

# Modelo 2: Product-Level Model (Agregación por Producto)

---

## Descripción del Modelo

Este modelo agrega datos a nivel de **producto**. Calcula métricas de desempeño, inventario, rentabilidad y comportamiento por cada producto único. Corresponde a la etapa **ATTRIBUTES** del workflow con GROUP BY producto.

## Input

Output del Modelo 1 (Transaction Enriched)

## Aggregation Key

```
GROUP BY producto_clean
```

## Features Soportados

- **A1** - Análisis Pareto (productos)
- **A2** - Alertas de Inventario
- **A3** - Salud de Inventario
- **A5** - Velocidad de Productos
- **B1** - Rentabilidad por Producto
- **B5** - Punto de Reorden

---

## Cálculos del Modelo

### 1. Métricas de Volumen y Frecuencia

#### 1.1. Volumen Total

```
-- Volumen de ventas
total_transacciones = COUNT(trans_id)
total_unidades_vendidas = SUM(cantidad)
total_ingresos = SUM(total)
total_costo = SUM(costo)

-- Período de análisis
primera_venta = MIN(fecha)
ultima_venta = MAX(fecha)
dias_activo = DATEDIFF(ultima_venta, primera_venta) + 1
```

**Python:**

```

product_metrics = df.groupby('producto_clean').agg({
    'trans_id': 'count',
    'cantidad': 'sum',
    'total': 'sum',
    'costo': 'sum',
    'fecha': ['min', 'max']
}).reset_index()

product_metrics.columns = [
    'producto', 'total_transacciones', 'total_unidades_vendidas',
    'total_ingresos', 'total_costo', 'primera_venta', 'ultima_venta'
]

product_metrics['dias_activo'] = (
    product_metrics['ultima_venta'] - product_metrics['primera_venta']
).dt.days + 1

```

## 1.2. Promedios

```

-- Promedios por transacción
avg_cantidad_por_transaccion = AVG(cantidad)
avg_precio_por_transaccion = AVG(total)

-- Promedios unitarios
avg_precio_unitario = SUM(total) / SUM(cantidad)
avg_costo_unitario = SUM(costo) / SUM(cantidad)

```

### Python:

```

product_metrics['avg_cantidad_por_transaccion'] = (
    product_metrics['total_unidades_vendidas'] /
    product_metrics['total_transacciones']
)

product_metrics['avg_precio_unitario'] = (
    product_metrics['total_ingresos'] /
    product_metrics['total_unidades_vendidas']
)

product_metrics['avg_costo_unitario'] = (
    product_metrics['total_costo'] /
    product_metrics['total_unidades_vendidas']
)

```

## 2. Métricas de Rentabilidad (Feature B1)

### 2.1. Margen Total y Promedio

```
margen_total = SUM(margen_absoluto)
margen_promedio_porcentaje = AVG(margen_porcentaje)
margen_total_porcentaje = (SUM(total) - SUM(costo)) / SUM(total) * 100
```

#### Python:

```
# Del DataFrame original
margin_metrics = df.groupby('producto_clean').agg({
    'margen_absoluto': 'sum',
    'margen_porcentaje': 'mean'
}).reset_index()

margin_metrics['margen_total_porcentaje'] = (
    (product_metrics['total_ingresos'] - product_metrics['total_costo']) /
    product_metrics['total_ingresos'] * 100
)

product_metrics = product_metrics.merge(margin_metrics, on='producto_clean')
```

### 2.2. Score de Rentabilidad

```
-- Rentabilidad compuesta: margen * volumen
rentabilidad_score = margen_total * LOG(total_unidades_vendidas + 1)

-- Categorización
rentabilidad_categoria = CASE
    WHEN margen_total_porcentaje >= 40 THEN 'alta'
    WHEN margen_total_porcentaje >= 20 THEN 'media'
    ELSE 'baja'
END
```

#### Python:

```
import numpy as np

product_metrics['rentabilidad_score'] = (
    product_metrics['margen_total'] *
    np.log1p(product_metrics['total_unidades_vendidas'])
)
```

```
product_metrics['rentabilidad_categoria'] = pd.cut(
    product_metrics['margen_total_porcentaje'],
    bins=[0, 20, 40, 100],
    labels=['baja', 'media', 'alta']
)
```

### 3. Métricas de Velocidad (Feature A5)

#### 3.1. Frecuencia de Venta

```
-- Ventas por período
ventas_por_dia = total_transacciones / dias_activo
ventas_por_semana = total_transacciones / (dias_activo / 7.0)
ventas_por_mes = total_transacciones / (dias_activo / 30.0)
```

#### Python:

```
product_metrics['ventas_por_dia'] = (
    product_metrics['total_transacciones'] /
    product_metrics['dias_activo']
)

product_metrics['ventas_por_semana'] = product_metrics['ventas_por_dia'] * 7
product_metrics['ventas_por_mes'] = product_metrics['ventas_por_dia'] * 30
```

#### 3.2. Días Entre Ventas

```
-- Requiere calcular diferencias entre fechas consecutivas por producto
-- En SQL con window functions:
WITH fecha_diffs AS (
    SELECT
        producto_clean,
        fecha,
        DATEDIFF(
            fecha,
            LAG(fecha) OVER (PARTITION BY producto_clean ORDER BY fecha)
        ) as dias_desde_venta_anterior
    FROM transactions_enriched
)
SELECT
    producto_clean,
    AVG(dias_desde_venta_anterior) as avg_dias_entre_ventas,
    STDDEV(dias_desde_venta_anterior) as stddev_dias_entre_ventas,
```

```

    MIN(dias_desde_venta_anterior) as min_dias_entre_ventas,
    MAX(dias_desde_venta_anterior) as max_dias_entre_ventas
FROM fecha_diffs
WHERE dias_desde_venta_anterior IS NOT NULL
GROUP BY producto_clean

```

#### Python:

```

def calculate_days_between_sales(df):
    """Calcula días entre ventas consecutivas por producto"""
    df_sorted = df.sort_values(['producto_clean', 'fecha'])
    df_sorted['dias_desde_anterior'] = df_sorted.groupby('producto_clean')['fecha']

    days_between = df_sorted.groupby('producto_clean')['dias_desde_anterior'].agg(
        ('avg_dias_entre_ventas', 'mean'),
        ('stddev_dias_entre_ventas', 'std'),
        ('min_dias_entre_ventas', 'min'),
        ('max_dias_entre_ventas', 'max')
    ).reset_index()

    return days_between

velocity_metrics = calculate_days_between_sales(df)
product_metrics = product_metrics.merge(velocity_metrics, on='producto_clean')

```

### 3.3. Categorización de Velocidad

```

velocidad_categoria = CASE
    WHEN avg_dias_entre_ventas < 7 THEN 'rápida'
    WHEN avg_dias_entre_ventas < 30 THEN 'media'
    ELSE 'lenta'
END

```

#### Python:

```

product_metrics['velocidad_categoria'] = pd.cut(
    product_metrics['avg_dias_entre_ventas'],
    bins=[0, 7, 30, np.inf],
    labels=['rápida', 'media', 'lenta']
)

```

## 4. Métricas de Inventario (Features A2, A3, B5)

## 4.1. Estado Actual

```
-- Días desde última venta (relativo a fecha de análisis)
dias_sin_venta = DATEDIFF(CURRENT_DATE, MAX(fecha))

-- Stock actual (si disponible del campo, sino inferir)
-- stock_actual = [requiere campo externo o lógica de negocio]
```

### Python:

```
import datetime

fecha_analisis = datetime.date.today()

product_metrics['dias_sin_venta'] = (
    fecha_analisis - pd.to_datetime(product_metrics['ultima_venta']).dt.date
).dt.days
```

## 4.2. Proyección de Stock (Feature A2)

```
-- Requiere stock_actual como input adicional
-- Si no hay stock field, necesita ingresarse manualmente o de otro sistema

dias_stock_restante = stock_actual / ventas_por_dia

-- Alerta
alerta_stock_bajo = CASE WHEN dias_stock_restante < 7 THEN 1 ELSE 0 END
alerta_sin_movimiento = CASE WHEN dias_sin_venta > 90 THEN 1 ELSE 0 END
```

### Python:

```
# Asumiendo que stock_actual viene de archivo externo
# stock_df = pd.read_csv('inventario_actual.csv')
# product_metrics = product_metrics.merge(stock_df, on='producto_clean', how='left')

# Si no hay stock, se puede hacer input manual o usar lógica de negocio
product_metrics['dias_stock_restante'] = (
    product_metrics.get('stock_actual', 0) /
    product_metrics['ventas_por_dia']
).fillna(999) # 999 = stock desconocido

product_metrics['alerta_stock_bajo'] = (
    product_metrics['dias_stock_restante'] < 7
).astype(int)
```

```
product_metrics['alerta_sin_movimiento'] = (
    product_metrics['dias_sin_venta'] > 90
).astype(int)
```

### 4.3. Rotación de Inventario (Feature A3)

```
-- Rotación = Ventas totales / Inventario promedio
-- Si no hay inventario promedio, usar aproximación con stock actual

rotacion_inventario = total_ingresos / (stock_actual * avg_costo_unitario)
dias_inventario = 365 / rotacion_inventario

-- Valor en inventario lento
valor_inventario_lento = CASE
    WHEN dias_sin_venta > 60
    THEN stock_actual * avg_costo_unitario
    ELSE 0
END
```

#### Python:

```
# Rotación (aproximada si no hay inventario promedio histórico)
product_metrics['rotacion_inventario'] = (
    product_metrics['total_ingresos'] /
    (product_metrics.get('stock_actual', 1) * product_metrics['avg_costo_unitario']
).replace([np.inf, -np.inf], 0)

product_metrics['dias_inventario'] = (
    365 / product_metrics['rotacion_inventario']
).replace([np.inf, -np.inf], 999)

# Valor inventario lento
product_metrics['valor_inventario_lento'] = np.where(
    product_metrics['dias_sin_venta'] > 60,
    product_metrics.get('stock_actual', 0) * product_metrics['avg_costo_unitario']
    0
)
```

### 4.4. Score de Salud de Inventario (Feature A3)

```
-- Score compuesto (0-100)
salud_inventario_score =
    CASE
```

```

    WHEN rotacion_inventario > 12 THEN 100 -- Excelente (rota >1x/mes)
    WHEN rotacion_inventario > 6 THEN 80   -- Bueno (rota cada 2 meses)
    WHEN rotacion_inventario > 3 THEN 60   -- Aceptable
    WHEN rotacion_inventario > 1 THEN 40   -- Atención
    ELSE 20                               -- Crítico
END

salud_inventario_categoria = CASE
    WHEN salud_inventario_score >= 80 THEN 'saludable'
    WHEN salud_inventario_score >= 60 THEN 'atención'
    ELSE 'crítico'
END

```

#### Python:

```

def calculate_inventory_health_score(rotation):
    if rotation > 12:
        return 100
    elif rotation > 6:
        return 80
    elif rotation > 3:
        return 60
    elif rotation > 1:
        return 40
    else:
        return 20

product_metrics['salud_inventario_score'] = (
    product_metrics['rotacion_inventario'].apply(calculate_inventory_health_score)
)

product_metrics['salud_inventario_categoria'] = pd.cut(
    product_metrics['salud_inventario_score'],
    bins=[0, 60, 80, 100],
    labels=['crítico', 'atención', 'saludable']
)

```

#### 4.5. Punto de Reorden (Feature B5)

```

-- Safety Stock = Z * StdDev(demanda) * SQRT(Lead_time)
-- Reorder Point = (Demanda_diaria * Lead_time) + Safety_stock

-- Asumiendo servicio Level 95% (Z=1.65) y Lead_time conocido
lead_time_dias = 10 -- ejemplo, debe venir de config o tabla de proveedores

```



```

safety_stock = 1.65 * stddev_cantidad_diaria * SQRT(lead_time_dias)
punto_reorden = (ventas_por_dia * lead_time_dias) + safety_stock

-- Cantidad óptima pedido (EOQ - si hay costos de pedido)
-- EOQ = SQRT((2 * demanda_anual * costo_pedido) / costo_almacenamiento)

```

## Python:

```

# Requiere calcular stddev de demanda diaria
def calculate_daily_demand_stats(df):
    """Calcula estadísticas de demanda diaria por producto"""
    daily_demand = df.groupby(['producto_clean', 'fecha'])['cantidad'].sum().reset_index()

    demand_stats = daily_demand.groupby('producto_clean')['cantidad'].agg([
        ('demanda_diaria_promedio', 'mean'),
        ('demanda_diaria_stddev', 'std')
    ]).reset_index()

    return demand_stats

demand_stats = calculate_daily_demand_stats(df)
product_metrics = product_metrics.merge(demand_stats, on='producto_clean')

# Calcular punto de reorden
LEAD_TIME_DIAS = 10 # Configuración
Z_SCORE_95 = 1.65

product_metrics['safety_stock'] = (
    Z_SCORE_95 *
    product_metrics['demanda_diaria_stddev'].fillna(0) *
    np.sqrt(LEAD_TIME_DIAS)
)

product_metrics['punto_reorden'] = (
    product_metrics['demanda_diaria_promedio'] * LEAD_TIME_DIAS +
    product_metrics['safety_stock']
)

# Alerta: debe pedir cuando stock actual < punto reorden
product_metrics['debe_pedir'] = (
    product_metrics.get('stock_actual', 999) < product_metrics['punto_reorden']
).astype(int)

```

## 5. Métricas para Pareto (Feature A1)

## 5.1. Ranking y Contribución

```
-- Ranking por ingresos
rank_ingresos = RANK() OVER (ORDER BY total_ingresos DESC)

-- % del total
pct_ingresos = total_ingresos / SUM(total_ingresos) OVER() * 100

-- % acumulado
pct_ingresos_acumulado = SUM(pct_ingresos) OVER (ORDER BY total_ingresos DESC)

-- Clasificación Pareto
pareto_categoria = CASE
    WHEN pct_ingresos_acumulado <= 80 THEN 'A - Top 80%'
    WHEN pct_ingresos_acumulado <= 95 THEN 'B - Siguiete 15%'
    ELSE 'C - Resto 5%'
END
```

### Python:

```
# Ordenar por ingresos descendente
product_metrics = product_metrics.sort_values('total_ingresos', ascending=False)

# Ranking
product_metrics['rank_ingresos'] = range(1, len(product_metrics) + 1)

# Porcentaje del total
total_ingresos_global = product_metrics['total_ingresos'].sum()
product_metrics['pct_ingresos'] = (
    product_metrics['total_ingresos'] / total_ingresos_global * 100
)

# Porcentaje acumulado
product_metrics['pct_ingresos_acumulado'] = product_metrics['pct_ingresos'].cumsum()

# Categorización Pareto
product_metrics['pareto_categoria'] = pd.cut(
    product_metrics['pct_ingresos_acumulado'],
    bins=[0, 80, 95, 100],
    labels=['A - Top 80%', 'B - Siguiete 15%', 'C - Resto 5%'],
    include_lowest=True
)
```

---

## Output Schema

```
# Schema del modelo producto
{
    'producto_clean': str,

    # Volumen
    'total_transacciones': int,
    'total_unidades_vendidas': int,
    'total_ingresos': float,
    'total_costo': float,
    'primera_venta': date,
    'ultima_venta': date,
    'dias_activo': int,

    # Promedios
    'avg_cantidad_por_transaccion': float,
    'avg_precio_unitario': float,
    'avg_costo_unitario': float,

    # Rentabilidad
    'margen_total': float,
    'margen_promedio_porcentaje': float,
    'margen_total_porcentaje': float,
    'rentabilidad_score': float,
    'rentabilidad_categoria': str, # alta/media/baja

    # Velocidad
    'ventas_por_dia': float,
    'ventas_por_semana': float,
    'ventas_por_mes': float,
    'avg_dias_entre_ventas': float,
    'stddev_dias_entre_ventas': float,
    'velocidad_categoria': str, # rápida/media/lenta

    # Inventario
    'dias_sin_venta': int,
    'stock_actual': float, # si disponible
    'dias_stock_restante': float,
    'alerta_stock_bajo': int,
    'alerta_sin_movimiento': int,
    'rotacion_inventario': float,
    'dias_inventario': float,
    'valor_inventario_lento': float,
    'salud_inventario_score': int,
    'salud_inventario_categoria': str, # saludable/atención/crítico

    # Reorden
```

```

'demanda_diaria_promedio': float,
'demanda_diaria_stddev': float,
'safety_stock': float,
'punto_reorden': float,
'debe_pedir': int,

# Pareto
'rank_ingresos': int,
'pct_ingresos': float,
'pct_ingresos_acumulado': float,
'pareto_categoria': str # A/B/C
}

```

## Implementación Completa

```

def calculate_product_model(df_enriched, stock_df=None, lead_time=10):
    """
    Calcula todas las métricas del modelo de producto

    Parameters:
    -----
    df_enriched : DataFrame
        Output del Modelo 1 (transactions enriched)
    stock_df : DataFrame, optional
        DataFrame con stock actual por producto (columnas: producto_clean, stock_)
    lead_time : int, default=10
        Días de lead time del proveedor

    Returns:
    -----
    DataFrame con métricas agregadas por producto
    """
    from datetime import date

    # 1. Métricas básicas
    product_metrics = df_enriched.groupby('producto_clean').agg({
        'trans_id': 'count',
        'cantidad': 'sum',
        'total': 'sum',
        'costo': 'sum',
        'fecha': ['min', 'max'],
        'margen_absoluto': 'sum',
        'margen_porcentaje': 'mean'
    }).reset_index()

```

```

product_metrics.columns = [
    'producto', 'total_transacciones', 'total_unidades_vendidas',
    'total_ingresos', 'total_costo', 'primera_venta', 'ultima_venta',
    'margen_total', 'margen_promedio_porcentaje'
]

# 2. Días activo
product_metrics['dias_activo'] = (
    product_metrics['ultima_venta'] - product_metrics['primera_venta']
).dt.days + 1

# 3. Promedios
product_metrics['avg_precio_unitario'] = (
    product_metrics['total_ingresos'] / product_metrics['total_unidades_vendidas']
)
product_metrics['avg_costo_unitario'] = (
    product_metrics['total_costo'] / product_metrics['total_unidades_vendidas']
)

# 4. Rentabilidad
product_metrics['margen_total_porcentaje'] = (
    (product_metrics['total_ingresos'] - product_metrics['total_costo']) /
    product_metrics['total_ingresos'] * 100
)
product_metrics['rentabilidad_score'] = (
    product_metrics['margen_total'] *
    np.log1p(product_metrics['total_unidades_vendidas'])
)
product_metrics['rentabilidad_categoria'] = pd.cut(
    product_metrics['margen_total_porcentaje'],
    bins=[0, 20, 40, 100],
    labels=['baja', 'media', 'alta']
)

# 5. Velocidad
product_metrics['ventas_por_dia'] = (
    product_metrics['total_transacciones'] / product_metrics['dias_activo']
)

# Días entre ventas
velocity_metrics = calculate_days_between_sales(df_enriched)
product_metrics = product_metrics.merge(velocity_metrics, on='producto')

product_metrics['velocidad_categoria'] = pd.cut(
    product_metrics['avg_dias_entre_ventas'],
    bins=[0, 7, 30, np.inf],
    labels=['rápida', 'media', 'lenta']
)

```

```

)

# 6. Inventario - estado actual
fecha_analisis = date.today()
product_metrics['dias_sin_venta'] = (
    fecha_analisis - pd.to_datetime(product_metrics['ultima_venta']).dt.date
).dt.days

# Merge stock actual si está disponible
if stock_df is not None:
    product_metrics = product_metrics.merge(
        stock_df[['producto_clean', 'stock_actual']],
        left_on='producto',
        right_on='producto_clean',
        how='left'
    )
else:
    product_metrics['stock_actual'] = np.nan

# 7. Alertas y métricas de inventario
product_metrics['dias_stock_restante'] = (
    product_metrics['stock_actual'] / product_metrics['ventas_por_dia']
).fillna(999)

product_metrics['alerta_stock_bajo'] = (
    product_metrics['dias_stock_restante'] < 7
).astype(int)

product_metrics['alerta_sin_movimiento'] = (
    product_metrics['dias_sin_venta'] > 90
).astype(int)

# 8. Rotación
product_metrics['rotacion_inventario'] = (
    product_metrics['total_ingresos'] /
    (product_metrics['stock_actual'] * product_metrics['avg_costo_unitario'])
).replace([np.inf, -np.inf], 0).fillna(0)

product_metrics['salud_inventario_score'] = (
    product_metrics['rotacion_inventario'].apply(calculate_inventory_health_s
)

# 9. Punto de reorden
demand_stats = calculate_daily_demand_stats(df_enriched)
product_metrics = product_metrics.merge(demand_stats, on='producto')

Z_SCORE_95 = 1.65

```

```

product_metrics['safety_stock'] = (
    Z_SCORE_95 *
    product_metrics['demanda_diaria_stddev'].fillna(0) *
    np.sqrt(lead_time)
)

product_metrics['punto_reorden'] = (
    product_metrics['demanda_diaria_promedio'] * lead_time +
    product_metrics['safety_stock']
)

# 10. Pareto
product_metrics = product_metrics.sort_values('total_ingresos', ascending=False)
product_metrics['rank_ingresos'] = range(1, len(product_metrics) + 1)

total_ingresos_global = product_metrics['total_ingresos'].sum()
product_metrics['pct_ingresos'] = (
    product_metrics['total_ingresos'] / total_ingresos_global * 100
)
product_metrics['pct_ingresos_acumulado'] = product_metrics['pct_ingresos'].cumsum()

product_metrics['pareto_categoria'] = pd.cut(
    product_metrics['pct_ingresos_acumulado'],
    bins=[0, 80, 95, 100],
    labels=['A - Top 80%', 'B - Siguiete 15%', 'C - Resto 5%'],
    include_lowest=True
)

return product_metrics




# Uso
product_model = calculate_product_model(
    df_enriched=df_enriched,
    stock_df=stock_current, # opcional
    lead_time=10
)

```

---

## Dependencies

### Input Requirements:

-  Output de Modelo 1 (Transaction Enriched)
-  Stock actual por producto (para métricas de inventario precisas)
-  Lead time por producto o proveedor (para reorder point)

### Configuration:

- `lead_time_dias`: Días que tarda el proveedor en entregar
  - `z_score_servicio`: Z-score para nivel de servicio deseado (default: 1.65 para 95%)
  - `umbral_stock_bajo`: Días de stock para alerta (default: 7)
  - `umbral_sin_movimiento`: Días sin venta para alerta (default: 90)
- 

## Notas

- Este modelo se recalcula típicamente **mensualmente**
- Para alertas en tiempo real, calcular solo métricas críticas (stock, días sin venta)
- Pareto debe recalcularse cuando cambia sustancialmente el mix de ventas