Instituto Tecnológico y de Estudios Superiores de Monterrey

Inteligencia Artificial Avanzada para la Ciencia de Datos

Data Preparation

Los concentrados v2

Daniel Queijeiro Albo - A01710441

Diego Alfaro Pinto - A01709971

Diego Isaac Fuentes Juvera - A01705506

Jesus Ramirez Delgado - A01274723

Mauricio Anguiano Juarez - A01703337

Luis Adrián Uribe Cruz - A01783129

Adaptaciones de CRISP-DM

Data selection

De todos los datos que recibimos sólo nos resultan útiles algunos, por lo que hacemos una selección de cuales datos incluir y excluir de nuestro dataset final. Adicionalmente, manipulamos el dataset para adecuarlo a nuestras necesidades, por ejemplo, combinar columnas de cada cuarto de la vaca para obtener el total.

Datos incluidos

1. Dataset 3: Patadas (patadas_180725.csv)
    a. Registros: 37 observaciones
    b. Columnas: 43 atributos
    c. Justificación: Este archivo es un registro detallado incluye el comportamiento, asociados a estrés. dolor o incomodidad que sienten las vacas dentro de la máquina de ordeño en el CAETEC, lo cual nos es relevante para evaluar el bienestar animal y asegurar la calidad en el proceso.
    d. Atributos clave incluidos:
        i. Número de patadas por extremidad (DI, DD, TI, TD)
        ii. Duración de extracción por cuarto
        iii. Desconexiones/Fallos por pezón
        iv. Puntaje AMD
        v. Intervalo entre ordeños
        vi. Éxito del ordeño

2. Dataset 2: Reporte (reporte_180725.csv)
    a. Registros: 34 observaciones
    b. Columnas: 22 atributos
    c. Justificación: Resume el estado actual y la actividad reciente de las vacas que se encuentran en producción. Nos muestra métricas de comportamiento y salud para poder así monitorear el bienestar de las vacas, a su vez la eficiencia del sistema de ordeño.
    d. Atributos clave incluidos:
        i. Número del animal
        ii. Grupo al que pertenece (VMS 3)
        iii. Estado de reproducción (Inseminada, Preñada, Abierta, Fresca)
        iv. Días en ordeño / Días preñada
        v. Pezones no encontrados en último ordeño
        vi. Media diaria de pasos por puerta (últimos 7 días)

vii. Media MDI (Índice de Desviación de Ordeño - últimos 3 días)
viii. Producción de leche acumulada por período
ix. Días desde eventos reproductivos (último celo, control de gestación, inseminación)
x. Días desde tratamientos (salud, vacunación, inseminación artificial temporizada)
xi. Días desde eventos reproductivos (aborto, parto, secado)

Datos excluidos

1. Inventario total:
    a. Este archivo que contiene información sobre la salud reproductiva de las vacas, ha sido excluido debido a que los datos capturados en este documento no coinciden con los registros de sesiones de ordeño.
2. Registro de sesiones de ordeño:
    a. Misc | Razón de la desviación: Esta columna ha sido excluida debido a una falta de datos del 99% por lo que no nos ayudaría para entrenar el modelo
3. Imagenes de vacas:
    a. Requieren un procesamiento de visión separado al objetivo actual del proyecto, del mismo modo que cuentan con una calidad variable, desalineación de cámaras, condiciones de iluminación inconsistentes y obstrucciones del lentes por lo que quedan fuera del proyecto.

Datos derivados

1. **Registros de Sesiones de ordeño (registros_sesiones_merged.csv)**
    a. **Origen**: 33 archivos individuales por vaca, fusionados
    b. **Registros**: 7,239 sesiones de ordeño
    c. **Columnas**: 34 atributos
    d. **Periodo**: Marzo - Julio 2025
    e. **Transformaciones**: Se integraron los archivos por vaca en un solo dataset y se aplicaron procesos de imputación para manejar valores vacíos, utilizando ventanas de entré más-menos 3 días por vaca junto con sus promedios locales. En el caso de sangre en la leche los valores faltantes se rellenaron con 0 al interpretarse como ausencia de detección
    f. **Justificación**: Estos archivos contienen información detallada de cada sesión de ordeño, con métricas de importancia sobre la producción, calidad de leche y comportamiento por cuarto mamario.
    g. Atributos clave incluidos
        i. ID de vaca: Identificación única del ejemplar

ii. timestamp: Fecha y hora de la sesión
iii. Producción por cuarto (DI, DD, TT, TD): Cantidad de leche en kg
iv. Flujo y promedio máximo: Velocidad de extracción por cuarto
v. Conductividad eléctrica: Característica de la leche que sirve cómo indicador de mastitis.
vi. Sangre en leche (partículas por millón ppm): Cantidad de partículas de sangre que hay en la leche, indica posibles lesiones y/o mastitis.
vii. Estados del ordeño: Patadas, incompleto, pezones no encontrados
viii. Destino de leche: Tanque, drenaje, divert

2. Total de Media de flujos (kg/min)

a. **Cálculo**: TI + TD + DI + DD (suma de los 4 cuartos)

```python
# Código de util/load_dataframes.py
mediaFlujosColumns = df.xs('Media de los flujos (kg/min)', axis=1, level=0)
df[('Media de los flujos (kg/min)', 'Total')] = mediaFlujosColumns.sum(axis=1)
```

a. **Relevancia**:
   i. La suma representa el flujo volumétrico total de leche que extrae el robot por minuto
   ii. Valores bajos (<2kg/min) pueden indicar obstrucciones o problemas de salud
   iii. Para optimizar los tiempos de ordeño nos interesa el flujo agregado por vaca, no el promedio
b. **Interpretación clínica**:
   i. < 1.5 kg/min: Flujo bajo → posible estrés, mastitis o problema de eyección
   ii. 1.5 - 2.5 kg/min: Rango normal para vacas Holstein
   iii. > 2.5 kg/min: Flujo excelente → óptima salud y eyección láctea
   iv. > 3.5 kg/min: Considerar problemas de esfínter o sobre-estimulación

3. Total de Conductividad:

a. **Cálculo**: TI + TD + DI + DD (suma de conductividades)

```python
# Código de util/load_dataframes.py
conductividadColumns = df.xs('Conductividad (mS / cm)', axis=1, level=0)
df[('Conductividad (mS / cm)', 'Total')] = conductividadColumns.sum(axis=1)
```

b. **Relevancia**:
   i. Indicador primario de mastitis subclínica
   ii. Valores >20 mS/cm sugieren infección
   iii. Cambios bruscos (>15% entre ordeños) son señales de alerta temprana
c. **Rasgos de interpretación**
   i. **16-22 mS/cm:** Normal - leche saludable
   ii. **22-24 mS/cm:** Sospechoso - monitorear de cerca
   iii. **24-28 mS/cm:** Probable mastitis subclínica - evaluar cuartos individuales
   iv. **> 28 mS/cm:** Mastitis clínica - intervención inmediata

4. Total de flujos máximos (kg/min)

a. Cálculo: TI + TD + DI + DD (suma de flujos máximos)

```python
# Código de util/load_dataframes.py
maxFlujosColumns = df.xs('Flujos maximos (kg/min)', axis=1, level=0)
df[('Flujos maximos (kg/min)', 'Total')] = maxFlujosColumns.sum(axis=1)
```

b. **Relevancia**:
   i. Refleja el pico de producción durante el ordeño
   ii. Útil para identificar potenciales productivos
   iii. Correlaciona con salud del sistema mamario
c. **Interpretación**
   i. < 2.5 kg/min total: Reflejo de eyección deficiente → posible estrés
   ii. 2.5 - 4.0 kg/min: Normal
   iii. 4.0 - 5.5 kg/min: Excelente reflejo de eyección
   iv. > 5.5 kg/min: Verificar calibración de sensores

5. Total de Producciones (kg)

a. **Cálculo**: TI + TD + DI + DD (suma de producciones por cuarto)

```python
# Código de util/load_dataframes.py
produccionesColumns = df.xs('Producciones (kg)', axis=1, level=0)
df[('Producciones (kg)', 'Total')] = produccionesColumns.sum(axis=1)
```

a. **Relevancia**:
    i. Es la métrica principal de productividad por ordeño que tenemos.
    ii. Permite estimar ingresos aproximados:
        1. Ingresos = Producción x Precio de Leche x Calidad
    iii. Es la base para análisis económicos y de eficiencia
    iv. La tendencia de producción ayuda a identificar el estado de salud y la etapa de lactancia.

b. Factores que afectan la producción

i. Fisiológicos: Días en lactancia y número de partos. La producción sigue típicamente una curva como la de Wood, esta curva es una referencia teórica estándar en producción lechera para describir curvas de lactancia, esta se caracteriza por tener un pico con un descenso gradual.
ii. Nutricionales: Aporte de energía, proteína y minerales en la dieta, el que las vacas del CAETEC tengan una dieta insuficiente o mal balanceada se refleja rápidamente en la producción del hato.
iii. Ambientales: Estrés térmico que van de temperaturas >25°C que suceden normalmente en meses de veran y a esto agregar el hacinamiento lo cual se refiere al tener a muchas vacas dentro del mismo establo que puede llegar a ser un espacio reducido, lo que genera competencia por recursos, que se traduce en estrés que normalmente genera una mejor producción de leche.
iv. Sanitarios: Mastitis, cojeras, enfermedades metabólicas

6. Total de Sangre (ppm)

a. **Cálculo**: TI + TD + DI + DD (suma de partículas por millón en la sangre)

```python
# Código de util/load_dataframes.py
sangreColumns = df.xs('Sangre (ppm)', axis=1, level=0)
df[('Sangre (ppm)', 'Total')] = sangreColumns.sum(axis=1)
```

a. **Relevancia**:
    i. Es el principal indicador de lesiones en la ubre de la vaca que nos indica si que contiene mastitis severa.
    ii. Cualquier valor > 0 partículas por millón de sangre en la leche, ya que afecta la calidad de la leche.

iii. Valores elevados pueden hacer que la leche ya no sea apta para la comercialización que se traduce en pérdidas de dinero para el CAETEC

b. Detección de sangre en la leche

i. Se relaciona con la concentración de hemoglobina en la sangre
ii. Unidad: ppm (partes por millón)

c. Umbrales de acción

i. 0 ppm: Normal - leche apta para consumo
ii. 1-5 ppm: Trazas - monitorear, puede ser aceptable
iii. 5-20 ppm: Leche rosa - separar y descartar
iv. > 20 ppm: Leche rojiza - alerta veterinaria inmediata

7. Fecha y Hora separadas

a. **Cálculo**: Realizamos un *split* del **timestamp** original para separar fecha y hora en columnas distintas .
b. **Formato Fecha**: datetime (YYYY-MM-DD)
c. **Formato Hora**: time (HH:MM:SS)
d. **Relevancia**:
   i. Permite análisis de patrones diarios.
   ii. Identifica horarios óptimos de ordeño.
   iii. Facilita agregaciones temporales.

8. Producción Suavizada (Lowess)

a. Cálculo:

```python
# Código de util/load_dataframes.py
from statsmodels.nonparametric.smoothers_lowess import lowess

# Suavizado LOWESS con frac=0.1
df[('Producciones (kg)', 'Suavizado')] = lowess(
    df['Producciones (kg)', 'Total'],
    df[('Fecha y hora de inicio','fecha')],
    frac=0.1
)[:,1]
```

b. Explicación:

i. Las mediciones que tenemos de producción son bastante dispares día con día, lo que hace LOWESS es que nos permite extraer la tendencia haciendo lo siguiente:
   1. Tomamos un punto específico en el tiempo.
   2. Consideramos los datos que tiene cercanos lo que llamamos una vecindad temporal

3. Con esta vecindad se ajusta una regresión local
4. Repetimos estos pasos para cada punto dentro de la serie
5. A partir de todas las regresiones que generamos localmente, construimos una curva suavizada que representa una tendencia real.

c. **Relevancia**:
   i. Reduce el ruido diario de mediciones
   ii. Permite observar tendencias reales de producción
   iii. Captura relaciones no lineales y es relativamente robusto a outliers debido a que trabaja localmente, con pesos que dependen de la distancia y puede hacer iteraciones robustas que bajan el peso de los puntos con residuos muy grandes
   iv. Detecta cambios graduales en salud/productividad
   v. Mejora capacidad predictiva del modelo

d. **Justificación de frac=0.1**
   i. frac < 0.1: El suavizado es insuficiente y permanece demasiado ruido diario
   ii. frac = 0.1:
      1. Logra un buen balance entre suavizado y preservación de la tendencia
      2. Permite ver de forma más precisa en qué parte del ciclo productivo se encuentra la vaca.
      3. Conserva variaciones semanales/diarias importantes
      4. Muestra la tendencia global sin perder detalles relevantes
   iii. frac > 0.15:
      1. Genera un sobre-suavizado severo
      2. Desaparecen los patrones semanales inclusive diarios
      3. Se pierden detalles, como lo pueden ser las caídas abruptas por mastitis
      4. Nos dificulta el ubicar el momento exacto dentro del ciclo de lactancia
   iv. Valor óptimo encontrado experimentalmente: frac=0.1 es el porcentaje del dataset que se toma en cuenta para hacer cada regresión lineal local, logrando el mejor balance.

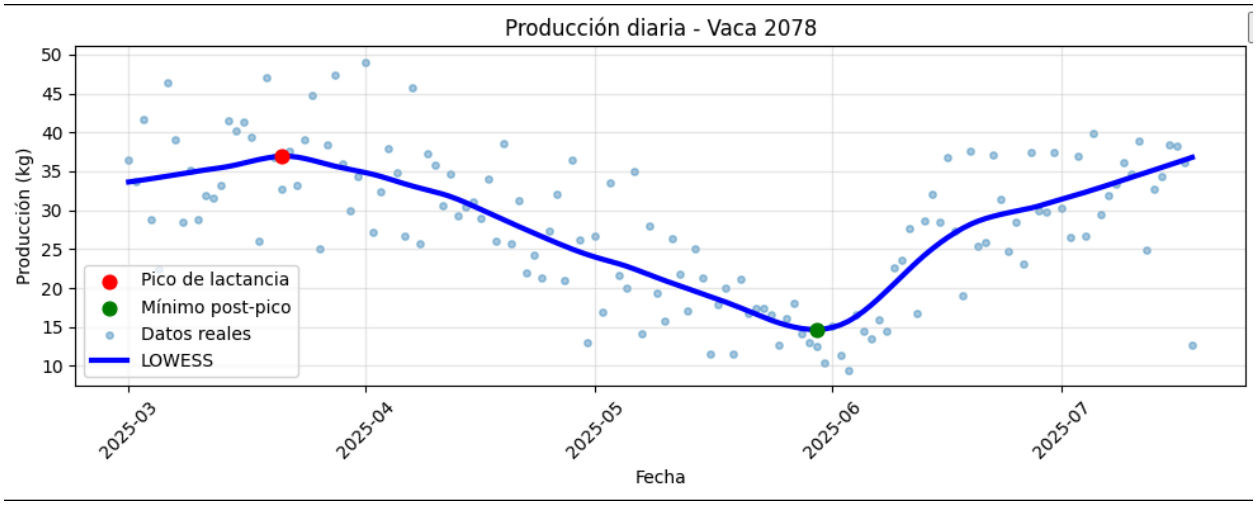
9. Tasa de decaimiento:

a. Cálculo:
   i. A partir de la curva de producción suavizada obtenida con LOWESS, se encuentra el punto máximo y mínimo del ciclo de producción para obtener la tasa de decaimiento, que indica que tan bien se mantiene la producción sobre el tiempo.

```python
peaks, _ = find_peaks(y, distance=10, prominence=0.5)
if len(peaks) > 0:
       peak_index = peaks[0]   # Primer máximo local
else:
       peak_index = y.argmax() # Si no encuentra picos, usar máximo global
```

```
peak_date = sub.iloc[peak_index]['fecha']
peak_value = sub.iloc[peak_index]['smooth']

# Detectar mínimos después del pico (inicio decaimiento)
y_post = y[peak_index:]
min_candidates, _ = find_peaks(-y_post, distance=5, prominence=0.5)

if len(min_candidates) > 0:
        min_index = peak_index + min_candidates[0]
else:
        min_index = y.argmin()  # Se usa mínimo global si no encuentra local

min_date = sub.iloc[min_index]['fecha']
min_value = sub.iloc[min_index]['smooth']

# Cálculo de pendiente de decaimiento (kg/día)
days = (min_date - peak_date).days if (min_date - peak_date).days != 0 else 1
slope = (min_value - peak_value) / days
```

b. Resultado



c. Relevancia

i. Permite obtener la consistencia en la producción de leche durante el ciclo de producción de la vaca.

ii. Muestra cuánto tiempo la vaca puede mantenerse produciendo una cantidad alta de leche.

10. Identificadores de Vaca (Extraído del nombre de archivo)

a. **Cálculo**: Regex extraction de nombres de CSV individuales
b. **Relevancia**:
    i. Permite el seguimiento individual de cada vaca
    ii. Es la clave para poder unir diferentes *datasets* (por ejemplo, sesiones de ordeño, reporte y patadas)
    iii. Es nuestra base para realizar análisis por vaca (curva de producción, comportamiento, etc)

11. Conteo de patadas

a. Cálculo:

i. Se hizo un conteo de la aparición de cada cuarto en la columna de patadas para cada vaca.
ii. Se calcularon medidas derivadas como patadas por día, por sesión de ordeño, y por hora dentro de la máquina para describir de mejor forma el comportamiento individual sin perjudicar a las vacas que están más presentes en los datos.

```python
datosPorVaca['PatadasPorDia'] = datosPorVaca['patadas_total'] /
datosPorVaca['DiasTotales']

datosPorVaca['PatadasPorOrdeño'] = datosPorVaca['patadas_total'] /
datosPorVaca['OrdeñosTotales']

datosPorVaca['PatadasPorHora'] = datosPorVaca['patadas_total'] /
(datosPorVaca['DuracionPromedio'].dt.total_seconds()*
datosPorVaca['OrdeñosTotales'] / 3600)
```

b. Relevancia

i. Permite saber el comportamiento de las vacas al usar el sistema.
ii. Obtener datos estadísticos que describan el comportamiento generalizado de cada vaca, sin perjudicar más a las vacas que han estado más días en el establo, o que hayan tenido más sesiones de ordeño.

12. Conteo de pezones no encontrados

```python
conteos['Total'] = conteos['DD']+ conteos['DI'] + conteos['TD'] + conteos['TI']
conteos = conteos.add_prefix('NoEncontrados')
```

Datos adicionales a obtener y/o solicitar

Hay dos principales conjuntos de datos que es necesario de poder conseguir:

1. Históricos: Conseguir todos los datos disponibles de las vacas del establo para poder alimentar al modelo. Además de los datos en sí mismos, poder ver la estructura que tienen para adaptarlo a la entrada del modelo y su preprocesamiento.
2. Etiquetas: Dado el enfoque de aprendizaje supervisado, es necesario poseer las etiquetas que califiquen a cada vaca, pues es la salida del modelo.
    a. En su defecto, obtener los criterios necesarios para poder realizar la etiquetación

Datos Imputados

Dentro de los registros de ordeño hay valores faltantes. En lugar de eliminar esas filas lo que significaría perder información que puede resultar valiosa, implementamos una estrategia de imputación de datos, la cuál rellena valores razonables utilizando el historial de cada vaca, seguimos el siguiente procedimiento:

1. Separación por tipo de registro:
    a. Se dividieron filas en dos grupos, de acuerdo con la columna (***'Main', 'Acción'***):
    b. Filas con ***'Ordeño'***, son las sesiones que sí se van usar en el análisis y por lo tanto que se imputan.
    c. Filas con otras acciones (por ejemplo ***'Rechazada'***) se dejan sin cambios y se vuelven a unir al final.

```python
# Separar filas de "Ordeño" y "Rechazada"
mask_ordeno = df_original[('Main', 'Accion')] == 'Ordeño'
df_ordeno = df_original[mask_ordeno].copy()
df_rechazada = df_original[~mask_ordeno].copy()
```

2. Ordenamiento por vaca y por fecha:
    a. Para las filas de ***'Ordeño'*** el dataframe se ordenó por:
        i. ID de la vaca
        ii. Fecha de inicio del ordeño
    b. Esto nos permite trabajar con una línea de tiempo más coherente para cada animal y aplicar ventanas de días hacía adelante y hacia atrás.

```python
# Ordenar por ID de vaca y fecha
df_ordeno = df_ordeno.sort_values([('ID', 'ID Vaca'), ('Fecha y hora de inicio', 'fecha')])
df_ordeno = df_ordeno.reset_index(drop=True)
```

3. Columnas en las que se imputan los datos:
    a. La imputación se aplicó únicamente a las variables más críticas para el análisis:
        i. Media de los flujos (kg/min) - TI, TD, DI, DD
        ii. Flujos máximos (kg/min) - TI, TD, DI, DD
        iii. Producciones (kg)
    b. Conductividad por cuarto
        i. Conductividad (mS/cm) - TI, TD, DI, DD
    c. Sangre por cuarto:
        i. Sangre (ppm) - TI, TD, DI, DD

```python
# Columnas de flujo y producción (12 columnas)
flow_production_columns = [
    ('Media de los flujos (kg/min)', 'TI'),
    ('Media de los flujos (kg/min)', 'TD'),
    ('Media de los flujos (kg/min)', 'DI'),
    ('Media de los flujos (kg/min)', 'DD'),
    ('Flujos maximos (kg/min)', 'TI'),
    ('Flujos maximos (kg/min)', 'TD'),
    ('Flujos maximos (kg/min)', 'DI'),
    ('Flujos maximos (kg/min)', 'DD'),
    ('Producciones (kg)', 'TI'),
    ('Producciones (kg)', 'TD'),
    ('Producciones (kg)', 'DI'),
    ('Producciones (kg)', 'DD')
```

```
]
# Columnas de conductividad (4 columnas)
conductivity_columns = [
    ('Conductividad (mS / cm)', 'TI'),
    ('Conductividad (mS / cm)', 'TD'),
    ('Conductividad (mS / cm)', 'DI'),
    ('Conductividad (mS / cm)', 'DD')
]
# Columnas de sangre (4 columnas)
blood_columns = [
    ('Sangre (ppm)', 'TI'),
    ('Sangre (ppm)', 'TD'),
    ('Sangre (ppm)', 'DI'),
    ('Sangre (ppm)', 'DD')
]
```

4. Tratamiento de ceros en flujos, producciones y conductividad
    a. En las filas de ***'Ordeño'***, los valores 0 en estas columnas (flujos, producciones y conductividad) se interpretaron como posibles "huecos" de medición y se reemplazaron con NaN. De esta manera, el sistema sabe que son candidatos a imputación y no simples ceros reales.

Python
```
# Reemplazar 0 con NaN en columnas de flujo/producción y conductividad
for col in flow_production_columns:
    if col in df_ordeno.columns:
        df_ordeno[col] = df_ordeno[col].replace(0, np.nan)

for col in conductivity_columns:
    if col in df_ordeno.columns:
        df_ordeno[col] = df_ordeno[col].replace(0, np.nan)
```

5. Imputación con ventana de entre más menos 3 días por vaca
    a. Para cada valor faltante en columnas de flujo, producción y conductividad:
        i. Se toma la vaca correspondiente y la fecha del ordeño con el dato faltante.

ii. Se busca dentro de una ventana de más-menos 3 días otras sesiones de ordeño de la misma vaca donde sí hay datos válidos en esa columna
iii. Si se encuentran registros en esa ventana, se calcula el promedio local y utilizamos para rellenar el hueco.

b. Con esto la primera prioridad es usar información de la misma vaca en fechas cercanas.

Python

```
def fill_with_window(df, col, window_days=3):

    """Rellena valores NaN usando el promedio de la vaca en una ventana de
±window_days días"""

    filled_values = df[col].copy()

    # Encontrar índices con valores NaN

    nan_indices = df[df[col].isna()].index

     for idx in nan_indices:

        cow_id = df.loc[idx, ('ID', 'ID Vaca')]

        date = df.loc[idx, ('Fecha y hora de inicio', 'fecha')]

         # Definir ventana de fechas (±3 días)

        date_min = date - pd.Timedelta(days=window_days)

        date_max = date + pd.Timedelta(days=window_days)

          # Filtrar datos de la misma vaca dentro de la ventana de tiempo

        mask = (

            (df[('ID', 'ID Vaca')] == cow_id) &

            (df[('Fecha y hora de inicio', 'fecha')] >= date_min) &

            (df[('Fecha y hora de inicio', 'fecha')] <= date_max) &

            (df[col].notna())  # Solo valores no-NaN

        )

        window_data = df.loc[mask, col]

        # Si hay datos en la ventana, usar el promedio
```

```
        if len(window_data) > 0:
            filled_values.loc[idx] = window_data.mean()
    return filled_values
```

6. Promedio por vaca y promedio global (último recurso)
    a. Si después de usar la ventana de más-menos 3 días todavía quedan valores faltantes:
        i. Primero se calcula el promedio de esa columna para la misma vaca (a lo largo de todas las sesiones de ordeño) se utiliza este valor
        ii. Si aún así quedan valores NaN (por ejemplo, vacas con muy pocos datos), se recurre al promedio global de la columna para todo el conjunto de vacas
    b. Con esto se logra rellenar todas las celdas, pero siempre intentando respetar primero el comportamiento individual de cada animal.

```python
for col in flow_production_columns:
    if col in df_ordeno.columns:
        # Paso 1: Rellenar con ventana de ±3 días
        df_ordeno[col] = fill_with_window(df_ordeno, col, window_days=3)

        # Paso 2: Si todavía hay NaN, usar el promedio general de la vaca
        cow_means = df_ordeno.groupby(('ID', 'ID
Vaca'))[[col]].transform('mean')
        df_ordeno[col] = df_ordeno[col].fillna(cow_means[col])

        # Paso 3: Si todavía hay NaN, rellenar con la media general de la
columna
        df_ordeno[col] = df_ordeno[col].fillna(df_ordeno[col].mean())
```

7. Imputación de sangre en leche (ppm)
    a. Para la sangre en leche, la lógica es distinta:
        i. Cuando una celda de sangre queda vacía, se rellena directamente con 0..
        ii. La interpretación de esto es que si el sistema no registró sangre, se asume que no hubo detección de sangre en la muestra de leche.

```python
# Rellenar Sangre (ppm) con 0 (solo filas de Ordeño)
for col in blood_columns:
    if col in df_ordeno.columns:
        df_ordeno[col] = df_ordeno[col].fillna(0)
```

8. Reconstrucción del dataset final
    a. Una vez rellenados los datos de las filas con “Ordeño”:
        i. Se vuelven unir las filas imputadas con las filas que no se modificaron
        ii. Se restaura el orden original de las filas usando el índice.
    b. El resultado es un dataset final con el mismo número de registros que el original, pero con muchos menos valores faltantes en las variables de interés, lo que facilita el análisis y la posterior etapa de modelado.

```python
# Combinar de nuevo las filas de Ordeño (rellenadas) con las de Rechazada (sin cambios)
df_final = pd.concat([df_ordeno, df_rechazada], ignore_index=False)

# Ordenar por el índice original para mantener el orden
df_final = df_final.sort_index()
df_final = df_final.reset_index(drop=True)
```