Manual Avanzado de Pandas

Procesamiento de Datos para Agroindustria 4.0

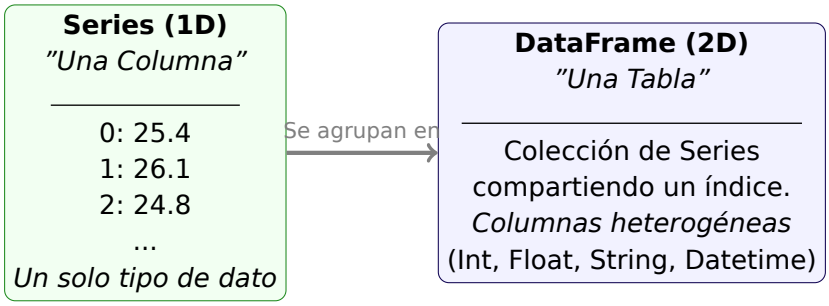
Curso de IA Aplicada al Agro

11 de enero de 2026

Índice general

Introducción: La Estructura de los Datos

Antes de limpiar datos, debemos entender cómo Pandas los organiza en la memoria RAM. A diferencia de Excel, donde todo es una celda visual, Pandas distingue rigurosamente entre tipos de estructuras.



1 Capítulo I: Ingesta de Datos (I/O)

El mundo real no vive solo en CSV. En agroindustria, lidiarás con Excel antiguos y JSON de APIs modernas.

1.1 Lectura Robusta

El comando `read_csv` tiene más de 50 parámetros. Aquí están los esenciales para evitar errores al cargar.

Listing 1: Carga Avanzada de Datos

```python
import pandas as pd

# 1. Cargar CSV definiendo fechas y nulos personalizados
df_sensor = pd.read_csv(
    'sensores_raw.csv',
    sep=';',                    # A veces usan punto y coma
    parse_dates=['fecha_hora'], # Convertir autom. a datetime
    na_values=['error', '-', 'null'], # Tratar estos textos como NaN
    dtype={'id_sensor': str}    # Forzar ID como texto, no numero
)

# 2. Cargar Excel (requiere libreria 'openpyxl')
df_suelos = pd.read_excel(
    'analisis_suelos.xlsx',
    sheet_name='Lote_Norte',
    header=1  # Usar la segunda fila como titulos
)
```

1.2 Inspección Inicial

Lo primero que haces al recibir un dataset:

```python
print(df.shape)   # (filas, columnas)
print(df.dtypes)  # ¿Los numeros son float o object (texto)?
print(df.head(3)) # Muestra visual rapida
print(df['cultivo'].value_counts()) # Conteo de categorias unicas
```

2 Capítulo II: Indexación y Selección Quirúrgica

Acceder a los datos incorrectamente es la fuente #1 de *bugs* silenciosos.

Método	¿Qué usa?	Ejemplo
`.loc[]`	**Etiquetas** (Nombres)	`df.loc["2024-01-01", "Humedad"]`
`.iloc[]`	**Posición** (Enteros)	`df.iloc[0, 3]`
`df[ ]`	**Columnas** (Principalmente)	`df["Temperatura"]`

2.1 Filtrado Booleano (Máscaras)

No uses `if` dentro de un `for`. Usa máscaras vectorizadas.

Listing 2: Consultas Complejas

```
# 1. Crear la mascara (devuelve Series de True/False)
mask_calor = df['Temperatura'] > 30
mask_seco  = df['Humedad'] < 40

# 2. Aplicar filtro
alerta_estres = df[mask_calor & mask_seco]

# 3. Filtrar por lista de valores permitidos
lotes_interes = ['Lote_A1', 'Lote_B5']
df_foco = df[df['id_lote'].isin(lotes_interes)]
```

3 Capítulo III: Limpieza Profunda (Data Cleaning)

Tipos de Datos Incorrectos

Si una columna de números tiene un solo texto (ej: ”Sin dato”), Pandas convertirá **toda** la columna a `object` (texto). No podrás hacer sumas ni promedios.

3.1 Sanitización de Tipos

```
# Convertir a numerico, forzando errores a NaN
# Si hay un texto "Error", se vuelve NaN en lugar de romper el codigo
df['ph_suelo'] = pd.to_numeric(df['ph_suelo'], errors='coerce')

# Eliminar duplicados (ej: sensor envio el dato 2 veces)
df = df.drop_duplicates(subset=['fecha', 'id_sensor'])
```

3.2 Tratamiento de Nulos (Imputación)

- `fillna(0)`: Peligroso en agro (0 de lluvia es válido, 0 de pH es ácido mortal).
- `ffill()`: Rellenar con el último valor conocido (ideal sensores).
- `interpolate()`: Trazar una línea entre puntos.

```
# Interpolacion lineal limitada a huecos pequeños
df['Humedad'] = df['Humedad'].interpolate(method='time', limit=2)
```

4 Capítulo IV: Ingeniería de Características

La IA no aprende sola; a veces debemos darle ”pistas” creando nuevas columnas.

4.1 Operaciones Vectorizadas

```
# Crear columna nueva basada en calculo matematico
# Diferencia de temperatura (Dia - Noche)
df['amplitud_termica'] = df['temp_max'] - df['temp_min']
```

4.2 Lógica Personalizada (.apply)

Cuando la matemática simple no basta, aplicamos una función a cada fila. **Nota:** Es más lento que vectorizar, úsalo con cuidado.

```
def clasificar_riesgo(fila):
    if fila['Humedad'] > 90 and fila['Temperatura'] > 25:
        return "ALTO (Hongos)"
    elif fila['Viento'] > 50:
        return "ALTO (Daño Fisico)"
    else:
        return "BAJO"

# axis=1 significa "pasar la fila completa a la funcion"
df['tipo_riesgo'] = df.apply(clasificar_riesgo, axis=1)
```

5 Capítulo V: Series Temporales y Resampling

El agro se mueve por ciclos (diarios, mensuales, estacionales).

Resampling vs Rolling

- **Resample:** Cambiar la frecuencia (de Hora $\rightarrow$ Día). Como un ”Zoom Out”.
- **Rolling:** Ventana deslizante. Suaviza el gráfico manteniendo la frecuencia.

Listing 3: Análisis Temporal

```
# Asegurar que el indice es Datetime
df = df.set_index('fecha')

# 1. RESAMPLE: Promedio Mensual ('M')
clima_mensual = df.resample('M').mean()

# 2. ROLLING: Media movil de 7 dias (Suavizar picos)
# Elimina el ruido de dias atipicos
df['temp_suavizada'] = df['Temperatura'].rolling(window=7).mean()
```

6 Capítulo VI: Fusión de Datos (Merge)

Rara vez tienes toda la info en una tabla.

- Tabla A: Cosecha (kilos, fecha, `id_lote`)

- Tabla B: Suelo (`id_lote`, `tipo_tierra`, ph)

```
# Unir tablas usando 'id_lote' como llave
# how='left' mantiene todas las cosechas, y pega info de suelo si existe
df_completo = pd.merge(
    df_cosecha,
    df_suelo,
    on='id_lote',
    how='left'
)
```

7 Capítulo VII: Ética y Documentación

La limpieza de datos es un proceso subjetivo que altera la realidad observada.

Checklist de Integridad de Datos

1. **Trazabilidad:** ¿Guardo el dataset `raw` original sin tocar?
2. **Transparencia:** ¿Documento cuántas filas eliminé y por qué?
3. **Sesgo de Imputación:** Al rellenar nulos con el promedio, ¿estoy ocultando un fallo sistemático de un sensor en una zona pobre?

Reto Profesional: Auditoría Climática

Se te entrega `estacion_falla.csv` (10,000 filas). Contiene datos horarios de 2023.

Problemas conocidos:

1. El sensor de lluvia se "traba" y repite el valor exacto por horas.
2. Hay temperaturas de $200°$C (error de voltaje).
3. Faltan días completos en Octubre.

Tu Misión:

```python
# 1. Cargar parseando fechas
df = pd.read_csv('estacion_falla.csv', parse_dates=['timestamp'], index_col='timestamp')

# 2. Filtrar Outliers (Temp > 50 es imposible)
df.loc[df['temp'] > 50, 'temp'] = pd.NA

# 3. Detectar "valores congelados" (Diff = 0 por mucho tiempo)
# shift(1) mueve la columna un paso abajo para comparar con la anterior
df['cambio_lluvia'] = df['lluvia'] - df['lluvia'].shift(1)

# 4. Resamplear a diario sumando lluvia y promediando temp
df_diario = df.resample('D').agg({
    'lluvia': 'sum',
    'temp': 'mean',
    'humedad': 'max'
})

# 5. Visualizar comparando Raw vs Limpio
import matplotlib.pyplot as plt
plt.figure(figsize=(12,6))
df['temp'].plot(alpha=0.3, label='Raw (Horario)', color='gray')
df_diario['temp'].plot(linewidth=2, label='Limpio (Diario)', color='blue')
plt.legend()
plt.title("Auditoria de Temperatura 2023")
plt.savefig("reporte_calidad.pdf")
```

8 Capítulo VIII: Taller de Ejercicios Prácticos

La única forma de aprender Pandas es escribiendo código. En este capítulo, resolveremos problemas reales de la agroindustria.

8.1 Nivel 1: Laboratorio Guiado (Producción Lechera)

En este ejercicio, crearemos un pequeño dataset manualmente para entender el flujo completo sin depender de archivos externos.

Objetivo

Analizar la calidad de leche de 3 vacas durante una semana y detectar anomalías en el pH (Rango normal: 6.6 - 6.8).

Listing 4: Simulación y Análisis

```python
import pandas as pd
import numpy as np

# 1. CREAR DATOS SIMULADOS
data = {
    'id_vaca': ['V01', 'V02', 'V01', 'V03', 'V02', 'V03'],
    'fecha': ['2024-01-01', '2024-01-01', '2024-01-02',
              '2024-01-01', '2024-01-02', '2024-01-02'],
    'litros': [20.5, 18.2, 21.0, 19.5, 18.0, 15.0],
    'ph': [6.7, 6.6, 6.2, 6.8, 6.9, 7.1] # Notar valores acidos/alcalinos
}

df_leche = pd.DataFrame(data)

# 2. LIMPIEZA: FILTRAR LECHE ACIDA (pH < 6.5)
# Esto se descarta para consumo humano
leche_mala = df_leche[df_leche['ph'] < 6.5]
print("--- ALERTA DE ACIDEZ ---")
print(leche_mala)

# 3. ANALISIS: TOTAL DE LITROS POR VACA (SOLO LECHE BUENA)
df_buena = df_leche[df_leche['ph'] >= 6.5]
reporte = df_buena.groupby('id_vaca')['litros'].sum()

print("\n--- REPORTE DE PAGO ---")
print(reporte)
```

8.2 Nivel 2: Ejercicios de Calistenia

Resuelve estos problemas cortos. Asume que tienes un DataFrame llamado `df_clima`.

- **Ejercicio A (Selección):** Crea un nuevo DataFrame llamado `df_lluvia` que solo contenga las columnas `"fecha"` y `"precipitacion"`.
- **Ejercicio B (Filtro Simple):** Filtra los días donde la `"temperatura"` fue mayor a 35 grados.
- **Ejercicio C (Corrección de Texto):** La columna `"zona"` tiene valores ” Norte” (con espacio extra). Usa `df['zona'].str.strip()` para arreglarlo.

- **Ejercicio D (Creación de Columnas):** Convierte la temperatura de Celsius a Fahrenheit en una nueva columna: $F = C \times 1.8 + 32$.

8.3 Nivel 3: El Desafío de Integración

Escenario Real

Tienes dos fuentes de datos no sincronizadas:

1. `ventas.csv`: Registro de sacos de café vendidos (con fechas).
2. `precio_internacional.csv`: Precio del café en bolsa (diario).

Problema: El archivo de ventas no tiene el precio. Debes cruzar los datos para saber cuánto dinero real ingresó.

Pasos para la solución:

```python
# PISTA PARA LA SOLUCION

# 1. Cargar ambos DataFrames
ventas = pd.read_csv('ventas.csv')
precios = pd.read_csv('precio_internacional.csv')

# 2. Asegurar que las fechas sean tipo 'datetime' en AMBOS
ventas['fecha'] = pd.to_datetime(ventas['fecha'])
precios['fecha'] = pd.to_datetime(precios['fecha'])

# 3. Realizar el MERGE (Cruce)
# Usamos 'left' para no perder ventas si falta algun precio
df_final = pd.merge(ventas, precios, on='fecha', how='left')

# 4. Calcular ingreso total
# Si faltan precios (NaN), rellenar con el promedio antes de calcular
precio_promedio = precios['valor'].mean()
df_final['valor'] = df_final['valor'].fillna(precio_promedio)

df_final['ingreso_total'] = df_final['kilos_vendidos'] * df_final['valor']
```

¿Por qué fallan los Merges?

El error más común es que la columna clave (`fecha`) tenga formatos distintos.

- Tabla A: ”2024-01-01” (String)
- Tabla B: 2024-01-01 (Timestamp)

Regla de Oro: Siempre normaliza tipos con `.info()` antes de unir tablas.