Diagnóstico de calidad de una base de datos real o simulada

Contexto: 🙌

Detectar problemas de calidad en bases de datos es el primer paso para garantizar decisiones confiables y procesos eficientes. Esta actividad permite aplicar criterios reales sobre un conjunto de datos para identificar errores frecuentes.

Consigna: ✍️

Analiza una base de datos proporcionada por el docente (o simulada) y detecta problemas de calidad según al menos 5 dimensiones (por ejemplo: completitud, unicidad, validez, consistencia y actualidad).

Tiempo 🕐**:** *35 minutos*

🧭 **Paso a paso:**

1. Revisa la base de datos entregada.
2. Elige 5 dimensiones de calidad a evaluar.
3. Define cómo vas a medir cada una (porcentaje de nulos, duplicados, valores fuera de rango, etc.).
4. Detecta los problemas presentes y cuantifica su impacto.
5. Documenta los hallazgos en un pequeño informe grupal con propuestas de mejora.

1. Elección de dimensiones de calidad a evaluar

Tomaremos estas 5 dimensiones:

1. **Completitud** → porcentaje de datos faltantes en cada campo.
2. **Unicidad** → registros duplicados.
3. **Validez** → si los valores cumplen reglas de negocio (ej. formato de email, rango de edad).
4. **Consistencia** → coherencia entre campos (ej. ciudad y código postal).
5. **Actualidad** → fecha de última actualización.

2. Definición de métricas para medir cada dimensión

- **Completitud**: (N° de valores no nulos / Total de valores) * 100
- **Unicidad**: (N° de registros únicos / Total de registros) * 100
- **Validez**: (N° de valores válidos / Total de valores) * 100
- **Consistencia**: (N° de registros consistentes / Total de registros) * 100
- **Actualidad**: porcentaje de registros actualizados dentro del período esperado.

3. Ejemplo de código Python para análisis

Este código te permite medir estas dimensiones usando pandas con una base en CSV, Excel o exportada desde SQL/MongoDB.

```
import pandas as pd

from datetime import datetime, timedelta

# Cargar datos

df = pd.read_csv("base_datos.csv")


# 1. Completitud

completitud = df.notnull().mean() * 100


# 2. Unicidad (ej. usando columna ID)

unicidad = (df['ID'].nunique() / len(df)) * 100


# 3. Validez (ej. email y rango de edad)

email_valido = df['Email'].str.contains(r'^[\w\.-]+@[\w\.-]+\.\w+$', na=False).mean() * 100

edad_valida = df['Edad'].between(18, 99).mean() * 100


# 4. Consistencia (ej. código postal coincide con ciudad)

# Esto requiere una tabla de referencia, aquí un ejemplo simplificado

df['consistente'] = df.apply(lambda x: x['Ciudad'] in codigos_ciudad.get(x['CodigoPostal'], []), axis=1)

consistencia = df['consistente'].mean() * 100


# 5. Actualidad (últimos 12 meses)

fecha_limite = datetime.now() - timedelta(days=365)
```

```
actualidad = (pd.to_datetime(df['FechaActualizacion']) > fecha_limite).mean() * 100


# Resultados

print("Completitud:\n", completitud)

print(f"Unicidad: {unicidad:.2f}%")

print(f"Validez Email: {email_valido:.2f}% | Edad: {edad_valida:.2f}%")

print(f"Consistencia: {consistencia:.2f}%")

print(f"Actualidad: {actualidad:.2f}%")
```

4. Informe con hallazgos y propuestas

Ejemplo de síntesis:

Dimensión	Problema Detectado	Métrica	Impacto	Propuesta de Mejora
Completitud	12% de correos faltantes	88% completitud	Medio	Implementar validación obligatoria en formularios
Unicidad	5% registros duplicados	95% unicidad	Alto	Crear reglas de deduplicación en la carga
Validez	15% emails inválidos	85% validez	Alto	Validar formato de email en captura
Consistencia	8% código postal no coincide	92% consistencia	Medio	Implementar catálogo de códigos postales
Actualidad	30% registros sin actualizar en 1 año	70% actualidad	Alto	Enviar recordatorios para actualizar datos