Modelo 5: Product-Basket Model (Market Basket Analysis)

Descripción del Modelo

Este modelo analiza **asociaciones entre productos** - qué productos se compran juntos. Identifica patrones de compra, combos naturales y oportunidades de cross-selling/upselling. Corresponde a agregación por **pares de productos** o **cestas de compra**.

Input

Output del Modelo 1 (Transaction Enriched)

Aggregation Keys

- `GROUP BY producto_A, producto_B` (pares)
- `GROUP BY basket_id` (cestas - transacciones del mismo cliente en mismo día/hora)

Features Soportados

- **B3** - Análisis de Canastas (Market Basket)

Conceptos Clave

Métricas de Asociación

Support (Soporte):

```
Support(A) = Transacciones con A / Total transacciones
Support(A,B) = Transacciones con A y B / Total transacciones
```

Confidence (Confianza):

```
Confidence(A → B) = Transacciones con A y B / Transacciones con A
```

Indica: "Si compro A, qué probabilidad hay de comprar B"

Lift (Elevación):

```
Lift(A → B) = Confidence(A → B) / Support(B)
            = P(A,B) / (P(A) * P(B))
```

- Lift > 1: Productos tienen asociación positiva
- Lift = 1: Productos independientes

- Lift < 1: Productos tienen asociación negativa

Conviction (Convicción):

```
Conviction(A → B) = (1 - Support(B)) / (1 - Confidence(A → B))
```

Indica qué tan dependiente es B de A

Cálculos del Modelo

1. Identificación de Cestas

1.1. Definir Cestas de Compra

```python
def create_baskets(df, time_window_hours=4):
    """
    Crea cestas de compra agrupando transacciones del mismo cliente
    en ventana de tiempo cercana

    Parameters:
    -----------
    df : DataFrame
        Transaction enriched data
    time_window_hours : int
        Horas para considerar transacciones como misma cesta

    Returns:
    --------
    DataFrame con basket_id por transacción
    """
    df = df.sort_values(['customer_id_clean', 'fecha', 'inith', 'initm'])

    # Crear timestamp completo
    df['timestamp'] = pd.to_datetime(
        df['fecha'].astype(str) + ' ' +
        df['inith'].astype(str).str.zfill(2) + ':' +
        df['initm'].astype(str).str.zfill(2)
    )

    # Calcular diferencia de tiempo con transacción anterior del mismo cliente
    df['time_diff'] = df.groupby('customer_id_clean')['timestamp'].diff()

    # Nueva cesta si diferencia > time_window o diferente cliente
    df['new_basket'] = (
        (df['time_diff'] > pd.Timedelta(hours=time_window_hours)) |
        (df['time_diff'].isna())
```

```
    ).astype(int)

    # Basket ID único
    df['basket_id'] = df.groupby('customer_id_clean')['new_basket'].cumsum()
    df['basket_id'] = (
        df['customer_id_clean'] + '_B' +
        df['basket_id'].astype(str)
    )

    return df[['trans_id', 'basket_id', 'customer_id_clean', 'producto_clean',
               'fecha', 'timestamp', 'total']]

baskets_df = create_baskets(df_enriched)
```

1.2. Análisis de Cestas

```
def analyze_baskets(baskets_df):
    """Analiza características de las cestas"""
    basket_stats = baskets_df.groupby('basket_id').agg({
        'trans_id': 'count',
        'producto_clean': 'nunique',
        'total': 'sum',
        'customer_id_clean': 'first',
        'fecha': 'first'
    }).reset_index()

    basket_stats.columns = [
        'basket_id', 'num_items', 'num_productos_unicos',
        'valor_cesta', 'customer_id', 'fecha'
    ]

    # Estadísticas generales
    stats = {
        'total_cestas': len(basket_stats),
        'avg_items_por_cesta': basket_stats['num_items'].mean(),
        'avg_productos_unicos': basket_stats['num_productos_unicos'].mean(),
        'avg_valor_cesta': basket_stats['valor_cesta'].mean(),
        'cestas_single_item': (basket_stats['num_productos_unicos'] == 1).sum(),
        'cestas_multi_item': (basket_stats['num_productos_unicos'] > 1).sum(),
        'pct_cestas_multi_item': (basket_stats['num_productos_unicos'] > 1).mean(
    }

    return basket_stats, stats

basket_stats, basket_summary = analyze_baskets(baskets_df)
print(f"Resumen de Cestas:")
```

```
for key, value in basket_summary.items():
    print(f"  {key}: {value}")
```

---

2. Análisis de Frecuencia Individual

2.1. Support Individual de Productos

```
def calculate_individual_support(baskets_df):
    """Calcula support individual de cada producto"""
    total_baskets = baskets_df['basket_id'].nunique()

    product_freq = baskets_df.groupby('producto_clean')['basket_id'].nunique().re
    product_freq.columns = ['producto', 'freq_cestas']

    product_freq['support'] = product_freq['freq_cestas'] / total_baskets
    product_freq['pct_cestas'] = product_freq['support'] * 100

    return product_freq.sort_values('support', ascending=False)

product_support = calculate_individual_support(baskets_df)
```

---

3. Análisis de Pares de Productos

3.1. Identificar Todos los Pares

```
from itertools import combinations

def create_product_pairs(baskets_df):
    """
    Crea matriz de pares de productos que aparecen en mismas cestas
    """
    # Agrupar productos por cesta
    basket_products = baskets_df.groupby('basket_id')['producto_clean'].apply(lis

    # Crear pares de productos por cada cesta
    pairs_list = []

    for _, row in basket_products.iterrows():
        products = row['producto_clean']
        if len(products) >= 2:
            # Crear todos los pares únicos de la cesta
            for prod_a, prod_b in combinations(sorted(set(products)), 2):
                pairs_list.append({
```

```
                    'basket_id': row['basket_id'],
                    'producto_a': prod_a,
                    'producto_b': prod_b
                })

    pairs_df = pd.DataFrame(pairs_list)
    return pairs_df

product_pairs = create_product_pairs(baskets_df)
```

3.2. Calcular Métricas de Asociación

```
def calculate_association_metrics(baskets_df, min_support=0.01):
    """
    Calcula métricas de asociación para pares de productos

    Parameters:
    -----------
    baskets_df : DataFrame
        Dataframe con basket_id y producto_clean
    min_support : float
        Support mínimo para considerar pares (default 1%)
    """
    total_baskets = baskets_df['basket_id'].nunique()

    # 1. Support individual
    product_support = calculate_individual_support(baskets_df)
    product_support_dict = dict(zip(
        product_support['producto'],
        product_support['support']
    ))

    # 2. Crear pares
    pairs_df = create_product_pairs(baskets_df)

    # 3. Frecuencia de pares
    pair_freq = pairs_df.groupby(['producto_a', 'producto_b']).size().reset_index

    # 4. Calcular métricas
    pair_freq['support_pair'] = pair_freq['freq_pares'] / total_baskets

    # Filtrar por support mínimo
    pair_freq = pair_freq[pair_freq['support_pair'] >= min_support]

    # Support individual de cada producto
    pair_freq['support_a'] = pair_freq['producto_a'].map(product_support_dict)
```

```
    pair_freq['support_b'] = pair_freq['producto_b'].map(product_support_dict)

    # Confidence A → B
    pair_freq['confidence_a_to_b'] = pair_freq['support_pair'] / pair_freq['suppo

    # Confidence B → A
    pair_freq['confidence_b_to_a'] = pair_freq['support_pair'] / pair_freq['suppo

    # Lift
    pair_freq['lift'] = pair_freq['support_pair'] / (pair_freq['support_a'] * pai

    # Conviction A → B
    pair_freq['conviction_a_to_b'] = (
        (1 - pair_freq['support_b']) /
        (1 - pair_freq['confidence_a_to_b'])
    ).replace([np.inf, -np.inf], np.nan)

    # Conviction B → A
    pair_freq['conviction_b_to_a'] = (
        (1 - pair_freq['support_a']) /
        (1 - pair_freq['confidence_b_to_a'])
    ).replace([np.inf, -np.inf], np.nan)

    return pair_freq.sort_values('lift', ascending=False)

association_rules = calculate_association_metrics(baskets_df, min_support=0.01)
```

---

4. Reglas de Asociación Fuertes

4.1. Filtrar Reglas Significativas

```
def filter_strong_rules(association_rules, min_confidence=0.3, min_lift=1.2):
    """
    Filtra reglas de asociación con métricas significativas

    Parameters:
    -----------
    association_rules : DataFrame
        Output de calculate_association_metrics
    min_confidence : float
        Confidence mínima (default 30%)
    min_lift : float
        Lift mínimo (default 1.2)
    """
    # Crear reglas direccionales (A → B y B → A)
```

```
    rules_a_to_b = association_rules[
        (association_rules['confidence_a_to_b'] >= min_confidence) &
        (association_rules['lift'] >= min_lift)
    ].copy()

    rules_a_to_b['antecedent'] = rules_a_to_b['producto_a']
    rules_a_to_b['consequent'] = rules_a_to_b['producto_b']
    rules_a_to_b['confidence'] = rules_a_to_b['confidence_a_to_b']
    rules_a_to_b['conviction'] = rules_a_to_b['conviction_a_to_b']

    rules_b_to_a = association_rules[
        (association_rules['confidence_b_to_a'] >= min_confidence) &
        (association_rules['lift'] >= min_lift)
    ].copy()

    rules_b_to_a['antecedent'] = rules_b_to_a['producto_b']
    rules_b_to_a['consequent'] = rules_b_to_a['producto_a']
    rules_b_to_a['confidence'] = rules_b_to_a['confidence_b_to_a']
    rules_b_to_a['conviction'] = rules_b_to_a['conviction_b_to_a']

    # Combinar
    strong_rules = pd.concat([
        rules_a_to_b[['antecedent', 'consequent', 'support_pair', 'confidence', '
        rules_b_to_a[['antecedent', 'consequent', 'support_pair', 'confidence', '
    ])

    # Ordenar por lift descendente
    strong_rules = strong_rules.sort_values('lift', ascending=False)

    # Crear descripción legible
    strong_rules['regla'] = (
        "Si compra " + strong_rules['antecedent'] +
        " → " + str(round(strong_rules['confidence'] * 100, 1)) + "% compra " +
        strong_rules['consequent']
    )

    return strong_rules

strong_rules = filter_strong_rules(association_rules, min_confidence=0.3, min_lif
```

---

5. Recomendaciones de Producto

5.1. Top N Recomendaciones por Producto

```python
def get_product_recommendations(producto, strong_rules, n=5):
    """
    Obtiene top N productos recomendados para un producto dado

    Parameters:
    -----------
    producto : str
        Nombre del producto
    strong_rules : DataFrame
        Reglas de asociación fuertes
    n : int
        Número de recomendaciones
    """
    recommendations = strong_rules[
        strong_rules['antecedent'] == producto
    ].head(n)

    if len(recommendations) == 0:
        return None

    return recommendations[['consequent', 'confidence', 'lift']].rename(columns={
        'consequent': 'producto_recomendado',
        'confidence': 'probabilidad',
        'lift': 'fuerza_asociacion'
    })

# Ejemplo
producto = "PRODUCTO_A"
recomendaciones = get_product_recommendations(producto, strong_rules, n=5)
print(f"\nSi cliente compra {producto}, recomendar:")
print(recomendaciones)
```

5.2. Matriz de Recomendaciones

```python
def create_recommendation_matrix(strong_rules, top_n=3):
    """
    Crea matriz de recomendaciones: para cada producto, sus top N recomendaciones
    """
    products = pd.concat([
        strong_rules['antecedent'],
        strong_rules['consequent']
    ]).unique()

    recommendations_dict = {}
```

```
    for product in products:
        recs = get_product_recommendations(product, strong_rules, n=top_n)
        if recs is not None:
            recommendations_dict[product] = recs['producto_recomendado'].tolist()
        else:
            recommendations_dict[product] = []

    return pd.DataFrame([
        {'producto': k, 'recomendaciones': ', '.join(v[:3])}
        for k, v in recommendations_dict.items()
    ])

recommendation_matrix = create_recommendation_matrix(strong_rules, top_n=3)
```

---

6. Análisis de Combos Naturales

6.1. Identificar Productos que Siempre van Juntos

```
def identify_natural_bundles(association_rules, min_confidence=0.7, min_lift=2.0):
    """
    Identifica pares de productos que frecuentemente se compran juntos
    (candidatos para combos/bundles)
    """
    bundles = association_rules[
        (association_rules['confidence_a_to_b'] >= min_confidence) &
        (association_rules['confidence_b_to_a'] >= min_confidence) &
        (association_rules['lift'] >= min_lift)
    ].copy()

    bundles['avg_confidence'] = (
        bundles['confidence_a_to_b'] + bundles['confidence_b_to_a']
    ) / 2

    bundles['bundle_score'] = bundles['lift'] * bundles['avg_confidence']

    bundles = bundles.sort_values('bundle_score', ascending=False)

    bundles['descripcion'] = (
        bundles['producto_a'] + " + " + bundles['producto_b'] +
        " (Lift: " + bundles['lift'].round(2).astype(str) +
        ", Conf: " + (bundles['avg_confidence'] * 100).round(1).astype(str) + "%)"
    )

    return bundles[['producto_a', 'producto_b', 'support_pair', 'avg_confidence',
```

```
natural_bundles = identify_natural_bundles(association_rules, min_confidence=0.7,
```

---

7. Análisis de Cestas Típicas

7.1. Patrones de Cestas Frecuentes

```
def find_frequent_basket_patterns(baskets_df, min_freq=5):
    """
    Identifica patrones de cestas que se repiten frecuentemente
    """
    # Crear signature de cesta (conjunto ordenado de productos)
    basket_signatures = baskets_df.groupby('basket_id')['producto_clean'].apply(
        lambda x: tuple(sorted(set(x)))
    ).reset_index()
    basket_signatures.columns = ['basket_id', 'signature']

    # Frecuencia de cada patrón
    pattern_freq = basket_signatures['signature'].value_counts().reset_index()
    pattern_freq.columns = ['productos', 'frecuencia']

    # Filtrar por frecuencia mínima y más de 1 producto
    pattern_freq = pattern_freq[
        (pattern_freq['frecuencia'] >= min_freq) &
        (pattern_freq['productos'].apply(len) > 1)
    ]

    # Formatear
    pattern_freq['num_productos'] = pattern_freq['productos'].apply(len)
    pattern_freq['productos_str'] = pattern_freq['productos'].apply(lambda x: ',

    return pattern_freq.sort_values('frecuencia', ascending=False)

frequent_patterns = find_frequent_basket_patterns(baskets_df, min_freq=5)
```

Output Schema

Product Pairs (Association Rules)

```
{
    'producto_a': str,
    'producto_b': str,
    'freq_pares': int,
    'support_pair': float,  # 0-1
```

```
    'support_a': float,
    'support_b': float,
    'confidence_a_to_b': float,  # 0-1
    'confidence_b_to_a': float,  # 0-1
    'lift': float,  # >0, idealmente >1
    'conviction_a_to_b': float,
    'conviction_b_to_a': float
}
```

Strong Rules (Directional)

```
{
    'antecedent': str,  # Si compra esto...
    'consequent': str,  # ...probablemente compre esto
    'support_pair': float,
    'confidence': float,  # 0-1
    'lift': float,
    'conviction': float,
    'regla': str  # Descripción legible
}
```

Natural Bundles

```
{
    'producto_a': str,
    'producto_b': str,
    'support_pair': float,
    'avg_confidence': float,
    'lift': float,
    'bundle_score': float,
    'descripcion': str
}
```

Implementación Completa

```
def calculate_market_basket_model(df_enriched,
                                  time_window_hours=4,
                                  min_support=0.01,
                                  min_confidence=0.3,
                                  min_lift=1.2):
    """
    Calcula análisis completo de Market Basket
```

```
    Returns:
    --------
    dict con baskets, product_support, association_rules, strong_rules, bundles
    """
    print("1. Creando cestas...")
    baskets_df = create_baskets(df_enriched, time_window_hours)

    print("2. Analizando cestas...")
    basket_stats, basket_summary = analyze_baskets(baskets_df)

    print("3. Calculando support individual...")
    product_support = calculate_individual_support(baskets_df)

    print("4. Calculando asociaciones...")
    association_rules = calculate_association_metrics(baskets_df, min_support)

    print("5. Filtrando reglas fuertes...")
    strong_rules = filter_strong_rules(association_rules, min_confidence, min_lif

    print("6. Identificando bundles naturales...")
    natural_bundles = identify_natural_bundles(association_rules)

    print("7. Creando matriz de recomendaciones...")
    recommendation_matrix = create_recommendation_matrix(strong_rules)

    print("8. Identificando patrones frecuentes...")
    frequent_patterns = find_frequent_basket_patterns(baskets_df)

    return {
        'baskets': baskets_df,
        'basket_stats': basket_stats,
        'basket_summary': basket_summary,
        'product_support': product_support,
        'association_rules': association_rules,
        'strong_rules': strong_rules,
        'natural_bundles': natural_bundles,
        'recommendation_matrix': recommendation_matrix,
        'frequent_patterns': frequent_patterns
    }

# Uso
market_basket = calculate_market_basket_model(
    df_enriched,
    time_window_hours=4,
    min_support=0.01,
    min_confidence=0.3,
```

```
    min_lift=1.2
)
```

Interpretación y Uso

1. Para Cross-Selling

```
# Cuando cliente agrega producto al carrito, mostrar recomendaciones
producto_en_carrito = "PRODUCTO_A"
sugerencias = get_product_recommendations(producto_en_carrito, market_basket['str
print(f"También te puede interesar: {sugerencias['producto_recomendado'].tolist()
```

2. Para Creación de Combos

```
# Top 5 combos sugeridos
top_bundles = market_basket['natural_bundles'].head(5)
print("Combos sugeridos para crear:")
for idx, row in top_bundles.iterrows():
    print(f"  - {row['producto_a']} + {row['producto_b']} (Score: {row['bundle_sc
```

3. Para Layout de Tienda

```
# Productos que deberían estar cerca físicamente
high_lift_pairs = market_basket['association_rules'][
    market_basket['association_rules']['lift'] > 2.0
][['producto_a', 'producto_b', 'lift']].head(10)

print("Productos que deberían estar cerca en tienda:")
print(high_lift_pairs)
```

Validaciones de Calidad

```
def validate_market_basket(market_basket):
    """Valida calidad del análisis de market basket"""
    checks = {
        'total_baskets': len(market_basket['baskets']['basket_id'].unique()),
        'multi_item_baskets_pct': market_basket['basket_summary']['pct_cestas_mul
        'total_association_rules': len(market_basket['association_rules']),
        'strong_rules': len(market_basket['strong_rules']),
        'natural_bundles': len(market_basket['natural_bundles']),
        'avg_lift': market_basket['association_rules']['lift'].mean()
    }
```

```
    # Warnings
    if checks['multi_item_baskets_pct'] < 20:
        print("⚠️  Menos del 20% de cestas tienen múltiples productos")

    if checks['strong_rules'] == 0:
        print("⚠️  No se encontraron reglas de asociación fuertes")

    if checks['avg_lift'] < 1.1:
        print("⚠️  Lift promedio bajo - asociaciones débiles")

    return checks
```

Dependencies

Input Requirements:

- ✅ Output de Modelo 1 (Transaction Enriched)
- ✅ Campos de hora (inith, initm) para definir cestas

External Libraries:

- pandas
- numpy
- itertools (combinations)

Configuration:

- `time_window_hours`: Ventana de tiempo para agrupar transacciones en cestas
- `min_support`: Support mínimo para considerar pares
- `min_confidence`: Confidence mínima para reglas
- `min_lift`: Lift mínimo para asociaciones significativas

Notas

- Market Basket requiere suficientes transacciones multi-producto (idealmente >20% del total)
- Lift > 1.2 es un umbral conservador para asociaciones significativas
- Para negocios con muchos productos, considerar análisis por categorías primero
- Actualizar análisis mensualmente o cuando cambien significativamente los productos