Manual de Ingeniería de IA I

Ciencia de Datos, Bash, Git y MLOps Inicial

AgroFuture AI Training

Enero 2026

Resumen

El 80 % del éxito de un proyecto de Inteligencia Artificial depende de la ingeniería de datos, automatización y reproducibilidad. Este manual forma al estudiante en las herramientas fundamentales que todo científico de datos profesional domina: terminal Linux, procesamiento de datos, control de versiones y gestión de entornos.

Índice

Prefacio: Tu Laboratorio Digital

Imagina una finca equipada con sensores que generan miles de registros diarios en archivos CSV comprimidos. Abrir archivos manualmente no es una opción sostenible cuando los datos crecen. [web:1][web:11] La ciencia de datos comienza cuando automatizas, validas y documentas cada paso del flujo de trabajo.

Principio Fundamental

Un modelo que no puede reproducirse no es ciencia: es magia.

1. ¿Qué Hace Realmente un Científico de Datos?

Un científico de datos profesional:

- Diseña flujos de datos reproducibles desde la captura hasta el reporte.
- Automatiza procesos repetitivos para evitar errores humanos.
- Detecta y corrige errores en los datos antes de entrenar modelos.
- Documenta decisiones técnicas para que otros puedan auditar y mejorar el trabajo.
- Piensa en escalabilidad, mantenimiento y colaboración con otros perfiles técnicos.

Distribución Real del Trabajo

Por cada hora entrenando modelos:

- 3 horas en limpieza y validación de datos.
- 2 horas en automatización y scripting.
- 1 hora en documentación y versionado.

Objetivo de este manual

Al finalizar este manual podrás:

- Navegar y manipular archivos desde la terminal Linux.
- Construir pequeños pipelines de procesamiento de datos con Bash.
- Usar Git como bitácora científica de tus experimentos.
- Gestionar entornos de Python para proyectos de ciencia de datos.

2. Fase 1: Supervivencia en la Terminal

La terminal es un lenguaje de programación declarativo para hablar con el sistema operativo. Todo flujo de datos sigue la lógica:

$$\text{entrada} \rightarrow \text{proceso} \rightarrow \text{salida}$$

En esta fase aprenderás a orientarte en el sistema de archivos y manipular archivos como un científico de datos.

2.1. Navegación Básica

Objetivo

Aprender a saber dónde estás, qué archivos hay y cómo moverte entre carpetas de un proyecto de datos. [web:2][web:24]

Comandos clave:

- `pwd`: muestra la ruta completa de la carpeta actual.
- `ls -lh`: lista archivos con tamaños legibles.
- `ls *.csv`: lista solo archivos CSV en la carpeta actual.
- `cd carpeta`: cambia a una subcarpeta.
- `cd ..`: sube un nivel en la jerarquía.
- `cd ~`: va a la carpeta personal del usuario.

```
# Ver en qué carpeta estás
pwd
/home/estudiante

# Listar el contenido de la carpeta actual
ls -lh
drwxr-xr-x 2 estudiante estudiante 4.0K ene 10 proyectos
-rw-r--r-- 1 estudiante estudiante 1.2M ene 10 sensores.csv

# Entrar a la carpeta del proyecto
cd proyectos
pwd
/home/estudiante/proyectos

# Listar solo archivos CSV
ls *.csv
sensores.csv resumen_mensual.csv
```

Ejercicio guiado

1. Crea una carpeta `proyecto_ia` dentro de tu `~`.
2. Dentro de `proyecto_ia`, crea las carpetas `data`, `scripts` y `reports`.
3. Usa `pwd` y `ls -lh` en cada paso para verificar que estás en la carpeta correcta.

2.2. Gestión de Archivos

Objetivo

Crear, copiar, mover y eliminar archivos y carpetas para organizar un proyecto de datos. [web:24]

Comandos frecuentes:

- `mkdir -p ruta`: crea una carpeta (y las intermedias si no existen).
- `touch archivo`: crea un archivo vacío o actualiza su fecha.
- `cp origen destino`: copia archivos.
- `mv origen destino`: mueve o renombra archivos.
- `rm archivo`: elimina archivos.

```
# Crear estructura de carpetas del proyecto
mkdir -p proyecto_ia/data/raw
mkdir -p proyecto_ia/data/processed
mkdir -p proyecto_ia/scripts
mkdir -p proyecto_ia/reports

# Crear un archivo de notas vacío
cd proyecto_ia
touch notas_proyecto.md

# Copiar un CSV de sensores al directorio raw
cp ~/sensores.csv data/raw/sensores.csv

# Renombrar el archivo de notas
mv notas_proyecto.md README.md

# Eliminar un archivo (con confirmación si usas alias de seguridad)
rm data/raw/archivo_innecesario.csv
```

Advertencia

`rm` elimina permanentemente. Usa siempre `rm -i` para que pregunte antes de borrar cada archivo. Más adelante definiremos alias de seguridad. [web:24]

Ejercicio

Diseña la estructura de carpetas para un proyecto de monitoreo de humedad de suelo y crea:

- Una carpeta `figures` para gráficos.
- Una carpeta `logs` para registros de ejecución.
- Un archivo `TODO.md` en la raíz del proyecto.

2.3. Comandos Clave de Científico de Datos

Objetivo

Medir uso de disco, buscar archivos de datos y operar sobre columnas de archivos CSV desde la terminal. [web:1][web:3][web:11]

Comandos útiles:

- `du -sh *`: muestra el tamaño de cada archivo/carpeta.
- `df -h`: muestra el espacio disponible en discos.
- `find . -name "*.csv"`: busca archivos con extensión `.csv`.
- `cut -d',' -f1`: extrae columnas de un CSV.
- `sort | uniq`: ordena y elimina duplicados.

```
# Ver qué carpeta ocupa más espacio dentro de data
cd proyecto_ia/data
du -sh *

# Ver espacio disponible en el disco
df -h

# Buscar todos los archivos CSV en el proyecto
cd ..
find . -name "*.csv"

# Extraer la primera columna (por ejemplo, id_sensor) de un CSV
cd data/raw
cut -d',' -f1 sensores.csv | head

# Obtener lista única de sensores
cut -d',' -f1 sensores.csv | sort | uniq | head
```

Ejercicio aplicado

Supón que `sensores.csv` tiene columnas `id_sensor,fecha,temperatura,humedad`.

1. Obtén la lista única de `id_sensor` ordenada alfabéticamente.
2. Cuenta cuántos registros totales tiene el archivo.
3. Calcula cuántos registros hay por cada `id_sensor` usando `sort | uniq -c`.

3. Fase 2: Procesamiento y Automatización

En esta fase aprenderás a inspeccionar datos reales, detectar errores frecuentes y construir pequeños pipelines de procesamiento usando la terminal. [web:3][web:11]

3.1. Inspección de Datos

Objetivo

Inspeccionar rápidamente el contenido de archivos grandes sin abrirlos en un editor gráfico. [web:1][web:3]

Comandos:

- `head -n 5 datos.csv`: muestra las primeras 5 líneas.
- `tail -n 5 datos.csv`: muestra las últimas 5 líneas.
- `wc -l datos.csv`: cuenta las líneas del archivo.
- `less datos.csv`: permite navegar por el archivo.

```
# Ver las primeras líneas de sensores.csv
cd proyecto_ia/data/raw
head -n 5 sensores.csv

# Ver las últimas líneas (útil para ver fechas recientes)
tail -n 5 sensores.csv

# Contar cuántos registros hay
wc -l sensores.csv

# Navegar por el archivo (q para salir)
less sensores.csv
```

Ejercicio

1. Usa `head` y `tail` para identificar si el archivo está ordenado por fecha.
2. Usa `wc -l` para estimar cuántos días de datos tienes si hay 1 registro por minuto.

3.2. Datos Reales y Errores Comunes

Los datos reales contienen errores:

- Valores faltantes representados como cadenas vacías o símbolos especiales.
- Separadores inconsistentes (comas, punto y coma, tabuladores).
- Registros corruptos con caracteres no ASCII. [web:11][web:23]

```
# Buscar columnas vacías consecutivas ,, que pueden indicar datos
    faltantes
grep ",," sensores.csv | head

# Detectar caracteres no ASCII (posibles errores de codificación)
grep -P "[^\x00-\x7F]" sensores.csv | head
```

Atención

Antes de entrenar cualquier modelo, debes conocer la calidad de tus datos. No confíes en que el archivo está limpio solo porque se abre en una hoja de cálculo. [web:11]

3.3. Pipelines con Pipes

El operador | (pipe) conecta la salida de un comando con la entrada de otro. Permite encadenar operaciones simples para construir un procesamiento más complejo. [web:3][web:20]

```
# Contar cuántas veces aparece cada id_sensor en el archivo
cut -d',' -f1 sensores.csv | sort | uniq -c | sort -nr | head

# Obtener los 10 valores de temperatura más altos registrados
cut -d',' -f3 sensores.csv | sort -nr | head

# Filtrar solo las líneas de un sensor específico y contarlas
grep "SENSOR_05" sensores.csv | wc -l
```

Ejercicio de pipeline

Construye un pipeline que:

1. Se quede solo con las columnas `id_sensor,temperatura`.
2. Ordene por temperatura descendente.
3. Muestre las 5 mediciones más altas por pantalla.

3.4. Procesamiento con awk

`awk` es un mini-lenguaje para procesar texto y columnas; permite calcular estadísticas básicas directamente desde la terminal. [web:3][web:19][web:23]

```
# Calcular la temperatura promedio (suponiendo que está en la columna 3)
awk -F',' '{s+=$3} END {print "Temperatura promedio:", s/NR}' sensores.
    csv

# Calcular la humedad máxima (columna 4)
awk -F',' 'NR>1 {if($4>max) max=$4} END {print "Humedad máxima:", max}'
    sensores.csv

# Filtrar solo filas con humedad menor a 30
awk -F',' '$4 < 30 {print $0}' sensores.csv | head
```

Ejercicio con awk

1. Usa `awk` para calcular la temperatura mínima.
2. Calcula la suma total de registros donde `humedad < 40`.
3. Guarda las filas con `humedad > 80` en un archivo `alertas_humedad.csv`.

4. Fase 3: Profesionalismo y Control

En esta fase transformarás tu carpeta de archivos sueltos en un proyecto versionado con Git, usando buenas prácticas de colaboración. [web:18][web:22][web:25]

4.1. Git como Bitácora Científica

Git es un sistema de control de versiones distribuido que registra la historia de tu proyecto, permitiéndote volver a estados anteriores y colaborar con otros. [web:18][web:22]

Flujo básico:

1. `git init`: inicializa el repositorio.
2. `git status`: muestra el estado actual.
3. `git add`: selecciona cambios para guardar.
4. `git commit`: guarda un "fotograma" con mensaje.

```
cd ~/proyecto_ia

# Inicializar repositorio Git
git init

# Ver estado (archivos sin seguimiento)
git status

# Añadir todos los archivos para el primer commit
git add .

# Crear el commit inicial con un mensaje descriptivo
git commit -m "Inicial: estructura del proyecto de monitoreo de sensores"

# Ver el historial de commits
git log --oneline
```

Buenas prácticas de commits

- Haz commits pequeños y frecuentes.
- Usa mensajes que expliquen el *por qué* del cambio.
- Agrupa cambios relacionados en un mismo commit. [web:9][web:15]

4.2. Uso de .gitignore

El archivo `.gitignore` indica a Git qué archivos o carpetas no deben versionarse (datos brutos, modelos grandes, archivos temporales). [web:9][web:18]

```
# Ejemplo de .gitignore para un proyecto de ciencia de datos
venv/
data/raw/
data/intermediate/
*.pth
*.log
```

```
*.tmp
.ipynb_checkpoints/
__pycache__/
```

```
# Crear el archivo .gitignore
cat > .gitignore << EOF
venv/
data/raw/
data/intermediate/
*.pth
*.log
*.tmp
.ipynb_checkpoints/
__pycache__/
EOF

git add .gitignore
git commit -m "Configura .gitignore para datos brutos y artefactos"
```

Regla práctica

Nunca subas datos sensibles ni archivos de gran tamaño al repositorio. Para datos grandes, considera herramientas como Git LFS u otros almacenes externos. [web:9][web:22]

4.3. Ramas sencillas para experimentos

Aunque este manual se centra en lo básico, es útil conocer el concepto de rama (*branch*) para experimentar sin romper la versión estable. [web:22][web:25]

```
# Crear una rama para experimentar con una nueva limpieza
git checkout -b feature/limpieza_avanzada

# Modificar scripts, probar...
git add scripts/limpieza_sensores.sh
git commit -m "Implementa limpieza de valores atipicos"

# Volver a la rama principal
git checkout main  # o master, según el nombre que uses
```

Ejercicio de bitácora

Crea un repositorio Git para tu proyecto de sensores y:

- Haz un commit inicial con la estructura de carpetas.
- Haz un segundo commit cuando añadas el primer script de limpieza.
- Escribe mensajes de commit que indiquen claramente qué cambió y por qué.

4.4. Entornos Virtuales de Python

Un entorno virtual aísla las dependencias de un proyecto para evitar conflictos entre versiones de librerías. [web:10]

```
# Crear un entorno virtual llamado ia_env
cd ~/proyecto_ia
python -m venv ia_env

# Activar el entorno (en Linux/macOS)
source ia_env/bin/activate

# Instalar librerías de ciencia de datos
pip install pandas numpy

# Congelar las versiones instaladas
pip freeze > requirements.txt

# Desactivar el entorno cuando termines
deactivate
```

Para recrear el entorno en otra máquina:

```
python -m venv ia_env
source ia_env/bin/activate
pip install -r requirements.txt
```

Ventajas de entornos virtuales

- Cada proyecto tiene sus propias versiones de librerías.
- Facilitan la reproducibilidad al compartir `requirements.txt`.
- Evitan romper otros proyectos al actualizar paquetes. [web:7][web:10]

5. Fase 4: Ciencia Reproducible

Una buena estructura de proyecto permite que cualquier persona entienda dónde están los datos, el código, los modelos y los reportes. [web:9][web:12]

```
proyecto_ia/
  data/
     raw/          # datos brutos (no limpios)
     processed/    # datos listos para análisis/modelado
  scripts/         # scripts de limpieza, descarga, validación
  notebooks/       # cuadernos exploratorios (Jupyter)
  models/          # modelos entrenados y artefactos
  reports/         # informes, figuras y tablas
  README.md        # descripción del proyecto
```

Regla

Si no está versionado, no existe.

```
# Crear toda la estructura con un solo comando
mkdir -p proyecto_ia/{data/raw,data/processed,scripts,notebooks,models,
    reports}

# Crear un README mínimo
cat > README.md << EOF
# Proyecto IA - Monitoreo de Sensores

Este proyecto analiza datos de sensores de una finca para detectar
anomalías de humedad y temperatura.
EOF
```

Ejercicio de organización

- Decide en qué carpeta guardarías un script que descarga datos desde una API.
- Decide dónde guardarías un gráfico de correlación temperatura-humedad.
- Anota en el README un breve flujo: de `data/raw` a `data/processed`.

6. Fase 5: Del Shell a Python

En esta fase se muestra cómo combinar la potencia de la terminal con Python y `pandas` para análisis más avanzados. [web:11]

6.1. Llamar Python desde la Terminal

```
# Ejecutar un bloque de Python directamente desde Bash
python - << EOF
import pandas as pd

# Cargar datos de sensores
df = pd.read_csv("data/raw/sensores.csv")

# Mostrar resumen estadístico
print(df.describe())

# Filtrar registros con humedad mayor a 80
alertas = df[df["humedad"] > 80]
print("Numero de alertas de alta humedad:", len(alertas))
EOF
```

6.2. Replicar un pipeline de shell en Python

Supón que en Bash calculaste la temperatura promedio con `awk`. Ahora harás lo mismo en Python. [web:11][web:19]

```
python - << EOF
import pandas as pd

df = pd.read_csv("data/raw/sensores.csv")

# Temperatura promedio
print("Temperatura promedio:", df["temperatura"].mean())

# Humedad máxima
print("Humedad máxima:", df["humedad"].max())

# Guardar solo columnas clave en processed
cols = ["id_sensor", "fecha", "temperatura", "humedad"]
df[cols].to_csv("data/processed/sensores_limpios.csv", index=False)
EOF
```

Ejercicio de traducción

Toma un pipeline de Bash de la Fase 2 que:

- Filtra registros por un `id_sensor`.
- Calcula una estadística (promedio, máximo, mínimo).
- Guarda un archivo filtrado.

Escríbelo ahora en Python usando `pandas`.

Reto Profesional

Construye un script Bash (`scripts/setup_proyecto.sh`) que:

1. Cree la estructura del proyecto (`data/`, `scripts/`, `models/`, `reports/`).
2. Descargue un dataset público (por ejemplo, desde una URL) y lo guarde en `data/raw`.
3. Valide los datos: cuente filas, busque valores faltantes y caracteres extraños.
4. Genere un reporte de texto en `reports/reporte_inicial.txt` con los resultados.
5. Registre todo en Git con commits claros y ordenados.

Idempotencia

El script debe ejecutarse múltiples veces sin fallar:

- Verifica si las carpetas ya existen antes de crearlas.
- No descargues el archivo si ya está presente.
- Evita sobrescribir reportes sin avisar.

Extensión opcional

Añade una llamada a Python al final del script para generar un pequeño resumen estadístico de las columnas numéricas y guardarlo en `reports/estadisticas.csv`.

7. Fase 6: Docker para Ciencia de Datos

Docker permite crear *contenedores*: ambientes ligeros que incluyen sistema base, librerías y tu código, de forma reproducible. [web:30][web:37] En ciencia de datos se usa para asegurar que un experimento funcione igual en tu portátil, en un servidor o en la nube.

7.1. Conceptos Fundamentales

Objetivo

Comprender la diferencia entre imagen y contenedor, y por qué Docker mejora la reproducibilidad frente a instalaciones manuales. [web:30][web:36]

Conceptos clave:

- **Imagen**: plantilla inmutable con sistema base (por ejemplo, `python:3.10`), librerías y tu código.
- **Contenedor**: instancia en ejecución de una imagen; se puede iniciar, detener y borrar sin afectar la imagen.
- **Dockerfile**: archivo de texto que describe cómo construir una imagen.
- **Registry**: lugar donde se publican y comparten imágenes, como Docker Hub. [web:36][web:40]

```
# Ver versión de Docker instalada
docker --version

# Descargar una imagen de Python oficial
docker pull python:3.10-slim

# Ver imágenes disponibles localmente
docker images

# Ver contenedores en ejecución
docker ps
```

Ejercicio

1. Instala Docker en tu sistema y ejecuta `docker run hello-world`.
2. Verifica que entiendes la diferencia entre `docker images` y `docker ps`.

7.2. Primer Contenedor para Análisis

Para probar Docker, puedes usar una imagen oficial de Python y ejecutar un análisis simple desde la línea de comandos. [web:30][web:36]

```
# Ejecutar Python interactivo en un contenedor temporal
docker run -it --rm python:3.10-slim python
```

```
# Ejecutar un script simple directamente
echo "print(2 + 3)" > test.py
docker run --rm -v $(pwd):/app -w /app python:3.10-slim python test.py
```

En este ejemplo:

- `-v $(pwd):/app` monta tu carpeta actual dentro del contenedor en `/app`.
- `-w /app` fija el directorio de trabajo dentro del contenedor.
- `--rm` borra el contenedor al finalizar, evitando basura. [web:36]

Datos y contenedores

Nunca guardes datos importantes sólo dentro del contenedor. Usa montajes de volúmenes para que los datos vivan en tu sistema o en volúmenes persistentes. [web:26][web:27]

7.3. Dockerfile para un Proyecto de Datos

La forma profesional de usar Docker es escribir un `Dockerfile` que defina el ambiente para tu proyecto de ciencia de datos. [web:26][web:29][web:36]

```
# Dockerfile para proyecto_ia

# 1. Imagen base con Python
FROM python:3.10-slim

# 2. Directorio de trabajo dentro del contenedor
WORKDIR /app

# 3. Copiar solo requirements primero (mejora la cache de build)
COPY requirements.txt .

# 4. Instalar dependencias
RUN pip install --no-cache-dir -r requirements.txt

# 5. Copiar el resto del código
COPY . .

# 6. Crear directorios de datos (si no existen)
RUN mkdir -p /app/data/raw /app/data/processed /app/reports

# 7. Declarar un volumen para datos (opcional)
VOLUME ["/app/data"]

# 8. Comando por defecto (se puede sobreescribir)
CMD ["python", "scripts/analisis_inicial.py"]
```

```
# Construir la imagen (ejecutar en la raíz del proyecto)
docker build -t proyecto_ia:latest .

# Ejecutar un contenedor montando tus datos locales
docker run --rm \
  -v $(pwd)/data:/app/data \
  -v $(pwd)/reports:/app/reports \
  proyecto_ia:latest
```

Buenas prácticas de Dockerfile

- Usa imágenes base oficiales y ligeras, como `python:3.X-slim`.
- Copia primero `requirements.txt` para aprovechar la cache de instalación de librerías.
- Evita copiar datos brutos dentro de la imagen; monta datos desde fuera. [web:26][web:32]

7.4. Jupyter y Entornos Interactivos en Docker

Es habitual correr Jupyter Lab/Notebook dentro de un contenedor con todas las librerías de análisis. [web:33][web:34]

```
# Dockerfile minimal para Jupyter con ciencia de datos
FROM python:3.10-slim

WORKDIR /app

RUN pip install --no-cache-dir \
    jupyterlab pandas numpy matplotlib seaborn

# Exponer el puerto de Jupyter
EXPOSE 8888

CMD ["jupyter", "lab", "--ip=0.0.0.0", "--port=8888", "--no-browser", "--
    allow-root"]
```

```
# Construir imagen
docker build -t ds-notebook:latest .

# Ejecutar Jupyter con acceso a tu código y datos
docker run --rm -p 8888:8888 \
  -v $(pwd):/app \
  ds-notebook:latest
```

Luego abre en el navegador la URL con el token que se muestra en la terminal. Así puedes trabajar en notebooks en un ambiente controlado y replicable. [web:33][web:34]

Ejercicio con Jupyter

1. Crea un `Dockerfile` para correr Jupyter con tus librerías favoritas (`pandas`, `scikit-learn`, `xgboost`).
2. Ejecuta un notebook que cargue `data/processed/sensores_limpios.csv` y genere un gráfico guardado en `reports/`.

7.5. docker-compose para Pipelines de Datos

Cuando tu proyecto tiene varios servicios (por ejemplo, base de datos + notebook + API), `docker-compose` te ayuda a orquestarlos con un solo archivo. [web:27][web:30][web:36]

```
# docker-compose.yml
version: "3.8"
```

```yaml
services:
  db:
    image: postgres:13
    environment:
      POSTGRES_DB: sensoresdb
      POSTGRES_USER: ds_user
      POSTGRES_PASSWORD: ds_pass
    volumes:
      - postgres_data:/var/lib/postgresql/data

  notebook:
    build: .
    ports:
      - "8888:8888"
    depends_on:
      - db
    volumes:
      - .:/app

volumes:
  postgres_data:
```

```bash
# Levantar todos los servicios
docker compose up

# Detenerlos
docker compose down
```

Esta estructura permite reproducir un entorno completo de análisis, con almacenamiento persistente en el volumen `postgres_data`. [web:27]

7.6. Buenas Prácticas para Ciencia de Datos

Checklist profesional

- Versiona el `Dockerfile` junto con tu código y `requirements.txt`. [web:30][web:32]
- Usa etiquetas (`tags`) significativas en las imágenes, por ejemplo `:v1.0`, `:exp-humedad-2026-01`. [web:32][web:35]
- Mantén las imágenes pequeñas: elimina paquetes innecesarios y usa `--no-cache-dir` en `pip`. [web:32][web:38]
- No incluyas secretos (claves, tokens) dentro de la imagen; pásalos como variables de entorno o mediante volúmenes. [web:32][web:38]

Reto Docker

Crea una imagen para tu *Reto Final Profesional* que:

- Construya la estructura del proyecto y ejecute el pipeline completo de validación y reporte.
- Use volúmenes para leer datos desde `data/raw` y escribir resultados en `reports/`.
- Pueda ejecutarse con un único comando `docker run ...` en cualquier máquina con Docker instalado.

Anexo: Alias de Seguridad y Productividad

```
# Alias de seguridad para evitar borrados accidentales
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'

# Alias útiles para proyectos de datos
alias ll='ls -lh'
alias la='ls -lha'
alias actenv='source ia_env/bin/activate'
```

Siguiente nivel

A partir de este manual, el siguiente paso es integrar estas prácticas con herramientas de MLOps (como `make`, `Docker` y sistemas de CI/CD) para automatizar aún más el ciclo de vida de modelos. [web:9][web:25]