

# Semana 1: El Entorno del Ingeniero

Linux, Shell Scripting y Control de Versiones

Curso IA Agroindustria 2026

Enero 2026

## Resumen

El 80 % del éxito de un proyecto de Inteligencia Artificial depende de la ingeniería de datos, automatización y reproducibilidad. Esta semana aprenderás las herramientas fundamentales: terminal Linux, procesamiento de datos con pipes y control de versiones con Git.

## Índice

<b>1. Fase 1: Supervivencia en la Terminal</b>	<b>5</b>
1.1. Navegación Básica . . . . .	5
1.2. Gestión de Archivos . . . . .	6
1.3. Comandos Clave de Científico de Datos . . . . .	7
<b>2. Fase 2: Procesamiento y Automatización</b>	<b>9</b>
2.1. Inspección de Datos . . . . .	9
2.2. Datos Reales y Errores Comunes . . . . .	10
2.3. Pipelines con Pipes . . . . .	10
2.4. Procesamiento con awk . . . . .	11
<b>3. Fase 3: Control de Versiones con Git</b>	<b>13</b>
3.1. Git como Bitácora Científica . . . . .	13
3.2. Uso de .gitignore . . . . .	13
3.3. Flujo de Trabajo Básico . . . . .	14
<b>4. Casos de Uso por Carrera</b>	<b>16</b>
4.1. Agroindustria Alimenticia . . . . .	16
4.2. Agronomía . . . . .	16
4.3. Ingeniería Agrícola . . . . .	16
4.4. Ingeniería Ambiental . . . . .	17
<b>5. Entregable A1: Script de Setup Reproducible</b>	<b>18</b>
5.1. Conceptos Nuevos Necesarios . . . . .	18
5.2. Requisitos Mínimos . . . . .	18
5.3. Plantilla del Script (Base) . . . . .	18

<b>6. Desafío: Minería de Datos con Bash</b>	<b>19</b>
6.1. Instrucciones . . . . .	19
6.2. Entrega . . . . .	20

## Objetivos de Aprendizaje

Al finalizar esta semana podrás:

- Navegar y manipular archivos desde la terminal Linux sin GUI.
- Construir pipelines de procesamiento de datos con pipes y **awk**.
- Usar Git como bitácora científica de tus experimentos.
- Automatizar tareas repetitivas con scripts Bash reproducibles.

### Tiempo estimado

**En clase (4h):** Fases 1–3 (terminal + Git básico).

**Autoestudio:** Fases 4–5 (estructura de proyectos + Python desde terminal).

## 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. 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. 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:

entrada → proceso → salida

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

## 1.1. Navegación Básica

### Objetivo

Aprender a saber dónde estás, qué archivos hay y cómo moverte entre carpetas de un proyecto de datos.

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
/workspaces/curso-ia-agroindustria-2

# Listar el contenido de la carpeta actual
ls -lh
total 28K
-rw-rw-rw- 1 vscode root 1.1K Jan 19 03:22 LICENSE
-rw-rw-rw- 1 vscode root 4.7K Jan 19 03:22 README.md
drwxrwxrwx+ 5 vscode root 4.0K Jan 19 03:22 docs
drwxrwxrwx+ 2 vscode root 4.0K Jan 19 03:22 reports
drwxrwxrwx+ 2 vscode root 4.0K Jan 19 03:22 scripts
drwxrwxrwx+ 4 vscode root 4.0K Jan 19 03:22 talleres

# Para crear los archivos con datos ejecuta el siguiente script:
./scripts/setup_proyecto.sh

# Se creó la carpeta data, así que vamos a listar solo archivos CSV
ls data/raw/

datos_sensores.csv
```

**Ejercicio guiado (15 min)**

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.

## 1.2. Gestión de Archivos

**Objetivo**

Crear, copiar, mover y eliminar archivos y carpetas para organizar un proyecto de datos.

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 ../data/raw/datos_sensores.csv data/raw/

# Verificar que se copió
ls data/raw/

datos_sensores.csv

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

# Verificar el cambio
ls
README.md  data/  reports/  scripts/

# 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.

**Ejercicio (20 min)**

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.

### 1.3. 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.

Comandos útiles:

- `du -sh *`: muestra el tamaño de cada archivo/carpetas.
- `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 datos_sensores.csv | head

sepal_length
5.1
4.9
4.7
```

```
4.6
5.0
5.4
4.6
5.0
4.4

# Obtener lista ordenada y única de 10 sensores
cut -d',' -f1 datos_sensores.csv | sort | uniq | head

4.3
4.4
4.5
4.6
4.7
4.8
4.9
5.0
5.1
5.2
```

### Ejercicio aplicado (25 min)

Trabajaremos con el dataset `datos_sensores.csv` (originalmente Iris), que contiene medidas morfológicas. Las columnas son: `sepal_length`, `sepal_width`, `petal_length`, `petal_width`, `species`.

1. **Exploración Categórica:** Obtén la lista única de especies (`species`, columna 5) y cuenta cuántos registros existen por cada una.

```
# Pista: usa cut, sort y uniq -c
tail -n +2 datos_sensores.csv | cut -d',' -f5 | ...
```

2. **Conteo Total:** Verifica si el dataset está completo contando el número total de filas de datos (excluyendo el encabezado).
3. **Filtrado Numérico (Nuevo):** Encuentra cuántas flores tienen un `petal_length` (columna 3) mayor a 1.5 cm.

```
# Pista: usa awk con una condición condicional
awk -F',' ' $3 > 1.5 {print $0}' datos_sensores.csv | wc -l
```

4. **Detección de Outliers (Reto):** Identifica las 3 flores con el `sepal_width` (columna 2) más grande de todo el archivo.

```
# Pista: ordena numéricamente de forma descendente (sort -nr)
tail -n +2 datos_sensores.csv | sort -t',' -k2 -nr | head -n 3
```



## 2. 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.

### 2.1. Inspección de Datos

#### Objetivo

Inspeccionar rápidamente el contenido de archivos grandes sin abrirlos en un editor gráfico.

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.

```
# =====
# Inspección rápida del dataset
# Archivo real: data/raw/datos_sensores.csv (tiene encabezado)
# Columnas: sepal_length, sepal_width, petal_length, petal_width, species
# =====

# 0) (Recomendado) Ir a la raíz del repositorio para que las rutas
#     funcionen
cd ../../../../curso-ia-agroindustria-2/

# 1) Ver el encabezado (nombres de columnas)
head -n 1 data/raw/datos_sensores.csv

sepal_length,sepal_width,petal_length,petal_width,species

# 2) Ver las primeras 5 líneas del archivo (incluye encabezado)
head -n 5 data/raw/datos_sensores.csv

sepal_length,sepal_width,petal_length,petal_width,species
5.1,3.5,1.4,0.2,setosa
4.9,3.0,1.4,0.2,setosa
4.7,3.2,1.3,0.2,setosa
4.6,3.1,1.5,0.2,setosa

# 3) Ver las últimas 5 líneas (útil para validar que el archivo no está
#     cortado)
tail -n 5 data/raw/datos_sensores.csv

6.7,3.0,5.2,2.3, virginica
6.3,2.5,5.0,1.9, virginica
6.5,3.0,5.2,2.0, virginica
6.2,3.4,5.4,2.3, virginica
5.9,3.0,5.1,1.8, virginica
```

```
# 4) Contar líneas totales (incluye el encabezado)
wc -l data/raw/datos_sensores.csv

151 data/raw/datos_sensores.csv

# 5) Contar registros "reales" (sin encabezado)
tail -n +2 data/raw/datos_sensores.csv | wc -l

150

# 6) Navegar por el archivo sin abrirlo en un editor (q para salir)
less data/raw/datos_sensores.csv
```

## 2.2. 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.

```
# Buscar columnas vacías consecutivas ,, que pueden indicar datos
faltantes
grep ",," data/raw/datos_sensores.csv | head

# Detectar caracteres no ASCII (posibles errores de codificación)
grep -P "[^\x00-\x7F]" data/raw/datos_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.

## 2.3. 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.

```
# Contar cuántas veces aparece un dato id_sensor en el archivo, conteo
rápido de los 10 sensores más frecuentes
cut -d',' -f1 data/raw/datos_sensores.csv | sort | uniq -c | sort -nr |
head

10 5.0
9 6.3
9 5.1
8 6.7
8 5.7
7 6.4
7 5.8
7 5.5
6 6.1
```

```
6 6.0
```

```
# Obtener los 10 valores de temperatura más altos registrados
cut -d',' -f3 data/raw/datos_sensores.csv | sort -nr | head

# Filtrar solo las líneas de un sensor específico y contarlas
grep "setosa" data/raw/datos_sensores.csv | wc -l
```

### Ejercicio de pipeline (30 min)

Construye un pipeline usando `cut`, `sort` y `head` para analizar las flores más grandes. El objetivo es:

1. Extraer solo las columnas de **especie** (columna 5) y **longitud del pétalo** (columna 3).
2. Ordenar los resultados por longitud de pétalo de forma **descendente** (de mayor a menor).
3. Mostrar en pantalla las **5 flores con los pétalos más largos** del dataset.

Comando esperado:

```
# Pista: Recuerda que sort necesita saber qué columna usar (-k) o
# recibir solo los datos a ordenar
cut -d',' -f5,3 data/raw/datos_sensores.csv | sort -t',' -k2 -nr |
head -n 5
```

## 2.4. Procesamiento con awk

`awk` es un mini-lenguaje potente para procesar columnas y calcular estadísticas sin abrir Excel. Vamos a usarlo para analizar las dimensiones de las flores.

```
# Calcular la longitud promedio del sépalo (columna 1)
# NR = Número de Registro (filas totales procesadas)
awk -F',' 'NR>1 {s+=$1} END {print "Longitud promedio sépalo:", s/(NR-1)}'
  data/raw/datos_sensores.csv

# Encontrar el ancho de pétalo máximo (columna 4)
awk -F',' 'NR>1 {if($4>max) max=$4} END {print "Ancho máximo pétalo:",
  max}' data/raw/datos_sensores.csv

# Filtrar flores pequeñas: mostrar solo aquellas con largo de pétalo <
  1.5 cm (columna 3)
awk -F',' '$3 < 1.5 {print $0}' data/raw/datos_sensores.csv | head
```

**Ejercicio con awk (30 min)**

Usando el archivo `data/raw/datos_sensores.csv`:

1. Usa **awk** para calcular el **promedio del ancho de sépalo** (columna 2).
2. Cuenta cuántas flores son de la especie **"virginica"** (columna 5).
3. Filtra todas las flores con `petal_length > 5.0` y guarda esas filas en un nuevo archivo llamado `flores_gigantes.csv`.

### 3. Fase 3: Control de Versiones con Git

En esta fase transformarás tu carpeta de archivos sueltos en un proyecto versionado con Git, usando buenas prácticas de colaboración.

#### 3.1. 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.

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 "feat: estructura inicial del proyecto 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.
- Usa prefijos: `feat:`, `fix:`, `docs:`, `refactor:`.

#### 3.2. Uso de `.gitignore`

El archivo `.gitignore` indica a Git qué archivos o carpetas no deben versionarse (datos brutos, modelos grandes, archivos temporales).

```
# Crear el archivo .gitignore
cat > .gitignore << EOF
# Entornos virtuales
```

```
venv/  
ia_env/  
  
# Datos brutos (no versionables)  
data/raw/  
data/interim/  
  
# Artefactos Python  
*.pyc  
__pycache__/  
.ipynb_checkpoints/  
  
# Modelos pesados  
*.pth  
*.h5  
  
# Logs y temporales  
*.log  
*.tmp  
EOF  
  
git add .gitignore  
git commit -m "docs: configura .gitignore para datos 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.

## 3.3. Flujo de Trabajo Básico

Vamos a simular un ciclo de desarrollo real: crear una nueva herramienta, verificarla y guardarla en el historial.

```
# 1. Crear un nuevo script de validación  
echo "#!/bin/bash" > scripts/validar_datos.sh  
echo "echo 'Contando filas en el dataset maestro...'" >> scripts/  
    validar_datos.sh  
echo "wc -l data/raw/datos_sensores.csv" >> scripts/validar_datos.sh  
  
# 2. Hacerlo ejecutable  
chmod +x scripts/validar_datos.sh  
  
# 3. Verificar qué ha cambiado en el proyecto  
git status  
# (Debería mostrar scripts/validar_datos.sh como 'Untracked')  
  
# 4. Guardar el cambio en el historial (Staging + Commit)  
git add scripts/validar_datos.sh  
git commit -m "feat: agrega script para contar registros totales"  
  
# 5. Visualizar la historia de cambios  
git log --oneline --graph --all
```

**Ejercicio de Git (30 min)**

Sigue estos pasos en tu terminal (asegúrate de estar en la raíz del proyecto):

1. Crea un archivo nuevo `scripts/resumen.sh` que imprima las primeras 5 líneas de tus datos usando `head`.
2. Ejecuta `git status` para confirmar que Git detecta el archivo nuevo.
3. Añádelo al área de preparación con `git add scripts/resumen.sh`.
4. Haz un commit con el mensaje: "feat: script para previsualizar datos".
5. Modifica el archivo `README.md` agregando una línea al final y haz un nuevo commit.
6. Usa `git log` para comprobar que tus 2 nuevos commits aparecen en la lista.

## 4. Casos de Uso por Carrera

Los comandos aprendidos se aplican distinto según el dominio:

### 4.1. Agroindustria Alimenticia

#### Caso: Control de calidad de lotes

Archivos CSV con registros de inspección visual (color, tamaño, defectos) de fruta para exportación. Cada fila = 1 unidad inspeccionada, con columnas: lote\_id, fecha, peso\_g, calibre, color, defectos.

Pipeline típico:

```
# Cuántos lotes inspeccionados hoy
grep "$(date +%Y-%m-%d)" inspeccion.csv | cut -d',' -f1 | sort |
  uniq | wc -l

# Lotes con >5% de defectos (columna 6 > 5)
awk -F',' ' $6 > 5 {print $1}' inspeccion.csv | sort | uniq >
  lotes_rechazados.txt
```

### 4.2. Agronomía

#### Caso: Monitoreo de rendimiento por lote

CSV con datos de cosecha: lote\_id, ha, kg\_cosechados, fecha.

Pipeline típico:

```
# Calcular toneladas/ha promedio por lote
awk -F',' 'NR>1 {print $1, $3/$2/1000}' cosecha.csv | \
  awk '{s[$1]+=$2; c[$1]++;} END {for(l in s) print l, s[l]/c[l]}' |
  \
  sort -k2 -nr > rendimiento_promedio.txt
```

### 4.3. Ingeniería Agrícola

#### Caso: Telemetría de riego

Datos de sensores de humedad cada 15 min:  
sensor\_id, timestamp, humedad\_suelo, presion\_agua.

Pipeline típico:

```
# Detectar sensores con humedad <20% (alerta riego)
awk -F',' ' $3 < 20 {print $1,$2,$3}' riego.csv | \
  sort | uniq > alertas_riego.txt

# Contar alertas por sensor
cut -d' ' -f1 alertas_riego.txt | sort | uniq -c | sort -nr
```



## 4.4. Ingeniería Ambiental

### Caso: Monitoreo de calidad de agua

CSV con muestreos: estacion\_id, fecha, pH, turbidez\_NTU, coliformes\_UFC.  
Pipeline típico:

```
# Estaciones fuera de norma (pH <6.5 o >8.5)
awk -F',' ' $3<6.5 || $3>8.5 {print $1,$2,$3}' calidad_agua.csv >
  pH_fuera_norma.txt

# Promedios de turbidez por estación
awk -F',' ' NR>1 {s[$1]+=$4; c[$1]++} END {for(e in s) print e, s[e]/
  c[e]}' \
  calidad_agua.csv | sort -k2 -nr
```

### Ejercicio multi-carrera (30 min)

Adapta uno de estos pipelines a tu dataset real (o simula uno pequeño con 20 filas en Excel y exporta a CSV). Documenta en tu README qué hizo cada comando y qué insights descubriste.

## 5. Entregable A1: Script de Setup Reproducible

### Objetivo

Construir un script Bash (`scripts/setup_proyecto.sh`) que automatice la creación de estructura, descarga de datos reales y validación básica.

### 5.1. Conceptos Nuevos Necesarios

Para este script usaremos herramientas de automatización que no hemos visto en detalle:

- **`curl -o destino url`**: Descarga un archivo desde internet. Es como “Guardar como...” pero desde la terminal.
- **`if [ ! -f archivo ]; then ... fi`**: Bloque condicional que verifica si un archivo **NO** existe antes de intentar crearlo o descargarlo (evita sobrescribir trabajo previo).
- **`set -euo pipefail`**: Un “modo estricto” para Bash. Hace que el script se detenga inmediatamente si ocurre cualquier error, en lugar de seguir ejecutando comandos a ciegas.

### 5.2. Requisitos Mínimos

1. **Automatización de Carpetas**: El script debe crear `data/raw`, `data/processed`, `scripts` y `reports` usando `mkdir -p` (para no fallar si ya existen).
2. **Descarga Automática**: Debe descargar el dataset Iris desde una URL pública (ver plantilla) y guardarlo como `data/raw/datos_sensores.csv`.
3. **Validación de Calidad**: Generar un archivo `reports/validacion_inicial.txt` con:
  - Fecha de ejecución (`date`).
  - Conteo de filas y columnas.
  - Detección de posibles valores vacíos (`grep ",,"`).
4. **Control de Versiones**: El repositorio final debe tener un `.gitignore` adecuado y al menos 3 commits descriptivos.

### 5.3. Plantilla del Script (Base)

Copia este código en `scripts/setup_proyecto.sh` y estúdialo. Ya incluye la URL correcta para el ejercicio.

```
#!/usr/bin/env bash
# Modo estricto: el script falla si algún comando falla
set -euo pipefail

echo ">>> Iniciando setup del proyecto..."
```

```
# 1. Crear estructura de carpetas (silencioso si ya existen)
mkdir -p data/raw data/processed scripts reports

# 2. Definir variables de descarga
# Usamos el dataset Iris como "datos de sensores"
DATA_URL="https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv"
DATA_FILE="data/raw/datos_sensores.csv"

# 3. Descargar solo si no existe (Idempotencia)
if [ ! -f "$DATA_FILE" ]; then
    echo ">>> Descargando datos desde GitHub..."
    # -L sigue redirecciones, -o guarda en archivo
    curl -L -o "$DATA_FILE" "$DATA_URL"
else
    echo ">>> Los datos ya existen, saltando descarga."
fi

# 4. Generar reporte de validación
echo ">>> Generando reporte..."
REPORT_FILE="reports/validacion_inicial.txt"

{
    echo "=== REPORTE DE VALIDACIÓN ==="
    echo "Fecha: $(date)"
    echo "Archivo: $DATA_FILE"
    echo "Tamaño: $(du -h $DATA_FILE | cut -f1)"
    echo "Filas totales: $(wc -l < $DATA_FILE)"
    # Truco: contar comas en la primera línea + 1 para saber columnas
    echo "Columnas detectadas: $(head -n1 $DATA_FILE | tr ',' '\n' | wc -l)"
} > "$REPORT_FILE"

echo ">>> Setup completado exitosamente. Revisa $REPORT_FILE"
```

### Tips de Ejecución

Recuerda dar permisos de ejecución antes de probarlo: `chmod +x scripts/setup_proyecto.sh`  
Luego ejecútalo desde la raíz del proyecto: `./scripts/setup_proyecto.sh`

## 6. Desafío: Minería de Datos con Bash

Una vez ejecutado tu script de setup (`setup_proyecto.sh`), tendrás los datos listos en `data/raw/datos_sensores.csv`. Ahora deberás demostrar tu habilidad extrayendo información clave usando solo la terminal.

### 6.1. Instrucciones

Resuelve los siguientes retos y documenta los comandos que usaste en tu archivo `README.md`, bajo una nueva sección titulada **Resultados del Análisis**.

### Reto 1: Exploración de Especies

¿Cuántas muestras de cada especie contiene el archivo?

- Objetivo: Obtener una tabla de frecuencias.
- Pista: Usa el pipeline `cut → sort → uniq -c`.
- La columna de especies es la número 5.

### Reto 2: Detección de Anomalías (Outliers)

Encontremos las flores inusualmente grandes.

- Objetivo: Mostrar las 5 filas con el **sépalo más largo** (columna 1).
- Pista: Recuerda usar `sort -t',' -k1 -nr`.

### Reto 3: Filtrado Avanzado (awk)

Filtra y cuenta cuántas flores cumplen la condición de ser **"gigantes"**.

- Condición: `petal_length` (columna 3) mayor a **6.0** cm.
- Comando sugerido: `awk -F',' '$3 > 6.0 {print $0}' data/raw/datos_sensores.csv | wc -l`
- Pregunta para el reporte: ¿Cuántas flores cumplen esta condición?

### Ejemplo de documentación en README.md

Tu README.md debería verse así al final:

```
# Resultados del Análisis
1. Conteo de especies:
- 50 setosa
- 50 versicolor...
(Comando usado: cut -d',' -f5 ...)
...
```

## 6.2. Entrega

- Sube tu repositorio a GitHub.
- Asegúrate de que el README.md explique cómo ejecutar tu script de setup.
- Comparte la URL del repositorio.

## Material de Autoestudio y Profundización

Esta sección resume las herramientas clave vistas y prepara el terreno para el análisis avanzado con Python.

### Resumen de Comandos (Cheat Sheet)

Guarda esta referencia rápida. Combinar estos comandos te permite hacer una primera exploración de datos en segundos.

- **Exploración rápida:**
  - `head -n 5 file.csv`: Ver encabezados y primeras filas.
  - `wc -l file.csv`: Contar tamaño (filas).
  - `grep .^ROR"file.log`: Buscar textos específicos.
- **Filtrado y Selección (Columns):**
  - `cut -d',' -f1,3`: Extraer columnas 1 y 3.
  - `awk -F',' ' $3 > 50 {print $0}'`: Filtrar filas por condición numérica (columna 3 > 50).
- **Agregación y Estadística (Bash puro):**
  - `sort | uniq -c`: Tabla de frecuencias (histograma de texto).
  - `sort -nr | head`: Top 10 valores más altos.
- **Git (Ciclo de vida):**
  - `git status` → `git add .` → `git commit -m "msg"`.
  - `git log --oneline --graph`: Ver la historia del árbol.

### Profundización: Estructuras de Proyecto (Cookiecutter)

La estructura que creaste (`data`, `scripts`, `reports`) no es arbitraria. Sigue el estándar *Cookiecutter Data Science*, usado en la industria para garantizar que los proyectos sean reproducibles.

- **data/raw**: Inmutable. Jamás edites un archivo aquí manualmente.
- **data/processed**: El resultado de limpiar `raw`. Si borras esta carpeta, deberías poder regenerarla corriendo tus scripts.
- **reports**: El conocimiento generado (gráficos, PDFs, tablas).

## El Puente a Python: Cuando Bash no es suficiente

Bash es excelente para *pipes* y limpieza rápida, pero para estadística compleja (promedios, desviaciones, gráficos), Python es mejor.

Mira lo fácil que es obtener un resumen estadístico completo con la librería **Pandas** (que veremos en profundidad la próxima semana):

```
# "One-liner" de Python invocado desde Bash
# Calcula promedio, min, max y cuartiles de TODAS las columnas numéricas
python3 -c "import pandas as pd; df = pd.read_csv('data/raw/datos_sensores.csv'); print(df.describe())"
```

### Preparación para Semana 2

Para la próxima clase, asegúrate de tener tu entorno Python listo.

1. Crea un entorno: `python -m venv ia_env`
2. Actívalo: `source ia_env/bin/activate` (Linux/Mac) o `ia_env Scripts activate` (Windows).
3. Instala Pandas: `pip install pandas numpy`

## Anexo: Alias de Productividad

Copia estos alias en tu archivo de configuración (`~/.bashrc`) para trabajar más rápido y seguro:

```
# Seguridad (pregunta antes de borrar/sobrescribir)
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'

# Listados útiles
alias ll='ls -lh'           # Lista detallada con tamaño humano
alias la='ls -lha'         # Incluye archivos ocultos

# Git rápido
alias gs='git status'
alias gl='git log --oneline --graph --all'
```

### Siguiente Nivel

En la **Semana 2**, dejaremos atrás el procesamiento de texto línea por línea y pasaremos a la **vectorización con NumPy**: cómo procesar matrices de millones de datos en milisegundos usando el poder matemático de Python.