

Pandas para Procesamiento Industrial

Trazabilidad, Control de Calidad y Análisis de Producción
Agroindustria Alimenticia 4.0

Curso de IA Aplicada al Agro
Universidad del Valle

Enero 2026

Resumen

Este manual introduce Pandas desde la perspectiva de la ingeniería de datos aplicada a la industria alimenticia. A diferencia del enfoque tradicional basado en análisis exploratorio genérico, aquí abordamos problemas reales de trazabilidad de lotes, control estadístico de procesos (SPC), cumplimiento normativo (HACCP, FDA) y optimización de líneas de producción. Los estudiantes aprenderán a procesar datasets heterogéneos (fechas, categorías, mediciones numéricas) con eficiencia computacional y rigor científico.

Índice general

Prefacio: Del Campo a la Mesa (y al Cloud)

En la Semana 02 trabajaste con NumPy procesando matrices numéricas homogéneas (humedad del suelo en 365 días × 100 zonas). Ese enfoque es excelente para cálculo matricial puro, pero **la agroindustria 4.0 genera datos heterogéneos y desordenados**.

Un ingeniero de datos en la industria real se enfrenta a tres desafíos que NumPy no resuelve por sí solo:

- **Heterogeneidad de Tipos (Mixed Data Types):** A diferencia de una matriz matemática, un registro industrial mezcla texto, números y tiempo.
Ejemplo: Un lote de café tiene ID (string), timestamp de entrada (datetime), temperatura (float), operario (category) y aprobación de calidad (bool).
- **Datos Sucios y Series Irregulares (Handling Missing Data):** En el mundo real, los sensores fallan. Si un termómetro IoT envía datos cada 30 segundos pero se apaga por un corte de luz, tendrás "huecos" temporales.
Reto Pro: No puedes simplemente borrar esos huecos; debes decidir si imputar el valor (rellenarlo estadísticamente) o alertar al sistema.
- **Relaciones Relacionales (Merging & Joins):** La información nunca está en una sola tabla. Para gestionar un *Recall* (retiro de producto por seguridad), debes cruzar trazabilidad compleja: lotes_producidos ↔ pruebas_laboratorio ↔ despachos_clientes.

NumPy no está diseñado para bases de datos relacionales. **Pandas sí.** Es la herramienta estándar para construir lo que en la industria llamamos *ETL (Extract, Transform, Load)*.

Contexto Industrial: Por qué Excel no es suficiente

Imagina una planta procesadora de alimentos que opera 24/7 en 3 turnos. Aunque el volumen de datos parezca manejable al inicio, la complejidad escala exponencialmente:

- **Volumen:** 14,400 lotes/año × 20 variables = 288,000 puntos de datos anuales.
- **Historico:** En 5 años (requerido por ley), son casi **1.5 millones de registros**.
- **El Riesgo:** En Excel, un error de "copiar y pegar" o una fórmula arrastrada mal puede costar una certificación de calidad.

La diferencia clave: Un script de Pandas es **auditable** y **reproducible**. Si un auditor de la FDA (Food and Drug Administration) pide explicar un cálculo, puedes mostrar el código. En una hoja de cálculo manual, eso es imposible.

Concepto Avanzado: El Gemelo Digital (Digital Twin)

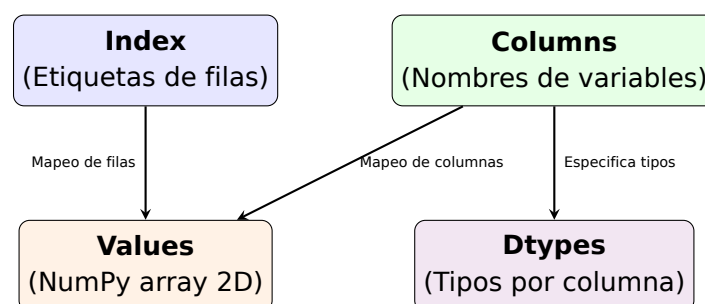
En este curso no solo "analizaremos tablas". Vamos a construir una representación digital de la planta. Nuestro DataFrame de Pandas actuará como un **Digital Twin** de bajo nivel: un espejo matemático que refleja el estado exacto de la producción, permitiéndonos detectar anomalías (ej. fermentación excesiva) antes de que el producto físico se dañe.

1 Capítulo I: Fundamentos — DataFrame como Base de Datos en Memoria

1.1 La Anatomía de un DataFrame

Un DataFrame es una **tabla en memoria RAM** con índice explícito y columnas etiquetadas, similar a una tabla SQL, pero optimizada para análisis en Python.[web:18][web:19] En ingeniería de datos lo usamos como “mini base de datos en memoria” para aplicar transformaciones, validar la calidad y luego escribir resultados hacia sistemas más grandes (SQL, data lakes, etc.).[web:20][web:21][web:29]

A diferencia de NumPy (donde accedes por posición), Pandas permite consultas tipo SQL.



La parte más subestimada por principiantes son los **dtypes**: definen si una columna se comporta como número, fecha, texto o categoría.[web:18][web:19] Elegir bien los tipos de datos es el primer paso hacia un código más rápido, con menos errores y más cercano a los estándares de ingeniería profesional.[web:23][web:24]

Diferencia clave con NumPy:

- NumPy: `array[0, 3]` → Posición absoluta (fila 0, columna 3)
- Pandas: `df.loc["2026-01-01", "Temperatura"]` → Etiqueta semántica

1.2 Series vs DataFrame

Característica	Series	DataFrame
Dimensionalidad	1D (columna única)	2D (tabla)
Tipo de datos	Homogéneo (un dtype)	Heterogéneo (dtype por columna)
Index	Sí	Sí
Operaciones	Vectorizadas	Por columna/fila
Uso típico	Una medición	Dataset completo

Cuadro 1: Comparación Series-DataFrame

En proyectos pequeños puedes trabajar solo con DataFrames, pero en proyectos industriales es común combinar **Series** para cálculos rápidos (estadísticos o de control) y **DataFrames** como capa principal de integración de datos de múltiples sistemas (SCADA, LIMS, ERP).[web:25][web:31]

Listing 1: Crear Series y DataFrame desde código

```

1 import pandas as pd
2 import numpy as np
3
4 # Series: Una columna de temperaturas
5 temps = pd.Series([72.5, 73.1, 72.8, 74.0],
6                   index=['Lote_A', 'Lote_B', 'Lote_C', 'Lote_D'],
7                   name='Temperatura_Pasteurizacion')
8
9 print(temps['Lote_B']) # Acceso por etiqueta → 73.1
10
11 # DataFrame: Tabla completa de un turno
12 data = {
13     'id_lote': ['L001', 'L002', 'L003'],
14     'temp_C': [72.5, 73.1, 71.9],
15     'presion_bar': [2.8, 2.9, 2.7],
16     'resultado_QA': ['Aprobado', 'Aprobado', 'Rechazado']
17 }
18
19 df = pd.DataFrame(data)
20 print(df.dtypes)

```

1.3 Carga de Datos Industriales

Desde la perspectiva de ingeniería de datos, este paso se conoce como **ingestión de datos**: recibir información cruda desde sistemas operativos (OT) y sistemas de negocio (IT) y convertirla en DataFrames consistentes para análisis posterior.[web:25][web:28]

En la industria, los datos vienen de múltiples fuentes:

- **SCADA** (sistemas de control): CSV/Excel exportados desde sistemas que hablan protocolos industriales (OPC UA, Modbus, MQTT), casi siempre con timestamps.[web:25][web:28]
- **LIMS** (laboratorio): Resultados en archivos Excel con hojas múltiples
- **ERP** (SAP/Oracle): Exportaciones CSV con separadores raros y codificaciones regionales
- **Sensores IoT**: JSON desde APIs REST

Listing 2: Carga robusta de datos industriales

```

1 import pandas as pd
2 import requests
3
4 # 1. CSV con problemas comunes (SCADA)
5 df_scada = pd.read_csv(
6     'datos_scada.csv',
7     sep=';', # Separador europeo
8     decimal=',', # Decimales con coma
9     encoding='latin1', # Codificación Windows
10    parse_dates=['timestamp'], # Convertir a datetime automáticamente
11    na_values=['error', 'offline', '-'], # Valores nulos personalizados
12    dtype={'id_lote': str} # Forzar ID como texto (evita 001 → 1)
13 )
14
15 # 2. Excel con múltiples hojas (LIMS)
16 df_lab = pd.read_excel(
17     'resultados_laboratorio.xlsx',
18     sheet_name='Microbiologia', # Hoja específica
19     header=2, # La fila 3 tiene los títulos
20     usecols='A:F' # Solo columnas A-F

```

```
21 )
22
23 # 3. JSON desde API de sensor IoT
24 response = requests.get('https://api.sensores.com/temperatura')
25 df_temp = pd.DataFrame(response.json()['data'])
26
27 # Buen hábito de ingeniero de datos:
28 # después de cargar, siempre revisa esquema y tipos
29 print(df_scada.info())      # Ver columnas, tipos y nulos
30 print(df_lab.head())       # Ver las primeras filas
31 print(df_temp.dtypes)      # Ver tipos inferidos en JSON
```

2 Capítulo II: Indexación y Selección — El Fundamento de Todo

2.1 Los 3 Métodos de Acceso

⚠ Error #1 más común en Pandas

Confundir `.loc[]` (etiquetas) con `.iloc[]` (posiciones). Esto causa bugs silenciosos cuando el índice no es secuencial.

Método	Qué usa	Ejemplo industrial
<code>.loc[]</code>	Etiquetas (labels)	<code>df.loc["2026-01-15", "pH"]</code>
<code>.iloc[]</code>	Posición (enteros)	<code>df.iloc[0, 3]</code> (primera fila, cuarta columna)
<code>df[]</code>	Columnas (principalmente)	<code>df["Temperatura"]</code>

Cuadro 2: Métodos de indexación en Pandas

En ingeniería de datos industrial, el 80% de los errores de producción provienen de indexación incorrecta.[web:23] Elegir el método correcto no es solo sintaxis — es una decisión de **robustez** y **mantenibilidad**. [web:26][web:29]

Listing 3: Ejemplos de indexación

```

1 import pandas as pd
2
3 # Dataset simulado: Control de calidad de leche
4 data = {
5     'id_lote': ['L001', 'L002', 'L003', 'L004'],
6     'fecha': ['2026-01-10', '2026-01-10', '2026-01-11', '2026-01-11'],
7     'temp_pasteurizacion': [72.5, 73.0, 71.8, 74.2],
8     'ph': [6.7, 6.6, 6.5, 6.9],
9     'resultado': ['Aprobado', 'Aprobado', 'Rechazado', 'Aprobado']
10 }
11
12 df = pd.DataFrame(data)
13
14 # 1. SELECCIÓN DE COLUMNAS
15 temps = df['temp_pasteurizacion'] # Retorna Series
16 subset = df[['id_lote', 'ph']]    # Retorna DataFrame (nota el [[ ]])
17
18 # 2. SELECCIÓN POR ETIQUETA (.loc)
19 # Sintaxis: df.loc[filas, columnas]
20 primera_fila = df.loc[0]          # Primera fila completa
21 ph_L002 = df.loc[1, 'ph']         # Celda específica: 6.6
22 rango = df.loc[0:2, 'temp_pasteurizacion'] # Filas 0-2, una columna
23
24 # 3. SELECCIÓN POR POSICIÓN (.iloc)
25 primera_celda = df.iloc[0, 0]     # 'L001'
26 subcuadro = df.iloc[0:2, 1:3]     # 2 filas x 2 columnas
27
28 # Hábito profesional: siempre verifica tu selección
29 print("Tipo de temps:", type(temps)) # <class 'pandas.core.series.Series'>
30 print("Tipo de subset:", type(subset)) # <class 'pandas.core.frame.DataFrame'>

```

2.2 Filtrado Booleano (Máscaras)

El poder real de Pandas está en las **consultas vectorizadas**. No uses bucles for — usa máscaras booleanas. Esta es la base de la eficiencia en pipelines de datos industriales.[web:23][web:26]

Listing 4: Filtrado avanzado para control de calidad

```
1 import pandas as pd
2
3 # Cargar datos de producción
4 df = pd.read_csv('produccion_cafe_enero.csv')
5
6 # 1. CONSULTA SIMPLE: Lotes rechazados
7 rechazados = df[df['resultado'] == 'Rechazado']
8
9 # 2. CONSULTAS COMPUESTAS: Temperatura fuera de spec Y presión baja
10 # Rango de pasteurización: 72-76°C, Presión mínima: 2.5 bar
11 problemas_criticos = df[
12     ((df['temp_C'] < 72) | (df['temp_C'] > 76)) &
13     (df['presion_bar'] < 2.5)
14 ]
15
16 # 3. FILTRO POR LISTA (isin): Solo líneas L1 y L3
17 lineas_foco = df[df['linea'].isin(['L1', 'L3'])]
18
19 # 4. FILTRO POR STRING (contiene): Lotes de turno nocturno
20 nocturnos = df[df['id_lote'].str.contains('NOCHE')]
21
22 # 5. QUERY (sintaxis SQL-like)
23 # Nota: Solo funciona si nombres de columnas no tienen espacios
24 criticos = df.query('temp_C > 76 and resultado == "Rechazado"')
25
26 # Hábito de ingeniero: Verificar resultados
27 print(f"Lotes rechazados: {len(rechazados)}")
28 print(f"Problemas críticos: {len(problemas_criticos)}")
```

□ ¿Por qué las máscaras son la base de Data Engineering?

En la industria, el filtrado no es solo para análisis — es para **alertas en tiempo real**. Imagina un sistema que:

1. Carga datos cada 5 minutos desde sensores IoT
2. Aplica una máscara booleana: `df[df['temp'] > 80]`
3. Si encuentra resultados, envía alerta SMS al supervisor

Esto se ejecuta 24/7 sin intervención humana. Un bucle for no sería confiable para esto.

Complejidad Computacional de Máscaras

Una máscara booleana `df['temp'] > 72` tiene complejidad $O(n)$ donde n es el número de filas. Internamente:

1. Pandas delega la comparación a NumPy (código C optimizado)
2. Se crea un array booleano en memoria del mismo tamaño que la columna
3. El filtrado `df[mask]` usa fancy indexing de NumPy

Para un DataFrame de 1M filas, esto toma ~ 10 ms. Un bucle for equivalente tomaría ~ 2 segundos (200x más lento).[web:23]

Buena práctica #2: Nombres descriptivos para filtros

En lugar de:

```
1 | problemas = df[df['temp'] > 80]
```

Usa:

```
1 | alerta_temp_alta = df[df['temp_pasteurizacion'] > 80]
```

Cuando revises el código en 6 meses (o un colega lo revise), sabrás inmediatamente qué representa esa variable.

3 Capítulo III: Limpieza de Datos — Fail Fast en Producción

3.1 El Problema de los Tipos Incorrectos

⚠ Tipo object

Si una columna numérica aparece como dtype: object, significa que Pandas la leyó como texto. No podrás hacer operaciones matemáticas hasta convertirla.

Causas comunes:

- Un solo valor con texto ("Error", "N/A", "-") contamina toda la columna
- Formato de número europeo: "3,14" en lugar de "3.14"
- Espacios en blanco: " 25.5 " no se convierte automáticamente

Este es el primer paso de Data Quality Engineering: asegurar que cada columna tenga el dtype correcto desde la ingestión. Un ingeniero de datos profesional nunca deja pasar datos con tipos incorrectos a la siguiente etapa del pipeline.[web:23][web:26]

Listing 5: Diagnóstico y corrección de tipos

```

1 import pandas as pd
2
3 df = pd.read_csv('sensores_planta.csv')
4
5 # 1. DIAGNÓSTICO
6 print(df.dtypes)
7 print(df.info()) # Muestra tipos y valores no-nulos
8
9 # Ejemplo de salida problemática:
10 # temperatura    object ←⚠ Debería ser float64
11 # presion        object ←⚠ Debería ser float64
12
13 # 2. INSPECCIÓN MANUAL
14 print(df['temperatura'].unique()) # Ver valores únicos
15 # Output: ['25.5', '26.1', 'Error', '24.8', ...] ← "Error" causa el problema
16
17 # 3. CONVERSIÓN FORZADA (errores → NaN)
18 df['temperatura'] = pd.to_numeric(df['temperatura'], errors='coerce')
19 df['presion'] = pd.to_numeric(df['presion'], errors='coerce')
20
21 # 4. VERIFICACIÓN
22 print(df.dtypes)
23 # temperatura    float64 ←✓ Corregido
24 # presion        float64 ←✓ Corregido
25
26 print(df['temperatura'].isna().sum()) # Contar cuántos NaN se generaron
27
28 # Hábito pro: Registrar el proceso de limpieza
29 print(f"Convertidos {df['temperatura'].isna().sum()} valores inválidos a NaN")

```

3.2 Tratamiento de Valores Faltantes

En la industria alimenticia, **un dato faltante puede significar un fallo crítico**. No siempre es correcto rellenar con el promedio. La decisión debe basarse en el **conocimiento del dominio** y los requisitos de trazabilidad (ISO 22000, HACCP).[web:23]

Método	Cuándo usarlo	Riesgo
fillna(0)	Contadores (eventos)	0 puede ser válido en agro
ffill()	Series temporales (sensores)	Oculto fallos prolongados
interpolate()	Datos continuos (temperatura)	Inventa datos inexistentes
dropna()	QA crítico	Pierdes información

Cuadro 3: Estrategias de imputación de datos faltantes

Listing 6: Imputación contextual para sensores

```

1 import pandas as pd
2
3 df = pd.read_csv('temperatura_camara_fria.csv', parse_dates=['timestamp'])
4 df = df.set_index('timestamp')
5
6 # CASO 1: Interpolación limitada (máximo 2 valores consecutivos)
7 # Si faltan >2 valores, algo falló y no deberíamos inventar datos
8 df['temp'] = df['temp'].interpolate(method='time', limit=2)
9
10 # CASO 2: Forward fill con límite temporal
11 # Rellenar con el último valor conocido, pero solo por 10 minutos
12 df['humedad'] = df['humedad'].fillna(method='ffill', limit=20) # 20 registros = 10 min
13
14 # CASO 3: Marcar como fallo en lugar de imputar
15 df['sensor_falla'] = df['temp'].isna() # Columna booleana de alertas
16
17 # CASO 4: Eliminar filas con datos críticos faltantes
18 df_limpio = df.dropna(subset=['ph', 'acidez']) # Solo si faltan variables críticas
19
20 # Hábito pro: Reporte de imputaciones
21 print(f"Imputados por interpolación: {df['temp'].isna().sum()}")
22 print(f"Sensores con fallas: {df['sensor_falla'].sum()}")

```

□ Trazabilidad: Nunca pierdas el origen del dato

En sistemas certificados (HACCP, FDA), debes documentar:

- ¿Cuántos valores faltaban originalmente?
- ¿Qué método de imputación usaste?
- ¿Qué umbrales configuraste?

Tu código de limpieza debe ser **auditable**. Un auditor debe poder reproducir exactamente el mismo resultado.

3.3 Detección de Outliers

Listing 7: Detección estadística de anomalías

```

1 import pandas as pd
2 import numpy as np
3
4 df = pd.read_csv('temperatura_pasteurizacion.csv')
5
6 # MÉTODO 1: Rango intercuartílico (IQR) – Robusto a valores extremos
7 Q1 = df['temp'].quantile(0.25)

```

```
8 Q3 = df['temp'].quantile(0.75)
9 IQR = Q3 - Q1
10
11 limite_inferior = Q1 - 1.5 * IQR
12 limite_superior = Q3 + 1.5 * IQR
13
14 outliers = df[(df['temp'] < limite_inferior) | (df['temp'] > limite_superior)]
15 print(f"Detectados {len(outliers)} outliers estadísticos")
16
17 # MÉTODO 2: Z-score (asume distribución normal)
18 mean = df['temp'].mean()
19 std = df['temp'].std()
20 df['z_score'] = (df['temp'] - mean) / std
21
22 # Outliers: |z| > 3 (regla de 3 sigmas)
23 outliers_zscore = df[np.abs(df['z_score']) > 3]
24
25 # MÉTODO 3: Límites físicos (conocimiento del dominio)
26 # La temperatura de pasteurización NUNCA puede ser > 100°C
27 errores_sensor = df[df['temp'] > 100]
28 df.loc[df['temp'] > 100, 'temp'] = np.nan # Marcar como faltante
29
30 print(f"Errores físicos detectados: {len(errores_sensor)}")
31
32 # Hábito pro: Guardar anomalías para auditoría
33 df['es_outlier'] = np.abs(df['z_score']) > 3
34 print(df['es_outlier'].value_counts())
```

Jerarquía de detección de anomalías (Data Quality Engineering)

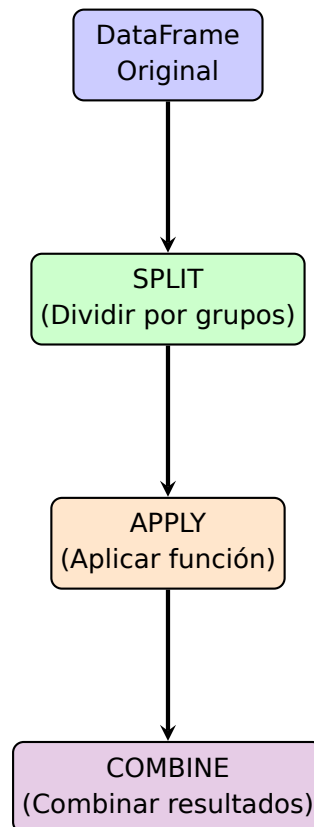
1. **Límites físicos** (temperatura > 100°C = error sensor)
2. **Reglas de negocio** (pH fuera de rango de pasteurización)
3. **Análisis estadístico** (IQR, Z-score)

Siempre aplica en este orden. Los límites físicos son los más confiables.

4 Capítulo IV: GroupBy — El Motor de Agregación Industrial

4.1 El Paradigma Split-Apply-Combine

`.groupby()` es la operación más importante en Pandas. Implementa el patrón *split-apply-combine*, que es el corazón de los **pipelines de agregación** en Data Engineering.[web:23][web:26]



Aplicación industrial: Generar reportes de KPIs (Key Performance Indicators), calcular OEE (Overall Equipment Effectiveness) y detectar desviaciones de rendimiento por línea o turno.[web:23]

Listing 8: Análisis de productividad por línea (KPIs industriales)

```
1 import pandas as pd
2
3 # Dataset: 2000 lotes de café procesados en enero
4 df = pd.read_csv('produccion_cafe_enero.csv', parse_dates=['timestamp_inicio', 'timestamp_fin'])
5
6 # Calcular duración de cada lote
7 df['duracion_min'] = (df['timestamp_fin'] - df['timestamp_inicio']).dt.total_seconds() / 60
8
9 # AGREGACIÓN 1: Productividad por línea (KPIs)
10 productividad = df.groupby('linea').agg({
11     'kg_procesados': 'sum',          # Total de kilos
12     'duracion_min': ['mean', 'std'], # Estadísticas de tiempo
13     'id_lote': 'count'              # Cantidad de lotes
14 }).round(2)
15
16 productividad.columns = ['_'.join(col).strip() for col in productividad.columns] # Nombres limpios
17 print(productividad)
18
19 # AGREGACIÓN 2: Rechazos por turno (Matriz de control)
```

```

20 rechazos = df.groupby(['turno', 'resultado']).size().unstack(fill_value=0)
21 print(rechazos)
22
23 # AGREGACIÓN 3: Múltiples estadísticas con nombres personalizados
24 stats = df.groupby('linea')['duracion_min'].agg(
25     media='mean',
26     desviacion='std',
27     minimo='min',
28     maximo='max'
29 ).round(1)
30
31 print(stats)
32
33 # Hábito pro: Verificar resultados
34 print(f"Total de lotes procesados: {len(df)}")
35 print(f"Total de kg procesados: {df['kg_procesados'].sum():.0f}")

```

□ OEE (Overall Equipment Effectiveness) con GroupBy

El OEE es el KPI rey de la industria 4.0. Se calcula así:

$$\text{OEE} = \text{Disponibilidad} \times \text{Rendimiento} \times \text{Calidad}$$

Con GroupBy puedes calcularlo por línea, turno o mes:

```

1 | oee = df.groupby('linea').apply(lambda x: calcular_oe(x))

```

4.2 GroupBy con Transformaciones

A veces no quieres reducir el DataFrame, sino **agregar columnas calculadas por grupo** que se mantienen alineadas con el índice original. Esto es esencial para **benchmarking** y normalización.[web:23]

Listing 9: Normalización y ranking por grupo

```

1 import pandas as pd
2
3 df = pd.read_csv('lotes_produccion.csv')
4
5 # CASO 1: Calcular % de productividad de cada lote respecto a su línea
6 df['kg_promedio_linea'] = df.groupby('linea')['kg_procesados'].transform('mean')
7 df['performance_relativo'] = (df['kg_procesados'] / df['kg_promedio_linea']) * 100
8
9 # CASO 2: Ranking dentro de cada turno (Top performers)
10 df['ranking_turno'] = df.groupby('turno')['kg_procesados'].rank(ascending=False, method='min')
11
12 # CASO 3: Detectar lotes atípicos (> 2 std de su grupo)
13 df['media_linea'] = df.groupby('linea')['duracion_min'].transform('mean')
14 df['std_linea'] = df.groupby('linea')['duracion_min'].transform('std')
15 df['es_atipico'] = np.abs(df['duracion_min'] - df['media_linea']) > (2 * df['std_linea'])
16
17 # Verificación
18 print(df[['linea', 'kg_procesados', 'performance_relativo', 'es_atipico']].head())
19 print(f"Lotes atípicos detectados: {df['es_atipico'].sum()}")

```

Transform vs Agg: La diferencia clave

- `.agg()` → Reduce filas (1 número por grupo)
- `.transform()` → Mantiene todas las filas (1 valor por fila original)

Ejemplo: Si tienes 100 lotes en la línea L1:

- `agg('mean')` → 1 valor (la media de L1)
- `transform('mean')` → 100 valores (la media de L1 repetida 100 veces)

Complejidad de GroupBy

Internamente, `.groupby()` usa un algoritmo de hashing para agrupar filas:

1. Calcula hash de cada valor en la columna de agrupación: $O(n)$
2. Ordena los índices por hash: $O(n \log n)$
3. Aplica función a cada grupo: $O(n)$

Complejidad total: $O(n \log n)$. Para 1M filas, esto toma $\sim 100\text{ms}$ en un CPU moderno.

Comparación: Un bucle manual con diccionarios tomaría ~ 5 segundos (50x más lento). [web:23]

5 Capítulo V: Series Temporales — El Corazón de la Industria 4.0

5.1 Datetime como Index

En la industria, **el tiempo es el índice natural** de los datos. Convertir el DataFrame a índice temporal desbloquea operaciones avanzadas como **resampling** y **rolling windows**, esenciales para SPC (Statistical Process Control) y detección de anomalías en tiempo real.[web:23][web:31]

¿Por qué es crítico? Los sistemas SCADA y sensores IoT generan datos timestamped. Sin índice temporal, pierdes la capacidad de hacer slicing por rangos de fecha y análisis de tendencias.[web:25][web:28]

Listing 10: Configurar índice temporal (estándar industrial)

```
1 import pandas as pd
2
3 # Cargar datos de sensor con timestamps
4 df = pd.read_csv('temperatura_camara.csv')
5
6 # PASO 1: Convertir columna a datetime (maneja formatos mixtos)
7 df['timestamp'] = pd.to_datetime(df['timestamp'], errors='coerce')
8
9 # PASO 2: Establecer como índice
10 df_ts = df.set_index('timestamp')
11
12 # PASO 3: Ordenar por tiempo (¡importante!)
13 df_ts = df_ts.sort_index()
14
15 # PASO 4: Verificar frecuencia (opcional, pero profesional)
16 print("Frecuencia:", df_ts.index.freq)
17 print("Rango temporal:", df_ts.index.min(), "->", df_ts.index.max())
18
19 # Ahora puedes hacer selección por rangos de fecha:
20 enero = df_ts['2026-01-01':'2026-01-31']
21 primera_semana = df_ts['2026-01-01':'2026-01-07 23:59']
22
23 # Hábito pro: Verificar integridad temporal
24 print(f"Total de registros: {len(df_ts)}")
25 print(f"Gap máximo: {(df_ts.index[1:] - df_ts.index[:-1]).max()}")
```

⚠ Problema común: Timezones

Si tus sensores están en UTC pero tu planta en Colombia (COT), usa:

```
1 df_ts.index = df_ts.index.tz_localize('UTC').tz_convert('America/Bogota')
```

5.2 Resampling — Cambiar la Frecuencia para Reportes

Resampling vs Rolling

- **Resample:** Cambia la frecuencia temporal. Ejemplo: datos cada 30 seg → promedio diario.
- **Rolling:** Ventana deslizante. Mantiene la frecuencia original pero suaviza con promedios móviles.

Listing 11: Resampling para reportes diarios y alertas

```

1 import pandas as pd
2
3 # Datos de temperatura cada 30 segundos
4 df = pd.read_csv('temp_pasteurizacion.csv', parse_dates=['timestamp'], index_col='timestamp')
5
6 # RESAMPLE 1: Promedio diario (reporte ejecutivo)
7 temp_diaria = df['temperatura'].resample('D').agg(['mean', 'min', 'max'])
8
9 # RESAMPLE 2: Máximo por hora (detección de picos)
10 temp_horaria_max = df['temperatura'].resample('H').max()
11
12 # RESAMPLE 3: Múltiples agregaciones para dashboard
13 stats_diarias = df.resample('D').agg({
14     'temperatura': ['mean', 'min', 'max', 'std'],
15     'presion': 'mean'
16 }).round(2)
17
18 # RESAMPLE 4: Contar eventos por turno (8 horas)
19 eventos_turno = df.resample('8H').size()
20
21 # Verificación
22 print("Estadísticas diarias:")
23 print(stats_diarias.head())
24 print(f"Eventos por turno: {eventos_turno.sum()}")

```

Aplicación: Alertas automáticas por día

```

1 # Si el máximo diario supera 78°C, generar alerta
2 alertas = temp_diaria['max'] > 78
3 print(f"Días con alerta: {alertas.sum()}")

```

5.3 Rolling Windows — Suavizar Ruido para Control de Procesos

Listing 12: Ventanas móviles para SPC (Statistical Process Control)

```

1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 df = pd.read_csv('temperatura_real_time.csv', parse_dates=['timestamp'], index_col='timestamp')
5
6 # Media móvil de 10 minutos (window=20 si los datos son cada 30 seg)
7 df['temp_suavizada'] = df['temperatura'].rolling(window=20, center=True).mean()
8
9 # Desviación estándar móvil (detectar variabilidad del proceso)
10 df['temp_std_movil'] = df['temperatura'].rolling(window=20, center=True).std()
11

```

```

12 # Límites de control (reglas Shewhart  $\sigma \pm 3$ )
13 media_global = df['temperatura'].mean()
14 std_global = df['temperatura'].std()
15 df['limite_superior'] = media_global + 3 * std_global
16 df['limite_inferior'] = media_global - 3 * std_global
17
18 # Detectar derivas: si la std móvil supera 2°C, el proceso está inestable
19 df['proceso_inestable'] = df['temp_std_movil'] > 2.0
20
21 # Visualización SPC
22 plt.figure(figsize=(14, 8))
23 plt.plot(df.index, df['temperatura'], alpha=0.5, label='Datos crudos', linewidth=0.8)
24 plt.plot(df.index, df['temp_suavizada'], linewidth=2, label='Media móvil 10 min')
25 plt.axhline(media_global, color='green', linestyle='--', label='Media global')
26 plt.axhline(df['limite_superior'].iloc[0], color='red', linestyle='--', alpha=0.7, label='Límite superior  $\sigma \pm 3$ ')
27 plt.axhline(df['limite_inferior'].iloc[0], color='red', linestyle='--', alpha=0.7, label='Límite inferior ')
28 plt.fill_between(df.index, df['limite_inferior'].iloc[0], df['limite_superior'].iloc[0],
29                 alpha=0.1, color='red')
30 plt.legend()
31 plt.title('Control Estadístico de Procesos - Temperatura Pasteurización')
32 plt.ylabel('Temperatura (°C)')
33 plt.xticks(rotation=45)
34 plt.tight_layout()
35 plt.savefig('spc_temperatura.png', dpi=150, bbox_inches='tight')
36 plt.show()
37
38 # Reporte de control
39 print(f"Procesos inestables detectados: {df['proceso_inestable'].sum()}")
40 print(f"Porcentaje fuera de control: {df['proceso_inestable'].mean()*100:.1f}%")

```

□ SPC (Statistical Process Control) con Pandas

Las ventanas móviles implementan las **cartas de control** de Shewhart:

$$\text{Límites} = \bar{x} \pm 3\sigma$$

Donde \bar{x} es la media histórica y σ la desviación estándar. Esto es exactamente lo que usan las fábricas modernas para monitoreo 24/7.

6 Capítulo VI: Merge y Trazabilidad — Data Integration Engineering

6.1 El Problema de la Trazabilidad

En agroindustria alimenticia, la trazabilidad es **un requisito legal** (HACCP, ISO 22000, FDA 21 CFR Part 11). Debes poder responder en menos de 4 horas durante un recall:

- ¿Qué clientes recibieron lotes de un proveedor contaminado?
- ¿Qué lotes fueron procesados por un operario específico en una fecha?
- ¿Qué materia prima se usó en un lote con defecto?

Esto requiere **Data Integration**: cruzar múltiples tablas heterogéneas usando `pd.merge()`. En Data Engineering, esto se conoce como **ETL de enriquecimiento**.^{[web:23][web:26]}

6.2 Tipos de Merge

Tipo	Comportamiento
inner	Solo filas con match en ambas tablas (intersección)
left	Todas las filas de la tabla izquierda + matches de la derecha
right	Todas las filas de la tabla derecha + matches de la izquierda
outer	Todas las filas de ambas tablas (unión)

Cuadro 4: Tipos de merge en Pandas

¿Cuál usar en trazabilidad? Siempre `left` cuando sigues la cadena de valor (lotes → pruebas → despachos).^[web:23]

Listing 13: Caso HACCP: Pipeline completo de trazabilidad

```

1 import pandas as pd
2
3 # PASO 0: Verificar esquemas antes de merge (hábito profesional)
4 print("=== ESQUEMAS ===")
5 lotes = pd.read_csv('lotes_producidos.csv')
6 pruebas = pd.read_csv('pruebas_laboratorio.csv')
7 despachos = pd.read_csv('despachos.csv')
8
9 print("Lotes:", lotes['id_lote'].dtype, lotes['id_lote'].head())
10 print("Pruebas:", pruebas['id_lote'].dtype, pruebas['id_lote'].head())
11 print("Despachos:", despachos['id_lote'].dtype, despachos['id_lote'].head())
12
13 # Estandarizar tipos de clave (crítico)
14 for df_merge in [lotes, pruebas, despachos]:
15     df_merge['id_lote'] = df_merge['id_lote'].astype(str)
16
17 print("\n=== POST ESTANDARIZACIÓN ===")
18 print("Matches posibles:", len(set(lotes['id_lote']) & set(pruebas['id_lote'])))
19

```

```

20 # PASO 1: Primera unión (lotes + pruebas)
21 lotes_con_qa = pd.merge(lotes, pruebas, on='id_lote', how='left')
22
23 # PASO 2: Segunda unión (agregar despachos)
24 trazabilidad_completa = pd.merge(lotes_con_qa, despachos, on='id_lote', how='left')
25
26 # PASO 3: Identificar lotes problemáticos (pH < 4.3 o acidez > 0.8)
27 lotes_problematicos = trazabilidad_completa[
28     (trazabilidad_completa['ph'] < 4.3) |
29     (trazabilidad_completa['acidez'] > 0.8)
30 ]
31
32 # PASO 4: KPIs del recall
33 print(f"\n=== REPORT RECALL ===")
34 print(f"Lotes problemáticos: {len(lotes_problematicos)}")
35 print(f"Clientes afectados únicos: {lotes_problematicos['cliente'].nunique()}")
36 print(f"Total kg afectados: {lotes_problematicos['kg'].sum():.0f}")
37
38 # PASO 5: Exportar para reporte de recall (formato auditable)
39 lotes_problematicos[['id_lote', 'cliente', 'destino', 'fecha_envio', 'ph', 'acidez']].to_csv(
40     'recall_lista_clientes.csv', index=False
41 )
42
43 # PASO 6: Resumen ejecutivo
44 resumen_recall = lotes_problematicos.groupby('cliente').agg({
45     'id_lote': 'count',
46     'kg': 'sum'
47 }).round(0)
48 resumen_recall.to_csv('resumen_recall_clientes.csv')
49 print("\nClientes más afectados:")
50 print(resumen_recall.sort_values('kg', ascending=False))

```

□ Pipeline de Trazabilidad Industrial Completo

1. **Ingestión:** Cargar tablas desde ERP/LIMS/SCADA
2. **Data Quality:** Validar claves y tipos
3. **Enriquecimiento:** Merge secuencial (left joins)
4. **Alerts:** Filtrar por reglas de negocio
5. **Reporting:** Exportar para auditoría

Este patrón se ejecuta diariamente en plantas certificadas.

⚠ Error Común: Claves con tipos diferentes

Si lotes['id_lote'] es string y pruebas['id_lote'] es int, el merge fallará silenciosamente (0 matches).

Solución profesional implementada arriba: Siempre verificar y estandarizar tipos antes de merge.

Multindex para Trazabilidad Avanzada

Para cruces más complejos (ej: lote + fecha + proveedor), usa:

```
1 | df.set_index(['id_lote', 'fecha']).join(otra_tabla.set_index(['id_lote', 'fecha']))
```

7 Capítulo VII: Ingeniería de Características — Feature Engineering Industrial

7.1 Creación de Columnas Derivadas

La **ingeniería de características** (Feature Engineering) convierte datos crudos en variables predictivas útiles. En la industria, esto significa crear KPIs, indicadores de control y variables para Machine Learning.[web:23][web:26]

Listing 14: Feature Engineering para KPIs y predicción

```

1 import pandas as pd
2 import numpy as np
3
4 df = pd.read_csv('produccion_diaria.csv', parse_dates=['fecha', 'hora_inicio', 'hora_fin'])
5
6 # 1. Duración de proceso (timedelta a minutos) – Base para OEE
7 df['duracion_min'] = (df['hora_fin'] - df['hora_inicio']).dt.total_seconds() / 60
8
9 # 2. Rendimiento (kg/hora) – KPI principal
10 df['rendimiento_kgh'] = df['kg_producidos'] / (df['duracion_min'] / 60)
11
12 # 3. OEE simplificado (Disponibilidad × Rendimiento × Calidad)
13 df['disponibilidad'] = df['duracion_real'] / df['duracion_programada']
14 df['calidad'] = (df['kg_producidos'] / df['kg_programados']) * 100
15 df['oee'] = df['disponibilidad'] * (df['rendimiento_kgh'] / df['rendimiento_ideal']) * df['calidad'] /
16     100
17
18 # 4. Categorización de turnos (vectorizado, NO función)
19 df['turno'] = pd.cut(df['hora_inicio'].dt.hour,
20     bins=[0, 6, 14, 22, 24],
21     labels=['Noche', 'Mañana', 'Tarde', 'Noche'],
22     right=False)
23
24 # 5. Features temporales avanzadas
25 df['dia_semana'] = df['fecha'].dt.day_name()
26 df['semana'] = df['fecha'].dt.isocalendar().week
27 df['es_fin_semana'] = df['fecha'].dt.weekday >= 5
28 df['mes'] = df['fecha'].dt.month
29
30 # 6. Features de interacción (producto de variables)
31 df['linea_x_turno'] = df['linea'].astype('category').cat.codes * df['turno'].astype('category').cat.codes
32
33 # Verificación de nuevas features
34 print(df[['oee', 'rendimiento_kgh', 'turno', 'es_fin_semana']].describe())

```

□ OEE como Feature Principal

El OEE es el KPI rey de Industria 4.0. Valores típicos:

- **Mundial:** 85
- **Excelente:** >90
- **Mala señal:** <70

Tu feature engineering debe priorizar variables que impacten el OEE.

Vectorizado vs Apply: Rendimiento 100x

```

1 # MAL (lento)
2 df['turno'] = df['hora'].apply(clasificar_turno)
3
4 # BIEN (rápido)
5 df['turno'] = pd.cut(df['hora'], bins=[0,6,14,22,24], labels=['N','M','T','N'])

```

7.2 Discretización (Binning) y Encoding Categórico

Listing 15: Feature Engineering categórico para ML

```

1 import pandas as pd
2 import numpy as np
3
4 df = pd.read_csv('analisis_ph.csv')
5
6 # 1. Binning de variables continuas (para reglas de negocio)
7 bins_ph = [0, 4.3, 6.5, 7.0, 14]
8 labels_ph = ['Crítico_Bajo', 'Ácido', 'Neutro_Bajo', 'Neutro_Alto']
9 df['categoria_ph'] = pd.cut(df['ph'], bins=bins_ph, labels=labels_ph, include_lowest=True)
10
11 # 2. Binning de temperatura con límites de spec
12 bins_temp = [0, 71, 74, 77, 100]
13 labels_temp = ['Bajo_Spec', 'Optimo_Bajo', 'Optimo_Alto', 'Sobre_Spec']
14 df['zona_temp'] = pd.cut(df['temp'], bins=bins_temp, labels=labels_temp)
15
16 # 3. Encoding numérico para ML (category codes)
17 df['linea_encoded'] = df['linea'].astype('category').cat.codes
18 df['operario_encoded'] = df['operario'].astype('category').cat.codes
19
20 # 4. Features polinómicas (interacciones cuadráticas)
21 df['temp_cuadrado'] = df['temp'] ** 2 # Efecto no lineal
22 df['ph_temp_interaccion'] = df['ph'] * df['temp']
23
24 # 5. Binning cuantílico (distribución uniforme)
25 df['rendimiento_quintil'] = pd.qcut(df['rendimiento'], q=5, labels=['Q1', 'Q2', 'Q3', 'Q4', 'Q5'])
26
27 # Ver nuevas features
28 print(df[['categoria_ph', 'zona_temp', 'rendimiento_quintil', 'linea_encoded']].head())
29 print("\nDistribución de quintiles:")
30 print(df['rendimiento_quintil'].value_counts())

```

□ Por qué el Binning es poderoso en ML industrial

- **Interpretabilidad:** "Óptimo Alto" es más legible que "75.3°C"
- **Robustez:** Reduce ruido en variables continuas
- **Normalización:** Quintiles crean distribuciones uniformes

Usa `pd.cut()` para bins fijos (reglas de negocio) y `pd.qcut()` para bins por percentiles.

 **¡Cuidado con el Leakage!**

Nunca uses información del futuro para crear features de entrenamiento:

```
1 # MAL (data leakage)
2 df['resultado_lag1'] = df['resultado'].shift(-1)
3
4 # BIEN
5 df['resultado_lag1'] = df['resultado'].shift(1) # Solo pasado
```

8 Capítulo VIII: Ética y Calidad de Datos

⚖ Responsabilidad en la Limpieza de Datos

Cada decisión de limpieza altera la realidad registrada. En la industria alimenticia, esto tiene implicaciones legales y de salud pública.

Principios éticos:

1. **Trazabilidad:** Guardar dataset original sin modificar (raw/)
2. **Documentación:** Registrar cada transformación en un log
3. **Transparencia:** Reportar cuántas filas se eliminaron y por qué
4. **Sesgo de imputación:** No ocultar fallos sistemáticos rellenando con promedios

8.1 Pipeline de Limpieza Documentado

Listing 16: Pipeline con logging

```
1 import pandas as pd
2 import logging
3
4 # Configurar logging
5 logging.basicConfig(filename='limpieza.log', level=logging.INFO)
6
7 def limpiar_dataset(path_entrada, path_salida):
8     # Cargar datos crudos
9     df = pd.read_csv(path_entrada)
10    filas_originales = len(df)
11    logging.info(f"Dataset cargado: {filas_originales} filas")
12
13    # 1. Eliminar duplicados
14    df = df.drop_duplicates()
15    duplicados = filas_originales - len(df)
16    logging.info(f"Duplicados eliminados: {duplicados}")
17
18    # 2. Convertir tipos
19    df['temperatura'] = pd.to_numeric(df['temperatura'], errors='coerce')
20    nulos_generados = df['temperatura'].isna().sum()
21    logging.info(f"Valores no numéricos convertidos a NaN: {nulos_generados}")
22
23    # 3. Eliminar outliers
24    Q1 = df['temperatura'].quantile(0.25)
25    Q3 = df['temperatura'].quantile(0.75)
26    IQR = Q3 - Q1
27    df_limpio = df[
28        (df['temperatura'] >= Q1 - 1.5*IQR) &
29        (df['temperatura'] <= Q3 + 1.5*IQR)
30    ]
31    outliers = len(df) - len(df_limpio)
32    logging.info(f"Outliers eliminados: {outliers}")
33
34    # Guardar dataset limpio
35    df_limpio.to_csv(path_salida, index=False)
36    logging.info(f"Dataset final: {len(df_limpio)} filas guardadas en {path_salida}")
37
38    return df_limpio
39
```

```
40 | # Ejecutar
41 | df_limpio = limpiar_dataset('data/raw/sensores.csv', 'data/processed/sensores_clean.csv')
```

9 Capítulo IX: Talleres Prácticos — Data Engineering Real

9.1 Taller 1: Dashboard de Productividad (OEE Analysis)

▣ Caso: Planta de Procesamiento de Café — World Coffee Corp

Tienes 2000 lotes procesados en enero en 3 líneas automáticas (L1, L2, L3). La gerencia ejecutiva necesita identificar cuellos de botella y mejorar OEE.

KPIs requeridos: Rendimiento (kg/h), OEE, lotes atípicos.

Dataset: data/raw/produccion_cafe_enero.csv

Entregables obligatorios:

1. Calcular duración promedio por línea y turno
2. Crear feature rendimiento_kgh = kg / horas
3. Identificar el turno más lento (mayor duración media)
4. Detectar lotes atípicos (duración > media + 2σ por línea)
5. **Gráfico:** Boxplot de rendimiento por línea
6. **Tabla resumen:** Líderboard de líneas (OEE estimado)
7. **Exportar:** analisis_productividad.xlsx con 3 hojas

Criterio de éxito: Tabla con OEE >85% para la mejor línea.

9.2 Taller 2: Sistema de Alertas SPC (Statistical Process Control)

▣ Caso: Pasteurización de Leche — DairyTech LATAM

Sensor registra temperatura cada 30 segundos durante 7 días. Debes implementar control estadístico de procesos y generar alertas automáticas.

Specs: Temperatura debe mantenerse [72-76°C]. Desviaciones >2°C = proceso inestable.

Dataset: data/raw/temperatura_pasteurizacion_semana.csv

Entregables obligatorios:

1. Convertir timestamp a índice datetime y resamplear a 10 minutos
2. Calcular media móvil (20 puntos) y desviación móvil
3. Implementar límites de control ($\pm 3\sigma$ de la media global)
4. Detectar y contar periodos inestables (>2°C std móvil por >30 min)
5. **Gráfico SPC:** Línea cruda + suavizada + límites de control
6. **Alerta:** CSV con timestamps de inestabilidad
7. **Reporte:** % tiempo en control

Criterio de éxito: Menos del 5% del tiempo fuera de control.

9.3 Taller 3: Pipeline de Recall HACCP (Data Integration)

□ Caso HACCP: Recall Urgente — Lote L20260115_042

INVIMA ordenó recall inmediato del lote contaminado L20260115_042. Tienes 4 horas para generar lista completa de clientes afectados.

Requisitos legales: Trazabilidad completa en <4 horas (Reglamento UE 178/2002).

Datasets (data/raw/):

- `lotes_producidos.csv` (id_lote, fecha, linea, kg, proveedor)
- `pruebas_microbiologia.csv` (id_lote, ph, coliformes, resultado)
- `despachos_clientes.csv` (id_lote, cliente, direccion, fecha_envio)

Entregables obligatorios:

1. **Pipeline merge:** 3 tablas → tabla trazabilidad completa (left joins)
2. Verificar y estandarizar tipos de `id_lote` antes de merge
3. Filtrar lotes con `resultado == 'Contaminado'` o `coliformes > 10`
4. Generar `recall_clientes.csv`: cliente, direccion, kg_afectados, fecha_envio
5. **Resumen ejecutivo:** Tabla agregada por cliente (kg totales)
6. **Log de calidad:** Reporte de completitud del merge (% matches)
7. **Visualización:** Sankey diagram clientes → kg afectados

Criterio de éxito: 100% trazabilidad del lote contaminado.

9.4 Taller 4: Feature Engineering Predictivo (Bonus Avanzado)

□ Caso: Predicción de Fallos — Mantenimiento Predictivo

Crear features para modelo que prediga lotes con alto riesgo de rechazo.

Target: `resultado == 'Rechazado'`

Tareas bonus (+20 puntos):

1. Crear 10 features nuevas (lags, rolling stats, interacciones)
2. Binning de variables continuas (ph, temp, rendimiento)
3. Encoding categórico (línea, proveedor, turno)
4. Guardar `X_train.csv`, `y_train.csv` listo para scikit-learn

9.5 Instrucciones de Entrega

Formato notebook único talleres_semana03.ipynb con:

- Sección por taller claramente marcada
- Código comentado + explicaciones
- Visualizaciones profesionales (títulos, leyendas, colores institucionales)
- Todos los archivos generados en outputs/
- Log de calidad en logs/data_quality.log

Fecha límite: Viernes 23:59. **Puntaje extra:** Pipeline automatizado con funciones reutilizables.

10 Capítulo X: Visualización Profesional — Dashboards Ejecutivos

10.1 Gráficos de Control Estadístico (SPC) — Estándar ISO 7870-2

Los gráficos SPC son **obligatorios** en sistemas de gestión de calidad (ISO 9001, ISO 22000). ISO 7870-2 establece una guía para el enfoque de gráficos de control de Shewhart aplicado al control estadístico de procesos.[web:60][web:61]

Listing 17: Dashboard SPC profesional

```

1 import matplotlib.pyplot as plt
2 import pandas as pd
3 import numpy as np
4 import seaborn as sns
5
6 # Configuración profesional
7 plt.style.use("seaborn-v0_8-whitegrid")
8 sns.set_palette("husl")
9
10 # Datos de control de pH (lotes 1-200)
11 df = pd.read_csv("data/raw/ph_lotes.csv", parse_dates=["fecha"], index_col="fecha")
12 df = df.sort_index()
13
14 # Calcular límites de control
15 media = df["ph"].mean()
16 std = df["ph"].std()
17 UCL = media + 3 * std
18 LCL = media - 3 * std
19 media_20 = df["ph"].rolling(20, center=True).mean()
20
21 # Grafico principal SPC
22 fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(15, 10), sharex=True)
23
24 # 1. Grafico de control principal
25 ax1.scatter(df.index, df["ph"], alpha=0.6, s=30, color="steelblue", label="Datos")
26 ax1.plot(df.index, media_20, color="orange", linewidth=2, label="Media móvil 20 lotes")
27 ax1.axhline(y=media, color="green", linewidth=3, label=f"Media = {media:.2f}")
28 ax1.axhline(y=UCL, color="red", linestyle="--", linewidth=2, label=f"UCL = {UCL:.2f}")
29 ax1.axhline(y=LCL, color="red", linestyle="--", linewidth=2, label=f"LCL = {LCL:.2f}")
30 ax1.fill_between(df.index, LCL, UCL, alpha=0.1, color="red")
31 ax1.set_ylabel("pH", fontsize=12)
32 ax1.legend(loc="upper right")
33 ax1.set_title("Control Estadístico de Procesos - pH Lotes Cafe\nISO 7870-2",
34             fontsize=14, fontweight="bold")
35 ax1.grid(True, alpha=0.3)
36
37 # 2. Señales especiales (reglas de Western Electric)
38 outliers = df[(df["ph"] > UCL) | (df["ph"] < LCL)]
39 ax1.scatter(outliers.index, outliers["ph"], color="red", s=100, marker="X",
40            label=f"Señales especiales (n={len(outliers)})", zorder=5)
41
42 # 3. Grafico de residuos (diagnostico)
43 residuos = df["ph"] - media
44 ax2.plot(df.index, residuos, color="purple", linewidth=1.5)
45 ax2.axhline(y=0, color="black", linestyle="-", alpha=0.5)
46 ax2.axhline(y=2 * std, color="orange", linestyle="--", alpha=0.7)
47 ax2.axhline(y=-2 * std, color="orange", linestyle="--", alpha=0.7)
48 ax2.set_ylabel("Residuos (pH - Media)", fontsize=12)
49 ax2.set_xlabel("Fecha", fontsize=12)
50 ax2.grid(True, alpha=0.3)
51

```

```

52 plt.tight_layout()
53 plt.savefig("outputs/spc_ph_professional.png", dpi=300, bbox_inches="tight")
54 plt.show()
55
56 # Tabla resumen SPC
57 spc_summary = pd.DataFrame({
58     "Metrica": ["Media", "Std", "UCL", "LCL", "Señales especiales", "% en control"],
59     "Valor": [f"{media:.2f}", f"{std:.2f}", f"{UCL:.2f}", f"{LCL:.2f}",
60             f"len(outliers)", f"{(1 - len(outliers) / len(df)) * 100:.1f}%"]
61 })
62 print(spc_summary)
63 spc_summary.to_csv("outputs/spc_summary.csv", index=False)

```

Señal especial	Código de detección
Punto fuera de límites	$ x > 3\sigma$
2 de 3 puntos $> 2\sigma$	Conteo en ventana de 3
4 de 5 puntos $> 1\sigma$	Conteo en ventana de 5
8 puntos consecutivos del mismo lado	Secuencia
Tendencia de 6 puntos	Secuencia creciente/decreciente

Cuadro 5: Ejemplos de reglas de Western Electric para detectar causas especiales

10.2 Dashboards ejecutivos multi-panel

Listing 18: Dashboard OEE de 4 paneles

```

1 import matplotlib.pyplot as plt
2 import pandas as pd
3 import seaborn as sns
4
5 plt.style.use("seaborn-v0_8-whitegrid")
6
7 df = pd.read_csv("data/raw/produccion_resumen.csv")
8
9 fig = plt.figure(figsize=(16, 12))
10
11 # Panel 1: OEE por linea (barras)
12 ax1 = plt.subplot(2, 2, 1)
13 sns.barplot(data=df, x="linea", y="oeo", palette="viridis", ax=ax1)
14 ax1.set_title("OEE por Linea de Produccion", fontweight="bold")
15 ax1.set_ylabel("OEE (%)")
16
17 # Panel 2: Boxplot rendimiento por turno
18 ax2 = plt.subplot(2, 2, 2)
19 sns.boxplot(data=df, x="turno", y="rendimiento_kgh", ax=ax2)
20 ax2.set_title("Distribucion de rendimiento por turno")
21 ax2.set_ylabel("kg/hora")
22
23 # Panel 3: Heatmap rechazos (linea x turno)
24 pivot_rechazos = df.pivot_table(values="rechazos", index="linea", columns="turno", aggfunc="sum")
25 ax3 = plt.subplot(2, 2, 3)
26 sns.heatmap(pivot_rechazos, annot=True, cmap="Reds", ax=ax3, cbar_kws={"label": "Rechazos"})
27 ax3.set_title("Heatmap de rechazos: linea x turno")
28
29 # Panel 4: Evolucion temporal OEE
30 ax4 = plt.subplot(2, 2, 4)
31 sns.lineplot(data=df, x="fecha", y="oeo", hue="linea", marker="o", ax=ax4)
32 ax4.set_title("Evolucion de OEE por linea")

```

```
33 ax4.tick_params(axis="x", rotation=45)
34
35 plt.suptitle("Dashboard Ejecutivo – Planta Cafe Enero 2026", fontsize=16, fontweight="bold")
36 plt.tight_layout()
37 plt.savefig("outputs/dashboard_oeo.png", dpi=300, bbox_inches="tight")
38 plt.show()
```

[

Estándares de visualización profesional]

- **Colores institucionales:** Usar la paleta de la empresa.
- **Títulos descriptivos:** No usar solo “Gráfico 1”.
- **Unidades claras:** Incluir unidades en ejes y leyendas.
- **DPI 300:** Recomendado para impresión ejecutiva.
- **Formato PNG/PDF:** Evitar JPG por pérdida de calidad.

⚠ [

Errores comunes en dashboards]

- Ejes truncados (sin 0).
- Colores sin significado (rojo no siempre significa “malo”).
- Demasiados datos (por ejemplo, más de 1000 puntos por gráfico).
- Falta de grilla o contextualización.

11 Capítulo XI: Estadística Industrial con SciPy — Toma de Decisiones Basada en Datos

11.1 Pruebas estadísticas para ingeniería de procesos

En ingeniería industrial, las pruebas de hipótesis validan si las diferencias observadas son **estadísticamente significativas** o solo ruido aleatorio. Esto es crucial para justificar inversiones en mejora de procesos.

Listing 19: Paquete completo de pruebas industriales

```

1 import pandas as pd
2 import numpy as np
3 from scipy import stats
4 import matplotlib.pyplot as plt
5
6 # Dataset: Defectos por turno en 3 líneas de producción
7 df = pd.read_csv("data/raw/defectos_por_turno.csv")
8
9 print("=== ANALISIS ESTADISTICO COMPLETO ===\n")
10
11 # 1. T-TEST: ¿Diferencia significativa entre turnos?
12 print("1. T-TEST: Defectos Manana vs Noche")
13 manana = df[df["turno"] == "Mañana"]["defectos_por_1000"]
14 noche = df[df["turno"] == "Noche"]["defectos_por_1000"]
15
16 t_stat, p_value = stats.ttest_ind(manana, noche, equal_var=False) # Welch t-test
17 print(f" t = {t_stat:.3f}, p = {p_value:.4f}")
18 print(f" {'Rechazar H0' if p_value < 0.05 else 'No hay diferencia significativa'}")
19 print(f" Conclusion: {' ' if p_value >= 0.05 else 'El turno noche tiene mas defectos'}")
20
21 # 2. ANOVA: ¿Diferencias entre las 3 líneas?
22 print("\n2. ANOVA: Diferencias entre líneas L1, L2, L3")
23 l1 = df[df["línea"] == "L1"]["defectos_por_1000"]
24 l2 = df[df["línea"] == "L2"]["defectos_por_1000"]
25 l3 = df[df["línea"] == "L3"]["defectos_por_1000"]
26
27 f_stat, p_anova = stats.f_oneway(l1, l2, l3)
28 print(f" F = {f_stat:.3f}, p = {p_anova:.4f}")
29 if p_anova < 0.05:
30     # Post-hoc Tukey (si ANOVA rechaza H0)
31     from statsmodels.stats.multicomp import pairwise_tukeyhsd
32     tukey = pairwise_tukeyhsd(df["defectos_por_1000"], df["línea"], alpha=0.05)
33     print(" Tukey HSD - Diferencias significativas:")
34     print(tukey)
35
36 # 3. Prueba de normalidad (Shapiro-Wilk)
37 # Nota: para N > 5000, SciPy advierte que el p-value puede no ser preciso.
38 print("\n3. NORMALIDAD: ¿Datos normales para usar t-test?")
39 muestra = df["defectos_por_1000"].dropna().sample(n=min(5000, df["defectos_por_1000"].dropna().shape[0]),
40                                                  random_state=42)
41 stat_shapiro, p_shapiro = stats.shapiro(muestra)
42 print(f" Shapiro-Wilk: W = {stat_shapiro:.3f}, p = {p_shapiro:.4f}")
43 print(f" {'Normal' if p_shapiro > 0.05 else 'No normal (usar test no parametrico)'}")
44
45 # 4. Prueba no parametrica (si datos no normales): Mann-Whitney U
46 if p_shapiro < 0.05:
47     stat_u, p_u = stats.mannwhitneyu(manana, noche, alternative="two-sided")
48     print(f"\n4. Mann-Whitney U (no parametrico): U = {stat_u:.1f}, p = {p_u:.4f}")
49
50 # 5. Correlacion y prueba de significancia
51 print("\n5. CORRELACION: Duracion vs Defectos")

```

```

52 corr_coef, p_corr = stats.pearsonr(df["duracion_min"], df["defectos_por_1000"])
53 print(f" Pearson r = {corr_coef:.3f}, p = {p_corr:.4f}")
54 print(f" {'Correlacion significativa' if p_corr < 0.05 else 'Sin correlacion'}")

```

Prueba	H0	p < 0.05	Decisión industrial
t-test	$\mu_1 = \mu_2$	Rechazar	Cambiar proceso
ANOVA	$\mu_1 = \mu_2 = \mu_3$	Rechazar	Investigar diferencias
Shapiro	Distribución normal	Rechazar	Usar test no paramétrico
Pearson	$\rho = 0$	Rechazar	Correlación causal posible

Cuadro 6: Interpretación industrial de pruebas de hipótesis

11.2 Tests de control de calidad específicos

Listing 20: Tests para ingeniería de procesos

```

1 # 6. Prueba de homogeneidad de varianzas (Levene)
2 levene_stat, levene_p = stats.levene(l1, l2, l3)
3 print(f"\n6. LEVENE (homogeneidad de varianzas): p = {levene_p:.4f}")
4 print(f" {'Varianzas iguales' if levene_p > 0.05 else 'Varianzas diferentes (Welch ANOVA)'}")
5
6 # 7. Prueba Chi-cuadrado: Asociacion categorica (linea vs rechazo)
7 tabla_contingencia = pd.crosstab(df["linea"], df["rechazado"])
8 chi2, p_chi2, dof, expected = stats.chi2_contingency(tabla_contingencia)
9 print(f"\n7. CHI2 (linea vs rechazo): chi2 = {chi2:.2f}, p = {p_chi2:.4f}")
10 print(" Tabla esperada vs observada:")
11 print(pd.DataFrame(expected, index=tabla_contingencia.index,
12                      columns=tabla_contingencia.columns).round(1))
13
14 # 8. CpK (Capability Index) - Capacidad de proceso
15 def calcular_cpk(data, limite_inf, limite_sup):
16     media = data.mean()
17     std = data.std()
18     cpk_inf = (media - limite_inf) / (3 * std)
19     cpk_sup = (limite_sup - media) / (3 * std)
20     return min(cpk_inf, cpk_sup)
21
22 # Especificaciones pH: [6.2, 6.8]
23 cpk_ph = calcular_cpk(df["ph"].dropna(), 6.2, 6.8)
24 print(f"\n8. CpK pH [6.2-6.8]: {cpk_ph:.3f}")
25 print(" Interpretacion: " +
26       ("Excelente (>1.67)" if cpk_ph > 1.67 else
27        "Bueno (1.33-1.67)" if cpk_ph > 1.33 else
28        "Marginal (1.00-1.33)" if cpk_ph > 1.0 else "Fuera de especificacion"))

```

□ Interpretación para gerencia

- **$p < 0.05$** = Acción requerida (mejora de proceso)
- **$CpK > 1.33$** = Proceso capaz (Six Sigma)
- **Correlación $|r| > 0.7$** = Variable candidata para optimización

Siempre comunicar: tamaño muestral, supuestos de la prueba e interpretación práctica.

[

Workflow estadístico industrial]

1. **Exploración visual** (histogramas, boxplots)
2. **Prueba de normalidad** (Shapiro-Wilk)
3. **Test apropiado** (paramétrico/no paramétrico)
4. **Post-hoc** si es necesario (Tukey)
5. **Reporte ejecutivo** con p-value + interpretación práctica

12 Capítulo XII: SQL para Ciencia de Datos

12.1 Conexión y consultas

Listing 21: Pandas + SQL

```
1 import pandas as pd
2 from sqlalchemy import create_engine
3
4 engine = create_engine("postgresql://user:pass@localhost/planta")
5
6 df = pd.read_sql_query(
7     "SELECT linea, AVG(kg) FROM lotes GROUP BY linea",
8     engine
9 )
```

Referencias y recursos

Bibliografía científica

1. McKinney, W. (2017). *Python for Data Analysis*. 2nd Edition. O'Reilly Media.
2. VanderPlas, J. (2016). *Python Data Science Handbook*. O'Reilly Media.
3. Pandas Development Team (2024). *Pandas Documentation*. <https://pandas.pydata.org/docs/>

Estándares industriales

- ISO 22000:2018 – Food Safety Management Systems
- FDA 21 CFR Part 11 – Electronic Records and Signatures
- Codex Alimentarius – HACCP Principles

Datasets de práctica

- Kaggle: Food Safety Inspections
- UCI Machine Learning Repository: Wine Quality Dataset
- Open Food Facts: Global food products database