Modelo 3: Customer-Level Model (Agregación por Cliente)

Descripción del Modelo

Este modelo agrega datos a nivel de **cliente**. Calcula métricas RFM (Recency, Frequency, Monetary), segmentación, y detección de churn. Corresponde a la etapa **ATTRIBUTES** del workflow con `GROUP BY customer_id`.

Input

Output del Modelo 1 (Transaction Enriched)

Aggregation Key

```
GROUP BY customer_id_clean
```

Features Soportados

- **A1** - Análisis Pareto (clientes)
- **A4** - Segmentación de Clientes
- **B4** - Detección de Clientes en Riesgo de Abandonar

Cálculos del Modelo

1. Métricas Básicas de Cliente

1.1. Volumen y Actividad

```
total_compras = COUNT(trans_id)
total_unidades_compradas = SUM(cantidad)
total_gastado = SUM(total)
ticket_promedio = AVG(total)

primera_compra = MIN(fecha)
ultima_compra = MAX(fecha)
antiguedad_dias = DATEDIFF(CURRENT_DATE, primera_compra)
```

Python:

```
def calculate_customer_basics(df):
    """Calcula métricas básicas por cliente"""
    customer_metrics = df.groupby('customer_id_clean').agg({
```

```
        'trans_id': 'count',
        'cantidad': 'sum',
        'total': ['sum', 'mean'],
        'fecha': ['min', 'max'],
        'customer_name': 'first',
        'customer_location': 'first'
    }).reset_index()

    customer_metrics.columns = [
        'customer_id', 'total_compras', 'total_unidades_compradas',
        'total_gastado', 'ticket_promedio', 'primera_compra',
        'ultima_compra', 'customer_name', 'customer_location'
    ]

    from datetime import date
    fecha_analisis = date.today()

    customer_metrics['antiguedad_dias'] = (
        fecha_analisis - pd.to_datetime(customer_metrics['primera_compra']).dt.da
    ).dt.days

    return customer_metrics

customer_model = calculate_customer_basics(df_enriched)
```

1.2. Productos Únicos y Diversidad

```
productos_unicos = COUNT(DISTINCT producto)
categorias_unicas = COUNT(DISTINCT categoria_producto)  -- si existe

-- Índice de diversidad (Shannon entropy simplificado)
-- Mide qué tan diverso es el portafolio de compra
```

Python:

```
# Productos únicos
products_per_customer = df_enriched.groupby('customer_id_clean')['producto_clean'
products_per_customer.columns = ['customer_id', 'productos_unicos']

customer_model = customer_model.merge(products_per_customer, on='customer_id')

# Producto más comprado
top_product = df_enriched.groupby(['customer_id_clean', 'producto_clean']).size()
top_product = top_product.sort_values('count', ascending=False).groupby('customer
top_product.columns = ['customer_id', 'producto_favorito', 'compras_producto_favo
```

```
customer_model = customer_model.merge(top_product[['customer_id', 'producto_favor
```

---

2. Análisis RFM (Feature A4 - Base para Segmentación)

2.1. Recency (Recencia)

```
-- Días desde última compra
recency_dias = DATEDIFF(CURRENT_DATE, MAX(fecha))
```

Python:

```
from datetime import date

fecha_analisis = date.today()

customer_model['recency_dias'] = (
    fecha_analisis - pd.to_datetime(customer_model['ultima_compra']).dt.date
).dt.days
```

2.2. Frequency (Frecuencia)

```
-- Ya calculado: total_compras

-- Frecuencia normalizada por tiempo activo
frecuencia_mensual = total_compras / (antiguedad_dias / 30.0)

-- Días promedio entre compras
dias_promedio_entre_compras = antiguedad_dias / NULLIF(total_compras - 1, 0)
```

Python:

```
customer_model['frecuencia_mensual'] = (
    customer_model['total_compras'] /
    (customer_model['antiguedad_dias'] / 30.0)
)

customer_model['dias_promedio_entre_compras'] = (
    customer_model['antiguedad_dias'] /
    (customer_model['total_compras'] - 1)
).replace([np.inf, -np.inf], np.nan)
```

2.3. Monetary (Valor Monetario)

```
-- Ya calculado: total_gastado
-- También: ticket_promedio

-- Valor de lifetime (LTV simplificado)
ltv_estimado = total_gastado
```

Python:

```
# Ya tenemos total_gastado y ticket_promedio
customer_model['ltv_estimado'] = customer_model['total_gastado']
```

2.4. Scores RFM

```
-- Quintiles para cada métrica (1-5, donde 5 es mejor)
-- Recency: menor días = mejor score
-- Frequency: más compras = mejor score
-- Monetary: más gastado = mejor score

recency_score = NTILE(5) OVER (ORDER BY recency_dias ASC)
frequency_score = NTILE(5) OVER (ORDER BY total_compras DESC)
monetary_score = NTILE(5) OVER (ORDER BY total_gastado DESC)

-- Score RFM compuesto (concatenación o suma ponderada)
rfm_score = CONCAT(recency_score, frequency_score, monetary_score)  -- ej: "555"
rfm_score_numeric = (recency_score * 100) + (frequency_score * 10) + monetary_sco
```

Python:

```
def calculate_rfm_scores(df):
    """Calcula scores RFM en quintiles"""
    # Recency: menor es mejor, invertir orden
    df['recency_score'] = pd.qcut(
        df['recency_dias'],
        q=5,
        labels=[5, 4, 3, 2, 1],  # invertido: menos días = mejor score
        duplicates='drop'
    ).astype(int)

    # Frequency: mayor es mejor
    df['frequency_score'] = pd.qcut(
        df['total_compras'],
        q=5,
```

```
        labels=[1, 2, 3, 4, 5],
        duplicates='drop'
    ).astype(int)

    # Monetary: mayor es mejor
    df['monetary_score'] = pd.qcut(
        df['total_gastado'],
        q=5,
        labels=[1, 2, 3, 4, 5],
        duplicates='drop'
    ).astype(int)

    # RFM compuesto
    df['rfm_score'] = (
        df['recency_score'].astype(str) +
        df['frequency_score'].astype(str) +
        df['monetary_score'].astype(str)
    )

    df['rfm_score_numeric'] = (
        df['recency_score'] * 100 +
        df['frequency_score'] * 10 +
        df['monetary_score']
    )

    return df

customer_model = calculate_rfm_scores(customer_model)
```

---

3. Segmentación de Clientes (Feature A4)

3.1. Segmentos Basados en RFM

```
-- Segmentación simplificada
segmento = CASE
    -- Champions: Compran frecuente y recientemente, alto valor
    WHEN recency_score >= 4 AND frequency_score >= 4 AND monetary_score >= 4
    THEN 'Champions'

    -- Leales: Compran frecuente, valor alto
    WHEN frequency_score >= 4 AND monetary_score >= 4
    THEN 'Leales'

    -- Potenciales Leales: Compran reciente, frecuencia media
    WHEN recency_score >= 4 AND frequency_score BETWEEN 2 AND 3
```

```sql
    THEN 'Potenciales Leales'

    -- Nuevos: Primera compra reciente
    WHEN recency_score >= 4 AND frequency_score <= 2
    THEN 'Nuevos'

    -- Necesitan Atención: Buenos clientes pero inactivos
    WHEN recency_score <= 2 AND frequency_score >= 3
    THEN 'Necesitan Atención'

    -- En Riesgo: Eran buenos, ahora inactivos
    WHEN recency_score <= 2 AND frequency_score >= 4 AND monetary_score >= 4
    THEN 'En Riesgo'

    -- Dormidos: Mucho tiempo sin comprar
    WHEN recency_score <= 2 AND frequency_score <= 2
    THEN 'Dormidos'

    -- Ocasionales: Compran poco y esporádico
    ELSE 'Ocasionales'
END
```

Python:

```python
def segment_customers_rfm(df):
    """Segmenta clientes basado en scores RFM"""
    conditions = [
        # Champions
        (df['recency_score'] >= 4) & (df['frequency_score'] >= 4) & (df['monetary

        # Leales
        (df['frequency_score'] >= 4) & (df['monetary_score'] >= 4),

        # Potenciales Leales
        (df['recency_score'] >= 4) & (df['frequency_score'].between(2, 3)),

        # Nuevos
        (df['recency_score'] >= 4) & (df['frequency_score'] <= 2),

        # Necesitan Atención
        (df['recency_score'] <= 2) & (df['frequency_score'] >= 3),

        # En Riesgo
        (df['recency_score'] <= 2) & (df['frequency_score'] >= 4) & (df['monetary

        # Dormidos
```

```
        (df['recency_score'] <= 2) & (df['frequency_score'] <= 2)
    ]

    choices = [
        'Champions',
        'Leales',
        'Potenciales Leales',
        'Nuevos',
        'Necesitan Atención',
        'En Riesgo',
        'Dormidos'
    ]

    df['segmento'] = np.select(conditions, choices, default='Ocasionales')

    return df

customer_model = segment_customers_rfm(customer_model)
```

3.2. Segmentación Alternativa Simple

```
-- Basada en frecuencia y recencia únicamente
segmento_simple = CASE
    WHEN recency_dias <= 30 AND total_compras >= 5 THEN 'Frecuentes'
    WHEN recency_dias <= 60 AND total_compras >= 2 THEN 'Ocasionales'
    WHEN recency_dias > 90 AND total_compras > 0 THEN 'Dormidos'
    WHEN total_compras = 1 AND recency_dias <= 60 THEN 'Nuevos'
    ELSE 'Inactivos'
END
```

Python:

```
def segment_customers_simple(df):
    """Segmentación simple basada en recencia y frecuencia"""
    conditions = [
        (df['recency_dias'] <= 30) & (df['total_compras'] >= 5),
        (df['recency_dias'] <= 60) & (df['total_compras'] >= 2),
        (df['recency_dias'] > 90) & (df['total_compras'] > 0),
        (df['total_compras'] == 1) & (df['recency_dias'] <= 60)
    ]

    choices = ['Frecuentes', 'Ocasionales', 'Dormidos', 'Nuevos']

    df['segmento_simple'] = np.select(conditions, choices, default='Inactivos')
```

```
    return df

customer_model = segment_customers_simple(customer_model)
```

---

4. Detección de Riesgo de Churn (Feature B4)

4.1. Patrón de Compra Esperado

```
-- Frecuencia esperada basada en histórico
frecuencia_esperada_dias = dias_promedio_entre_compras

-- Días desde última compra esperada
dias_desde_compra_esperada = recency_dias - frecuencia_esperada_dias

-- Desviación del patrón
desviacion_patron = dias_desde_compra_esperada / frecuencia_esperada_dias
```

Python:

```
customer_model['frecuencia_esperada_dias'] = customer_model['dias_promedio_entre_

customer_model['dias_desde_compra_esperada'] = (
    customer_model['recency_dias'] -
    customer_model['frecuencia_esperada_dias']
)

customer_model['desviacion_patron'] = (
    customer_model['dias_desde_compra_esperada'] /
    customer_model['frecuencia_esperada_dias']
).replace([np.inf, -np.inf], np.nan)
```

4.2. Score de Riesgo de Churn

```
-- Score 0-100, donde 100 = alto riesgo
riesgo_churn_score = CASE
    WHEN desviacion_patron <= 0 THEN 0  -- Comprando antes de lo esperado
    WHEN desviacion_patron < 0.5 THEN 10  -- Dentro de patrón normal
    WHEN desviacion_patron < 1.0 THEN 30  -- Ligeramente retrasado
    WHEN desviacion_patron < 1.5 THEN 60  -- Retrasado
    WHEN desviacion_patron < 2.0 THEN 80  -- Muy retrasado
    ELSE 100  -- Extremadamente retrasado
END
```

```
-- Categoría de riesgo
riesgo_categoria = CASE
    WHEN riesgo_churn_score >= 80 THEN 'Alto'
    WHEN riesgo_churn_score >= 60 THEN 'Medio'
    WHEN riesgo_churn_score >= 30 THEN 'Bajo'
    ELSE 'Activo'
END
```

Python:

```
def calculate_churn_risk(df):
    """Calcula score de riesgo de churn"""
    conditions = [
        df['desviacion_patron'] <= 0,
        df['desviacion_patron'] < 0.5,
        df['desviacion_patron'] < 1.0,
        df['desviacion_patron'] < 1.5,
        df['desviacion_patron'] < 2.0
    ]

    scores = [0, 10, 30, 60, 80]

    df['riesgo_churn_score'] = np.select(
        conditions,
        scores,
        default=100
    )

    # Manejar NaN (clientes con 1 sola compra)
    df.loc[df['total_compras'] == 1, 'riesgo_churn_score'] = 0

    # Categoría
    df['riesgo_categoria'] = pd.cut(
        df['riesgo_churn_score'],
        bins=[0, 30, 60, 80, 100],
        labels=['Activo', 'Bajo', 'Medio', 'Alto'],
        include_lowest=True
    )

    return df

customer_model = calculate_churn_risk(customer_model)
```

4.3. Prioridad de Contacto

```
-- Prioriza clientes por riesgo * valor
prioridad_contacto = riesgo_churn_score * LOG(total_gastado + 1)

-- Solo para clientes con riesgo >= Medio
debe_contactar = CASE
    WHEN riesgo_categoria IN ('Alto', 'Medio') AND total_gastado > 50000
    THEN 1
    ELSE 0
END
```

Python:

```
customer_model['prioridad_contacto'] = (
    customer_model['riesgo_churn_score'] *
    np.log1p(customer_model['total_gastado'])
)

customer_model['debe_contactar'] = (
    (customer_model['riesgo_categoria'].isin(['Alto', 'Medio'])) &
    (customer_model['total_gastado'] > 50000)
).astype(int)
```

---

5. Análisis Pareto de Clientes (Feature A1)

5.1. Ranking y Contribución

```
rank_valor = RANK() OVER (ORDER BY total_gastado DESC)
pct_valor = total_gastado / SUM(total_gastado) OVER() * 100
pct_valor_acumulado = SUM(pct_valor) OVER (ORDER BY total_gastado DESC)

pareto_categoria = CASE
    WHEN pct_valor_acumulado <= 80 THEN 'A - Top 80%'
    WHEN pct_valor_acumulado <= 95 THEN 'B - Siguiente 15%'
    ELSE 'C - Resto 5%'
END
```

Python:

```
# Ordenar por gasto descendente
customer_model = customer_model.sort_values('total_gastado', ascending=False)

# Ranking
customer_model['rank_valor'] = range(1, len(customer_model) + 1)
```

```
# Porcentaje del total
total_gastado_global = customer_model['total_gastado'].sum()
customer_model['pct_valor'] = (
    customer_model['total_gastado'] / total_gastado_global * 100
)

# Porcentaje acumulado
customer_model['pct_valor_acumulado'] = customer_model['pct_valor'].cumsum()

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

---

6. Métricas de Valor de Cliente

6.1. Customer Lifetime Value (CLV) Estimado

```
-- CLV Simple = (ticket_promedio * frecuencia_mensual) * meses_vida_esperada
-- Asumiendo vida esperada promedio de 12 meses

clv_estimado = ticket_promedio * frecuencia_mensual * 12

-- CLV basado en tendencia actual
clv_proyectado = total_gastado * (365 / antiguedad_dias) * 2  -- proyección 2 año
```

Python:

```
# CLV simple (proyección 12 meses)
customer_model['clv_estimado_12m'] = (
    customer_model['ticket_promedio'] *
    customer_model['frecuencia_mensual'] *
    12
)

# CLV basado en rate actual proyectado 2 años
customer_model['clv_proyectado_24m'] = (
    customer_model['total_gastado'] *
    (365 / customer_model['antiguedad_dias']) *
```

```
    2
).replace([np.inf, -np.inf], 0)
```

Output Schema

```
{
    'customer_id': str,
    'customer_name': str,
    'customer_location': str,

    # Métricas básicas
    'total_compras': int,
    'total_unidades_compradas': int,
    'total_gastado': float,
    'ticket_promedio': float,
    'primera_compra': date,
    'ultima_compra': date,
    'antiguedad_dias': int,
    'productos_unicos': int,
    'producto_favorito': str,

    # RFM
    'recency_dias': int,
    'frecuencia_mensual': float,
    'dias_promedio_entre_compras': float,
    'ltv_estimado': float,
    'recency_score': int,  # 1-5
    'frequency_score': int,  # 1-5
    'monetary_score': int,  # 1-5
    'rfm_score': str,  # "555"
    'rfm_score_numeric': int,  # 555

    # Segmentación
    'segmento': str,  # Champions/Leales/etc
    'segmento_simple': str,  # Frecuentes/Ocasionales/etc

    # Churn
    'frecuencia_esperada_dias': float,
    'dias_desde_compra_esperada': float,
    'desviacion_patron': float,
    'riesgo_churn_score': int,  # 0-100
    'riesgo_categoria': str,  # Alto/Medio/Bajo/Activo
    'prioridad_contacto': float,
    'debe_contactar': int,  # 0/1
```

```
    # Pareto
    'rank_valor': int,
    'pct_valor': float,
    'pct_valor_acumulado': float,
    'pareto_categoria': str,  # A/B/C

    # CLV
    'clv_estimado_12m': float,
    'clv_proyectado_24m': float
}
```

Implementación Completa

```
def calculate_customer_model(df_enriched):
    """
    Calcula todas las métricas del modelo de cliente

    Parameters:
    -----------
    df_enriched : DataFrame
        Output del Modelo 1 (transactions enriched)

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

    # 1. Métricas básicas
    customer_model = calculate_customer_basics(df_enriched)

    # 2. Productos únicos
    products_per_customer = df_enriched.groupby('customer_id_clean')['producto_cl
    products_per_customer.columns = ['customer_id', 'productos_unicos']
    customer_model = customer_model.merge(products_per_customer, on='customer_id'

    # Producto favorito
    top_product = df_enriched.groupby(['customer_id_clean', 'producto_clean']).si
    top_product = top_product.sort_values('count', ascending=False).groupby('cust
    customer_model = customer_model.merge(
        top_product[['customer_id_clean', 'producto_clean']].rename(
            columns={'customer_id_clean': 'customer_id', 'producto_clean': 'produ
        ),
        on='customer_id'
    )
```

```
    # 3. RFM
    fecha_analisis = date.today()
    customer_model['recency_dias'] = (
        fecha_analisis - pd.to_datetime(customer_model['ultima_compra']).dt.date
    ).dt.days

    customer_model['frecuencia_mensual'] = (
        customer_model['total_compras'] / (customer_model['antiguedad_dias'] / 30
    )

    customer_model['dias_promedio_entre_compras'] = (
        customer_model['antiguedad_dias'] / (customer_model['total_compras'] - 1)
    ).replace([np.inf, -np.inf], np.nan)

    customer_model['ltv_estimado'] = customer_model['total_gastado']

    # Scores RFM
    customer_model = calculate_rfm_scores(customer_model)

    # 4. Segmentación
    customer_model = segment_customers_rfm(customer_model)
    customer_model = segment_customers_simple(customer_model)

    # 5. Churn
    customer_model['frecuencia_esperada_dias'] = customer_model['dias_promedio_er
    customer_model['dias_desde_compra_esperada'] = (
        customer_model['recency_dias'] - customer_model['frecuencia_esperada_dias
    )
    customer_model['desviacion_patron'] = (
        customer_model['dias_desde_compra_esperada'] / customer_model['frecuencia
    ).replace([np.inf, -np.inf], np.nan)

    customer_model = calculate_churn_risk(customer_model)

    customer_model['prioridad_contacto'] = (
        customer_model['riesgo_churn_score'] * np.log1p(customer_model['total_gas
    )
    customer_model['debe_contactar'] = (
        (customer_model['riesgo_categoria'].isin(['Alto', 'Medio'])) &
        (customer_model['total_gastado'] > 50000)
    ).astype(int)

    # 6. Pareto
    customer_model = customer_model.sort_values('total_gastado', ascending=False)
    customer_model['rank_valor'] = range(1, len(customer_model) + 1)
```

```
    total_gastado_global = customer_model['total_gastado'].sum()
    customer_model['pct_valor'] = (customer_model['total_gastado'] / total_gastad
    customer_model['pct_valor_acumulado'] = customer_model['pct_valor'].cumsum()

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

    # 7. CLV
    customer_model['clv_estimado_12m'] = (
        customer_model['ticket_promedio'] * customer_model['frecuencia_mensual']
    )
    customer_model['clv_proyectado_24m'] = (
        customer_model['total_gastado'] * (365 / customer_model['antiguedad_dias'
    ).replace([np.inf, -np.inf], 0)

    return customer_model

# Uso
customer_model = calculate_customer_model(df_enriched)
```

Reporte de Segmentos

```
def segment_summary(customer_model):
    """Genera resumen de segmentos de clientes"""
    summary = customer_model.groupby('segmento').agg({
        'customer_id': 'count',
        'total_gastado': 'sum',
        'ticket_promedio': 'mean',
        'total_compras': 'mean',
        'recency_dias': 'mean'
    }).reset_index()

    summary.columns = [
        'segmento', 'num_clientes', 'valor_total',
        'ticket_promedio', 'compras_promedio', 'recency_promedio'
    ]

    summary['pct_clientes'] = summary['num_clientes'] / summary['num_clientes'].s
    summary['pct_valor'] = summary['valor_total'] / summary['valor_total'].sum()

    return summary.sort_values('valor_total', ascending=False)
```

```
# Uso
print(segment_summary(customer_model))
```

Dependencies

Input Requirements:

- ✅ Output de Modelo 1 (Transaction Enriched)

Configuration:

- `umbral_contacto_valor`: Valor mínimo para priorizar contacto (default: $50,000)
- `vida_esperada_meses`: Para cálculo CLV (default: 12)

Notas

- Este modelo se recalcula típicamente **mensualmente**
- Para alertas de churn, se puede ejecutar solo la sección de riesgo **semanalmente**
- Segmentos deben revisarse trimestralmente y ajustar umbrales según negocio