

Modelo 1: Transaction-Level Model (Enriquecimiento de Filas)

Descripción del Modelo

Este modelo opera a nivel de **fila individual** (cada transacción). No realiza agregaciones, sino que calcula campos derivados por cada registro. Corresponde a la etapa **FILTERS** del workflow.

Schema de Input

```
trans_id, fecha, producto, glosa, costo, total, cantidad,
inith, initm, customer_id, customer_name, customer_location
```

Schema de Output (Transaction Enriched)

Input original + columnas calculadas adicionales

Cálculos del Modelo

1. Campos Temporales Derivados

1.1. Componentes de Fecha

```
fecha_date = CAST(fecha AS DATE)
año = YEAR(fecha)
mes = MONTH(fecha)
dia = DAY(fecha)
dia_semana = DAYOFWEEK(fecha) -- 1=Domingo, 7=Sábado
nombre_dia_semana = DAYNAME(fecha)
semana_año = WEEK(fecha)
trimestre = QUARTER(fecha)
```

Usado en features: A7, B2, B7, C3, C8

1.2. Flags Temporales

```
es_fin_semana = CASE WHEN dia_semana IN (1, 7) THEN 1 ELSE 0 END
es_lunes = CASE WHEN dia_semana = 2 THEN 1 ELSE 0 END
-- agregar otros días según necesidad

-- Feriados (requiere tabla de calendario)
es_feriado = CASE WHEN fecha IN (SELECT fecha FROM calendario_feriados) THEN 1 EL
```

Usado en features: B2, C8

1.3. Período Relativo

```
-- Días desde inicio del dataset
dia_juliano = DATEDIFF(fecha, MIN(fecha) OVER())

-- Mes relativo (para análisis temporal)
mes_relativo = (año - MIN(año) OVER()) * 12 + mes

-- Timestamp unix (para ordenamiento)
timestamp = UNIX_TIMESTAMP(fecha)
```

Usado en features: A5, A7, C3

2. Campos Financieros Derivados

2.1. Margen por Transacción

```
margen_absoluto = total - costo
margen_porcentaje = ((total - costo) / total) * 100

-- Validación
margen_valido = CASE
    WHEN total > 0 AND costo >= 0 AND margen_porcentaje >= 0
    THEN 1
    ELSE 0
END
```

Usado en features: B1, C2

2.2. Precio Unitario

```
precio_unitario = total / cantidad
costo_unitario = costo / cantidad
margen_unitario = precio_unitario - costo_unitario
```

Usado en features: B1, A3

2.3. Categorización de Transacción

```
-- Tamaño de transacción
ticket_size_category = CASE
    WHEN total < 10000 THEN 'pequeño'
```

```

    WHEN total >= 10000 AND total < 50000 THEN 'medio'
    WHEN total >= 50000 THEN 'grande'
END

-- Volumen de cantidad
cantidad_category = CASE
    WHEN cantidad = 1 THEN 'unitario'
    WHEN cantidad BETWEEN 2 AND 5 THEN 'múltiple'
    WHEN cantidad > 5 THEN 'mayorista'
END

```

Usado en features: B3, análisis exploratorio

3. Campos de Identificación Enriquecidos

3.1. Normalizaciones

```

-- Normalización de IDs
customer_id_clean = TRIM(UPPER(customer_id))
producto_clean = TRIM(UPPER(producto))

-- Handle NULLs
customer_id_final = COALESCE(customer_id_clean, 'ANON_' || trans_id)

```

Usado en features: A4, B4

3.2. Categorización de Producto (si hay lógica)

```

-- Ejemplo: extraer categoría de nombre de producto
-- producto_categoria = SPLIT_PART(producto, '-', 1)
-- 0 mapear desde tabla externa

```

Usado en features: análisis categorial

4. Campos de Timing (de horarios si aplican)

4.1. Hora del Día

```

-- Si inith/initm son hora inicio
hora_transaccion = inith
minuto_transaccion = initm

-- Bloques de tiempo

```

```

bloque_tiempo = CASE
    WHEN hora_transaccion BETWEEN 6 AND 11 THEN 'mañana'
    WHEN hora_transaccion BETWEEN 12 AND 17 THEN 'tarde'
    WHEN hora_transaccion BETWEEN 18 AND 22 THEN 'noche'
    ELSE 'madrugada'
END

```

Usado en features: B2 (análisis intradiario)

5. Campos de Contexto de Cliente

5.1. Extracción de Ubicación

```

-- Si customer_location tiene estructura parseable
-- region = SPLIT_PART(customer_location, ',', 1)
-- ciudad = SPLIT_PART(customer_location, ',', 2)

-- 0 categorización simple
cliente_local = CASE
    WHEN customer_location LIKE '%Villarrica%' THEN 1
    ELSE 0
END

```

Usado en features: Análisis geográfico

Output Schema Completo

```

-- Campos originales
trans_id (string)
fecha (date)
producto (string)
glosa (string)
costo (integer)
total (integer)
cantidad (integer)
inith (integer)
initm (integer)
customer_id (string)
customer_name (string)
customer_location (string)

-- Campos temporales derivados
año (integer)
mes (integer)

```

```
dia (integer)
dia_semana (integer)
nombre_dia_semana (string)
semana_año (integer)
trimestre (integer)
es_fin_semana (boolean)
es_feriado (boolean)
dia_juliano (integer)
mes_relativo (integer)

-- Campos financieros derivados
margen_absoluto (integer)
margen_porcentaje (float)
precio_unitario (float)
costo_unitario (float)
margen_unitario (float)
ticket_size_category (string)
cantidad_category (string)

-- Campos normalizados
customer_id_clean (string)
producto_clean (string)

-- Campos de timing
hora_transaccion (integer)
bloque_tiempo (string)
```

Implementación

En Pandas (Python):

```
import pandas as pd
import numpy as np

def enrich_transactions(df):
    """
    Enriquece DataFrame de transacciones con campos derivados
    """
    # Convertir fecha a datetime
    df['fecha'] = pd.to_datetime(df['fecha'])

    # Componentes temporales
    df['año'] = df['fecha'].dt.year
    df['mes'] = df['fecha'].dt.month
    df['dia'] = df['fecha'].dt.day
```

```

df['dia_semana'] = df['fecha'].dt.dayofweek + 1 # 1=Lunes, 7=Domingo
df['nombre_dia_semana'] = df['fecha'].dt.day_name()
df['semana_año'] = df['fecha'].dt.isocalendar().week
df['trimestre'] = df['fecha'].dt.quarter

# Flags temporales
df['es_fin_semana'] = df['dia_semana'].isin([6, 7]).astype(int)

# Día juliano
min_fecha = df['fecha'].min()
df['dia_juliano'] = (df['fecha'] - min_fecha).dt.days

# Mes relativo
base_año = df['año'].min()
df['mes_relativo'] = (df['año'] - base_año) * 12 + df['mes']

# Campos financieros
df['margen_absoluto'] = df['total'] - df['costo']
df['margen_porcentaje'] = (df['margen_absoluto'] / df['total'] * 100).fillna(0)
df['precio_unitario'] = df['total'] / df['cantidad']
df['costo_unitario'] = df['costo'] / df['cantidad']
df['margen_unitario'] = df['precio_unitario'] - df['costo_unitario']

# Categorías de ticket
df['ticket_size_category'] = pd.cut(
    df['total'],
    bins=[0, 10000, 50000, np.inf],
    labels=['pequeño', 'medio', 'grande']
)

# Categorías de cantidad
df['cantidad_category'] = pd.cut(
    df['cantidad'],
    bins=[0, 1, 5, np.inf],
    labels=['unitario', 'múltiple', 'mayorista']
)

# Normalización
df['customer_id_clean'] = df['customer_id'].str.strip().str.upper()
df['producto_clean'] = df['producto'].str.strip().str.upper()

# Timing (si aplica)
if 'inith' in df.columns:
    df['hora_transaccion'] = df['inith']
    df['bloque_tiempo'] = pd.cut(
        df['hora_transaccion'],
        bins=[0, 6, 12, 18, 22, 24],

```

```

        labels=['madrugada', 'mañana', 'tarde', 'noche', 'madrugada'],
        ordered=False
    )

    return df

# Uso
df_enriched = enrich_transactions(df_raw)

```

Validaciones de Calidad

1. Validaciones de Integridad

```

# Validar que no hay valores negativos inválidos
assert (df_enriched['total'] >= 0).all(), "Total negativo detectado"
assert (df_enriched['costo'] >= 0).all(), "Costo negativo detectado"
assert (df_enriched['cantidad'] > 0).all(), "Cantidad cero o negativa detectada"

# Validar fechas razonables
assert df_enriched['fecha'].min() > pd.Timestamp('2000-01-01'), "Fechas muy antiguas"
assert df_enriched['fecha'].max() <= pd.Timestamp.now(), "Fechas futuras"

# Validar márgenes
assert (df_enriched['margen_porcentaje'] >= -100).all(), "Margen % inválido"

```

2. Reporte de Calidad

```

def quality_report(df):
    """Genera reporte de calidad de datos"""
    report = {
        'total_rows': len(df),
        'fecha_range': (df['fecha'].min(), df['fecha'].max()),
        'missing_customer_id': df['customer_id'].isna().sum(),
        'negative_margins': (df['margen_absoluto'] < 0).sum(),
        'zero_quantity': (df['cantidad'] == 0).sum(),
        'duplicated_trans_id': df['trans_id'].duplicated().sum(),
        'total_revenue': df['total'].sum(),
        'total_margin': df['margen_absoluto'].sum(),
        'avg_ticket': df['total'].mean(),
    }
    return report



```

Consideraciones de Performance

1. **Indexing:** Index en trans_id, fecha, producto, customer_id
 2. **Partitioning:** Particionar por año, mes para queries temporales
 3. **Caching:** Cachear campos temporales si dataset no cambia
 4. **Lazy evaluation:** En Spark, usar transformaciones lazy
-

Dependencies

Input Requirements:

-  CSV con schema especificado
-  Tabla de calendario de feriados (opcional para es_feriado)

External Libraries:

- pandas / PySpark
 - numpy
 - datetime
-

Notas de Implementación

- Este modelo se ejecuta **una vez** al inicio del pipeline
- Output se guarda como CSV/Parquet intermedio
- Todos los modelos posteriores usan este output como input
- Agregar campos derivados según surjan necesidades de nuevos features