Extensiones para Python, Jupyter, R, LaTeX...
- ▶ Terminal integrada
- ▶ IntelliSense (autocompletado inteligente)
- ▶ Depuración visual
- ▶ Multiplataforma (Windows, Mac, Linux)

Descargar VSCoide — Documentación oficial

# Configuración inicial de VSCode

## Extensiones esenciales para este curso:

1. **Python** (Microsoft): Soporte completo para Python
2. **Jupyter** (Microsoft): Ejecutar notebooks en VSCode
3. **GitLens**: Visualización avanzada de Git

# Configuración inicial de VSCode

## Extensiones esenciales para este curso:

1. **Python** (Microsoft): Soporte completo para Python
2. **Jupyter** (Microsoft): Ejecutar notebooks en VSCode
3. **GitLens**: Visualización avanzada de Git

## Cómo instalar extensiones:

- ▶ Icono de extensiones (Ctrl+Shift+X / Cmd+Shift+X)
- ▶ Buscar por nombre
- ▶ Clic en “Install”

# Trabajando con Git en VSCode

## VSCode integra Git visualmente:

1. **Source Control panel:** Ver cambios pendientes
2. **Iconos visuales:** M (modificado), U (untracked), D (eliminado)
3. **Stage changes:** Click en + junto a archivos
4. **Commit:** Escribir mensaje y hacer commit
5. **Sync:** Push/pull con un click

## Ventaja

No necesitas memorizar comandos de Git inicialmente; la interfaz visual te guía.

# Python: El lenguaje

## ¿Por qué Python para consultoría con IA?

- ▶ Ecosistema rico: pandas, numpy, scikit-learn, tensorflow
- ▶ Sintaxis clara y legible
- ▶ Comunidad activa y abundante documentación
- ▶ Integración con herramientas de IA/ML
- ▶ Usado en industria y academia

## Instalación:

- ▶ Opción 1: Python oficial
- ▶ Opción 2: Anaconda (incluye muchas librerías)

# El problema de las dependencias

## Escenario problemático:

- ▶ Proyecto A requiere pandas 1.3.0
- ▶ Proyecto B requiere pandas 2.0.0
- ▶ Instalas ambas versiones → **conflicto**

# El problema de las dependencias

## Escenario problemático:

- ▶ Proyecto A requiere pandas 1.3.0
- ▶ Proyecto B requiere pandas 2.0.0
- ▶ Instalas ambas versiones → **conflicto**

## Solución: Ambientes virtuales

Cada proyecto tiene su propio conjunto aislado de librerías.

# El problema de las dependencias

## Escenario problemático:

- ▶ Proyecto A requiere pandas 1.3.0
- ▶ Proyecto B requiere pandas 2.0.0
- ▶ Instalas ambas versiones → **conflicto**

## Solución: Ambientes virtuales

Cada proyecto tiene su propio conjunto aislado de librerías.

## Herramientas:

- ▶ **venv**: Incluido con Python (estándar)
- ▶ **conda**: Parte de Anaconda
- ▶ **poetry**: Gestor moderno de dependencias

# Creando un ambiente virtual con venv

## En la terminal de VSCode:

```
# Crear ambiente virtual
python -m venv venv

# Activar (Windows)
venv\Scripts\activate

# Activar (Mac/Linux)
source venv/bin/activate

# Instalar paquetes
pip install pandas numpy jupyter

# Guardar dependencias
pip freeze > requirements.txt

# Desactivar cuando termines
deactivate
```

# Buenas prácticas con ambientes virtuales

1. **Un ambiente por proyecto:** Aislar dependencias
2. **Archivo requirements.txt:** Documentar librerías necesarias
3. **No versionar el ambiente:** Agregar venv/ a .gitignore
4. **Activar antes de trabajar:** Asegurar el ambiente correcto
5. **Actualizar requirements:** Cuando agregues librerías nuevas

## Para reproducibilidad

Otro colaborador puede recrear tu ambiente exacto con:

```
pip install -r requirements.txt
```

# VSCode y Python: Configuración

## Seleccionar intérprete de Python:

1. Ctrl+Shift+P (Cmd+Shift+P en Mac)
2. Buscar: "Python: Select Interpreter"
3. Elegir el Python de tu ambiente virtual (venv)

# VSCode y Python: Configuración

## Seleccionar intérprete de Python:

1. Ctrl+Shift+P (Cmd+Shift+P en Mac)
2. Buscar: "Python: Select Interpreter"
3. Elegir el Python de tu ambiente virtual (venv)

## Ventajas:

- ▶ VSCode usa automáticamente ese ambiente
- ▶ Linting y autocompletado con las librerías correctas
- ▶ Terminal se activa con el ambiente automáticamente

# ¿Qué son los Jupyter Notebooks?

## Definición

Documentos interactivos que combinan:

- ▶ **Código** ejecutable (Python, R, Julia...)
- ▶ **Texto** con formato (Markdown)
- ▶ **Visualizaciones** (gráficos, tablas)
- ▶ **Ecuaciones** (LaTeX)

# ¿Qué son los Jupyter Notebooks?

## Definición

Documentos interactivos que combinan:

- ▶ **Código** ejecutable (Python, R, Julia...)
- ▶ **Texto** con formato (Markdown)
- ▶ **Visualizaciones** (gráficos, tablas)
- ▶ **Ecuaciones** (LaTeX)

## Ideal para:

- ▶ Análisis exploratorio de datos
- ▶ Reportes ejecutivos con código
- ▶ Enseñanza y documentación
- ▶ Prototipado rápido

# Estructura de un notebook

## Celdas (cells):

**Code cells** Código Python ejecutable

**Markdown cells** Texto con formato, títulos, listas

**Output** Resultados de ejecución (texto, gráficos, tablas)

# Estructura de un notebook

## Celdas (cells):

**Code cells** Código Python ejecutable

**Markdown cells** Texto con formato, títulos, listas

**Output** Resultados de ejecución (texto, gráficos, tablas)

## Ejecución:

- ▶ Las celdas se ejecutan en **orden** (pero puedes cambiar el orden)
- ▶ Cada celda comparte el mismo **estado** (variables)
- ▶ Shift+Enter: ejecutar celda y avanzar
- ▶ Ctrl+Enter: ejecutar celda sin avanzar

# Jupyter en VSCode vs Jupyter Lab

## VSCode

- ▶ Notebooks en tu editor habitual
- ▶ Sin necesidad de abrir navegador
- ▶ Integrado con Git
- ▶ Mismo ambiente de trabajo

## Jupyter Lab

- ▶ Interfaz web dedicada
- ▶ Más características avanzadas
- ▶ Múltiples notebooks simultáneos
- ▶ Terminal y editor incluidos

# Jupyter en VSCode vs Jupyter Lab

## VSCode

- ▶ Notebooks en tu editor habitual
- ▶ Sin necesidad de abrir navegador
- ▶ Integrado con Git
- ▶ Mismo ambiente de trabajo

## Jupyter Lab

- ▶ Interfaz web dedicada
- ▶ Más características avanzadas
- ▶ Múltiples notebooks simultáneos
- ▶ Terminal y editor incluidos

## Recomendación para el curso

Usar **VSCode** para integración con Git y flujo unificado.

Instalar Jupyter: `pip install jupyter notebook jupyterlab`

# Ejecutando notebooks en VSCode

## Pasos:

1. Abrir archivo `.ipynb` en VSCode
2. La extensión Jupyter se activa automáticamente
3. Seleccionar kernel (intérprete Python del ambiente)
4. Ejecutar celdas con botones o Shift+Enter
5. Ver outputs inline

# Ejecutando notebooks en VSCode

## Pasos:

1. Abrir archivo `.ipynb` en VSCode
2. La extensión Jupyter se activa automáticamente
3. Seleccionar kernel (intérprete Python del ambiente)
4. Ejecutar celdas con botones o Shift+Enter
5. Ver outputs inline

## Ventajas en VSCode:

- ▶ Variable explorer (ver variables en memoria)
- ▶ Debugging visual
- ▶ Exportar a PDF, HTML, Python script

# Buenas prácticas con notebooks

1. **Título y descripción:** Qué hace el notebook
2. **Imports al inicio:** Todas las librerías arriba
3. **Celdas ordenadas:** Lógica de arriba hacia abajo
4. **Comentarios claros:** Explicar pasos importantes
5. **Restart & Run All:** Verificar reproducibilidad
6. **Limpiar outputs:** Antes de commit (opcional)
7. **No hardcodear rutas:** Usar rutas relativas

## Regla de oro

Otro miembro del equipo debe poder ejecutar tu notebook de principio a fin sin errores.

# Flujo completo: Git + VSCode + Python + Jupyter

## Workflow típico de un proyecto:

1. **Clonar repo:** `git clone <url>`
2. **Abrir en VSCode:** `code .` en la carpeta
3. **Crear ambiente:** `python -m venv venv`
4. **Activar ambiente:** `source venv/bin/activate`
5. **Instalar deps:** `pip install -r requirements.txt`
6. **Seleccionar intérprete:** VSCode → Select Python
7. **Trabajar:** Editar notebooks, scripts
8. **Commit cambios:** Git panel en VSCode
9. **Push:** Compartir con equipo

# Estructura de proyecto sugerida

```
proyecto-he2/  
|-- data/  
|   |-- raw/           # Datos originales  
|   |-- processed/     # Datos limpios  
|-- notebooks/  
|   |-- 01_exploracion.ipynb  
|   |-- 02_analisis.ipynb  
|-- src/  
|   |-- utils.py       # Funciones reutilizables  
|-- outputs/  
|   |-- figuras/  
|   |-- reportes/  
|-- venv/              # Ambiente virtual (no  
    versionar)  
|-- requirements.txt  
|-- README.md  
|-- .gitignore
```

# El archivo .gitignore

¿Qué es? Archivo que indica a Git qué **no** versionar.

## Contenido típico para proyectos Python:

```
# Ambientes virtuales
venv/
env/

.ipynb_checkpoints/
*-checkpoint.ipynb

# Python
__pycache__/*
*.pyc
.pytest_cache/

# Sistema
.DS_Store
Thumbs.db
```

# Colaboración en equipo

## Estrategia para trabajar en grupo:

1. **Comunicación:** Quién trabaja en qué
2. **Pull antes de trabajar:** Obtener última versión
3. **Commits frecuentes:** Cambios pequeños y claros
4. **Resolver conflictos:** Si dos editan lo mismo
5. **Revisar código:** Code review en pull requests
6. **Documentación:** README actualizado

## Importante

La **comunicación** es tan importante como el código.

# Recursos de aprendizaje

## Git:

- ▶ Learn Git Branching (interactivo, español)
- ▶ Tutoriales Atlassian (español)

## VSCode:

- ▶ Tips and Tricks oficiales
- ▶ Curso VSCode (español, YouTube)

## Python:

- ▶ Tutorial oficial Python (español)
- ▶ Kaggle Learn Python

## Jupyter:

- ▶ Documentación Jupyter Notebook
- ▶ Tutorial completo Jupyter

# Checklist para empezar

## Asegúrate de tener instalado:

- ▶ Git ([git-scm.com/downloads](https://git-scm.com/downloads))
- ▶ VSCode ([code.visualstudio.com](https://code.visualstudio.com))
- ▶ Python 3.8+ ([python.org/downloads](https://python.org/downloads))
- ▶ Extensiones VSCode: Python, Jupyter, GitLens
- ▶ Cuenta GitHub ([github.com/signup](https://github.com/signup))

## Verificar instalación:

- ▶ Terminal: `git --version`
- ▶ Terminal: `python --version`
- ▶ Terminal: `code --version`

# Práctica sugerida, lo que haremos hoy

## Ejercicio para consolidar (individual):

1. Crear cuenta GitHub si no la tienes
2. Crear un repositorio nuevo: "practica-he2"
3. Clonarlo a tu computadora
4. Crear ambiente virtual
5. Crear un notebook simple (hola.ipynb)
6. Agregar celda Markdown con tu nombre
7. Agregar celda código: `print("Hola HE2")`
8. Ejecutar el notebook
9. Commit y push al repositorio
10. Ver tus cambios en GitHub

# Siguientes pasos en el curso

## **Con estas herramientas configuradas, podrán:**

- ▶ Trabajar en proyectos de consultoría en equipo
- ▶ Versionar análisis y código de forma profesional
- ▶ Crear reportes reproducibles con notebooks
- ▶ Compartir y colaborar eficientemente
- ▶ Integrar herramientas de IA de forma responsable

## Próxima semana

¡Empezaremos con manipulación de datos!

# ¿Preguntas?

*“El código que escribes hoy debe poder ser entendido  
y ejecutado por tu equipo mañana.”*