Modelo 6: Business Overview Model (KPI Dashboard)

Descripción del Modelo

Este modelo agrega y sintetiza métricas clave de **todos los modelos anteriores** para crear una vista ejecutiva del negocio. Proporciona KPIs principales, alertas críticas y resumen de salud del negocio. Es el "tablero de control" que consolida insights de todos los análisis.

Input

Outputs de Modelos 1-5:

- Modelo 1: Transaction Enriched
- Modelo 2: Product-Level Metrics
- Modelo 3: Customer-Level Metrics
- Modelo 4: Time-Period Metrics
- Modelo 5: Market Basket Analysis

Aggregation Level

Business-wide (nivel negocio completo)

Features Soportados

- **B6** - Dashboard de Indicadores Clave (KPIs)
- Base para **C2** - Explicaciones Inteligentes
- Base para **C4** - Recomendaciones Automáticas
- Base para **C6** - Coach de Negocios

Estructura del Dashboard

Secciones Principales:

1. **Vista General** - KPIs principales del período
2. **Salud Financiera** - Ingresos, márgenes, rentabilidad
3. **Desempeño de Productos** - Top/bottom performers
4. **Salud de Clientes** - Segmentos, retención, churn
5. **Inventario** - Alertas críticas
6. **Tendencias** - Dirección del negocio
7. **Oportunidades** - Acciones recomendadas

Cálculos del Modelo

1. Vista General (Overview)

1.1. KPIs Principales del Período Actual

```
def calculate_overview_kpis(df_enriched, monthly_metrics, fecha_inicio=None, fech
    """
    Calcula KPIs principales para el período especificado

    Parameters:
    -----------
    df_enriched : DataFrame
        Transactions enriched
    monthly_metrics : DataFrame
        Monthly aggregated metrics
    fecha_inicio : date, optional
        Inicio del período (default: últimos 30 días)
    fecha_fin : date, optional
        Fin del período (default: hoy)
    """
    from datetime import date, timedelta

    if fecha_fin is None:
        fecha_fin = date.today()
    if fecha_inicio is None:
        fecha_inicio = fecha_fin - timedelta(days=30)

    # Filtrar transacciones del período
    mask = (df_enriched['fecha'].dt.date >= fecha_inicio) & (df_enriched['fecha']
    df_period = df_enriched[mask]

    # Mismo período año anterior (para comparación YoY)
    fecha_inicio_yoy = fecha_inicio - timedelta(days=365)
    fecha_fin_yoy = fecha_fin - timedelta(days=365)
    mask_yoy = (df_enriched['fecha'].dt.date >= fecha_inicio_yoy) & (df_enriched[
    df_period_yoy = df_enriched[mask_yoy]

    overview = {
        'periodo': {
            'inicio': fecha_inicio,
            'fin': fecha_fin,
            'dias': (fecha_fin - fecha_inicio).days + 1
        },

        # Ventas
        'ingresos_periodo': df_period['total'].sum(),
        'ingresos_periodo_yoy': df_period_yoy['total'].sum(),
        'num_transacciones': len(df_period),
        'num_transacciones_yoy': len(df_period_yoy),
```

```
        # Productos
        'unidades_vendidas': df_period['cantidad'].sum(),
        'productos_unicos_vendidos': df_period['producto_clean'].nunique(),

        # Clientes
        'clientes_unicos': df_period['customer_id_clean'].nunique(),
        'clientes_unicos_yoy': df_period_yoy['customer_id_clean'].nunique(),

        # Financiero
        'costo_total': df_period['costo'].sum(),
        'margen_total': df_period['margen_absoluto'].sum(),
        'margen_porcentaje': (df_period['margen_absoluto'].sum() / df_period['tot

        # Promedios
        'ticket_promedio': df_period['total'].mean(),
        'ticket_promedio_yoy': df_period_yoy['total'].mean() if len(df_period_yoy
        'ventas_por_dia': df_period['total'].sum() / ((fecha_fin - fecha_inicio).
    }

    # Calcular cambios
    overview['cambio_ingresos_yoy_pct'] = (
        (overview['ingresos_periodo'] - overview['ingresos_periodo_yoy']) /
        overview['ingresos_periodo_yoy'] * 100
    ) if overview['ingresos_periodo_yoy'] > 0 else 0

    overview['cambio_clientes_yoy_pct'] = (
        (overview['clientes_unicos'] - overview['clientes_unicos_yoy']) /
        overview['clientes_unicos_yoy'] * 100
    ) if overview['clientes_unicos_yoy'] > 0 else 0

    overview['cambio_ticket_yoy_pct'] = (
        (overview['ticket_promedio'] - overview['ticket_promedio_yoy']) /
        overview['ticket_promedio_yoy'] * 100
    ) if overview['ticket_promedio_yoy'] > 0 else 0

    return overview

overview_kpis = calculate_overview_kpis(df_enriched, monthly_metrics)
```

---

2. Salud Financiera

2.1. Métricas Financieras Clave

```python
def calculate_financial_health(overview_kpis, monthly_metrics):
    """Calcula indicadores de salud financiera"""

    # Últimos 3 meses para tendencias
    last_3m = monthly_metrics.tail(3)

    financial_health = {
        # Período actual
        'ingresos_periodo': overview_kpis['ingresos_periodo'],
        'margen_periodo': overview_kpis['margen_total'],
        'margen_porcentaje': overview_kpis['margen_porcentaje'],

        # Tendencias
        'ingresos_tendencia': last_3m['cambio_mom_porcentual'].mean(),
        'margen_tendencia': (last_3m['margen_mes'] / last_3m['ingresos_mes'] * 10

        # Comparación YoY
        'crecimiento_yoy': overview_kpis['cambio_ingresos_yoy_pct'],

        # Proyección
        'proyeccion_mensual': overview_kpis['ventas_por_dia'] * 30,
        'proyeccion_anual': overview_kpis['ventas_por_dia'] * 365,
    }

    # Score de salud (0-100)
    health_score = 50  # Base

    # +20 si está creciendo YoY
    if financial_health['crecimiento_yoy'] > 0:
        health_score += 20
    elif financial_health['crecimiento_yoy'] < -10:
        health_score -= 20

    # +15 si margen es saludable (>20%)
    if financial_health['margen_porcentaje'] > 20:
        health_score += 15
    elif financial_health['margen_porcentaje'] < 10:
        health_score -= 15

    # +15 si tendencia es positiva
    if financial_health['ingresos_tendencia'] > 0:
        health_score += 15
    elif financial_health['ingresos_tendencia'] < -5:
        health_score -= 15

    financial_health['health_score'] = max(0, min(100, health_score))
```

```
    financial_health['health_categoria'] = categorize_health(health_score)

    return financial_health

def categorize_health(score):
    """Categoriza health score"""
    if score >= 80:
        return 'Excelente'
    elif score >= 60:
        return 'Bueno'
    elif score >= 40:
        return 'Aceptable'
    elif score >= 20:
        return 'Preocupante'
    else:
        return 'Crítico'

financial_health = calculate_financial_health(overview_kpis, monthly_metrics)
```

---

3. Desempeño de Productos

3.1. Top & Bottom Performers

```
def analyze_product_performance(product_model):
    """Analiza desempeño de productos"""

    # Top 10 por ingresos
    top_products = product_model.nlargest(10, 'total_ingresos')[
        ['producto', 'total_ingresos', 'margen_total_porcentaje',
         'pareto_categoria', 'velocidad_categoria']
    ]

    # Bottom 10 por rotación (excluyendo productos sin stock)
    bottom_products = product_model[
        product_model.get('stock_actual', 0) > 0
    ].nsmallest(10, 'rotacion_inventario')[
        ['producto', 'total_ingresos', 'dias_sin_venta',
         'rotacion_inventario', 'valor_inventario_lento']
    ]

    # Productos con alertas
    productos_alerta_stock = product_model[
        product_model['alerta_stock_bajo'] == 1
    ][['producto', 'stock_actual', 'dias_stock_restante']]
```

```
    productos_sin_movimiento = product_model[
        product_model['alerta_sin_movimiento'] == 1
    ][['producto', 'dias_sin_venta', 'valor_inventario_lento']]

    # Resumen
    summary = {
        'total_productos': len(product_model),
        'productos_activos': (product_model['dias_sin_venta'] <= 30).sum(),
        'productos_categoria_a': (product_model['pareto_categoria'] == 'A - Top 8
        'valor_top_10': top_products['total_ingresos'].sum(),
        'pct_valor_top_10': (top_products['total_ingresos'].sum() / product_model
        'productos_criticos_stock': len(productos_alerta_stock),
        'productos_sin_movimiento': len(productos_sin_movimiento),
        'valor_inventario_lento_total': product_model['valor_inventario_lento'].s
    }

    return {
        'top_products': top_products,
        'bottom_products': bottom_products,
        'alerta_stock': productos_alerta_stock,
        'sin_movimiento': productos_sin_movimiento,
        'summary': summary
    }

product_performance = analyze_product_performance(product_model)
```

---

4. Salud de Clientes

4.1. Análisis de Segmentos y Churn

```
def analyze_customer_health(customer_model):
    """Analiza salud de base de clientes"""

    # Distribución de segmentos
    segment_distribution = customer_model['segmento'].value_counts().to_dict()

    # Clientes de alto valor
    high_value_customers = customer_model[
        customer_model['pareto_categoria'] == 'A - Top 80%'
    ]

    # Clientes en riesgo
    at_risk = customer_model[
        customer_model['riesgo_categoria'].isin(['Alto', 'Medio'])
    ].sort_values('prioridad_contacto', ascending=False)
```

```
    # Métricas clave
    summary = {
        'total_clientes': len(customer_model),
        'clientes_activos': (customer_model['segmento'] != 'Dormidos').sum(),
        'clientes_champions': segment_distribution.get('Champions', 0),
        'clientes_leales': segment_distribution.get('Leales', 0),
        'clientes_nuevos': segment_distribution.get('Nuevos', 0),
        'clientes_en_riesgo_alto': (customer_model['riesgo_categoria'] == 'Alto')
        'clientes_en_riesgo_medio': (customer_model['riesgo_categoria'] == 'Medio
        'valor_en_riesgo': at_risk['total_gastado'].sum(),
        'pct_clientes_activos': ((customer_model['segmento'] != 'Dormidos').sum()
        'ltv_promedio': customer_model['ltv_estimado'].mean(),
        'ticket_promedio_general': customer_model['ticket_promedio'].mean(),
    }

    # Top 10 clientes por valor
    top_customers = customer_model.nlargest(10, 'total_gastado')[
        ['customer_id', 'customer_name', 'total_gastado', 'total_compras',
         'segmento', 'riesgo_categoria']
    ]

    # Clientes prioritarios para contactar
    priority_contacts = at_risk[at_risk['debe_contactar'] == 1].head(20)[
        ['customer_id', 'customer_name', 'total_gastado', 'dias_desde_compra_espe
         'riesgo_categoria', 'prioridad_contacto']
    ]

    return {
        'segment_distribution': segment_distribution,
        'top_customers': top_customers,
        'at_risk_customers': at_risk,
        'priority_contacts': priority_contacts,
        'summary': summary
    }

customer_health = analyze_customer_health(customer_model)
```

---

5. Alertas Críticas Consolidadas

5.1. Sistema de Alertas Priorizado

```
def generate_critical_alerts(product_performance, customer_health, financial_heal
    """Genera lista priorizada de alertas críticas"""
```

```
    alerts = []

    # ALERTAS DE INVENTARIO (Prioridad Alta)
    if product_performance['summary']['productos_criticos_stock'] > 0:
        alerts.append({
            'prioridad': 'Alta',
            'categoria': 'Inventario',
            'alerta': f"{product_performance['summary']['productos_criticos_stock
            'accion': 'Revisar y pedir inventario urgente',
            'productos': product_performance['alerta_stock']['producto'].tolist()
        })

    # ALERTAS DE PRODUCTOS SIN MOVIMIENTO (Prioridad Media)
    if product_performance['summary']['productos_sin_movimiento'] > 0:
        valor_lento = product_performance['summary']['valor_inventario_lento_tota
        alerts.append({
            'prioridad': 'Media',
            'categoria': 'Inventario',
            'alerta': f"{product_performance['summary']['productos_sin_movimiento
            'accion': 'Considerar promoción o liquidación',
            'productos': product_performance['sin_movimiento']['producto'].tolist
        })

    # ALERTAS DE CLIENTES EN RIESGO (Prioridad Alta si hay valor significativo)
    if customer_health['summary']['clientes_en_riesgo_alto'] > 0:
        alerts.append({
            'prioridad': 'Alta',
            'categoria': 'Clientes',
            'alerta': f"{customer_health['summary']['clientes_en_riesgo_alto']} c
            'accion': 'Contactar inmediatamente para retención',
            'valor_en_riesgo': customer_health['summary']['valor_en_riesgo'],
            'clientes': customer_health['priority_contacts']['customer_name'].to
        })

    # ALERTAS DE TENDENCIA NEGATIVA (Prioridad Alta)
    if financial_health['ingresos_tendencia'] < -10:
        alerts.append({
            'prioridad': 'Alta',
            'categoria': 'Financiero',
            'alerta': f"Ventas cayendo {abs(financial_health['ingresos_tendencia'
            'accion': 'Analizar causas y tomar acción correctiva',
            'tendencia': financial_health['ingresos_tendencia']
        })

    # ALERTAS DE MARGEN BAJO (Prioridad Media)
    if financial_health['margen_porcentaje'] < 15:
        alerts.append({
```

```
            'prioridad': 'Media',
            'categoria': 'Financiero',
            'alerta': f"Margen bajo: {financial_health['margen_porcentaje']:.1f}%
            'accion': 'Revisar costos y estrategia de precios'
        })

    # ALERTA DE POCOS CLIENTES ACTIVOS (Prioridad Media)
    pct_activos = customer_health['summary']['pct_clientes_activos']
    if pct_activos < 50:
        alerts.append({
            'prioridad': 'Media',
            'categoria': 'Clientes',
            'alerta': f"Solo {pct_activos:.1f}% de clientes activos",
            'accion': 'Campaña de reactivación de clientes dormidos'
        })

    # Ordenar por prioridad
    priority_order = {'Alta': 1, 'Media': 2, 'Baja': 3}
    alerts_sorted = sorted(alerts, key=lambda x: priority_order[x['prioridad']])

    return alerts_sorted

critical_alerts = generate_critical_alerts(
    product_performance,
    customer_health,
    financial_health,
    overview_kpis
)
```

---

6. Oportunidades Identificadas

6.1. Recomendaciones Accionables

```
def identify_opportunities(product_performance, customer_health, market_basket):
    """Identifica oportunidades de mejora del negocio"""

    opportunities = []

    # OPORTUNIDAD: Clientes dormidos de alto valor
    high_value_dormant = customer_health['at_risk_customers'][
        (customer_health['at_risk_customers']['total_gastado'] > customer_health[
        (customer_health['at_risk_customers']['riesgo_categoria'].isin(['Alto', '
    ]

    if len(high_value_dormant) > 0:
```

```
        valor_potencial = high_value_dormant['total_gastado'].sum() * 0.3  # 30%
        opportunities.append({
            'tipo': 'Reactivación',
            'oportunidad': f"Recuperar {len(high_value_dormant)} clientes de alto
            'valor_estimado': valor_potencial,
            'accion': 'Campaña de email/WhatsApp con descuento u oferta especial'
            'prioridad': 'Alta',
            'clientes': high_value_dormant['customer_id'].tolist()
        })

    # OPORTUNIDAD: Cross-sell de productos asociados
    if 'strong_rules' in market_basket and len(market_basket['strong_rules']) > 0
        top_rules = market_basket['strong_rules'].head(5)
        opportunities.append({
            'tipo': 'Cross-sell',
            'oportunidad': f"Implementar recomendaciones de productos (identifica
            'accion': 'Sugerir productos complementarios en punto de venta',
            'ejemplos': [
                f"{row['antecedent']} → {row['consequent']} ({row['confidence']*1
                for _, row in top_rules.iterrows()
            ],
            'prioridad': 'Media'
        })

    # OPORTUNIDAD: Bundles naturales
    if 'natural_bundles' in market_basket and len(market_basket['natural_bundles'
        top_bundles = market_basket['natural_bundles'].head(3)
        opportunities.append({
            'tipo': 'Bundles',
            'oportunidad': f"Crear {len(market_basket['natural_bundles'])} combos
            'accion': 'Ofrecer descuento en combo vs compra individual',
            'bundles_sugeridos': [
                f"{row['producto_a']} + {row['producto_b']}"
                for _, row in top_bundles.iterrows()
            ],
            'prioridad': 'Media'
        })

    # OPORTUNIDAD: Productos con buena rotación pero stock insuficiente
    productos_populares = product_performance['top_products'][
        product_performance['top_products']['velocidad_categoria'] == 'rápida'
    ]
    opportunities.append({
        'tipo': 'Optimización Inventario',
        'oportunidad': f"Aumentar stock de productos de alta rotación",
        'accion': 'Incrementar punto de reorden para evitar quiebres',
        'productos': productos_populares['producto'].tolist(),
```

```
        'prioridad': 'Alta'
    })

    # OPORTUNIDAD: Liquidar inventario lento
    if product_performance['summary']['valor_inventario_lento_total'] > 0:
        opportunities.append({
            'tipo': 'Liquidación',
            'oportunidad': f"Liberar ${product_performance['summary']['valor_inve
            'accion': 'Promoción 2x1, descuentos o consignación',
            'productos': product_performance['sin_movimiento']['producto'].tolist
            'prioridad': 'Media'
        })

    return opportunities

opportunities = identify_opportunities(product_performance, customer_health, mark
```

---

7. Resumen Ejecutivo

7.1. Dashboard Completo

```
def generate_executive_dashboard(overview_kpis, financial_health, product_perform
                                 customer_health, critical_alerts, opportunities
    """Genera dashboard ejecutivo completo"""

    dashboard = {
        'fecha_generacion': date.today(),
        'periodo_analisis': overview_kpis['periodo'],

        # SECCIÓN 1: Resumen de Salud
        'salud_general': {
            'score': financial_health['health_score'],
            'categoria': financial_health['health_categoria'],
            'alertas_criticas': len([a for a in critical_alerts if a['prioridad']
            'oportunidades_identificadas': len(opportunities)
        },

        # SECCIÓN 2: KPIs Principales
        'kpis_principales': {
            'ingresos': {
                'valor': overview_kpis['ingresos_periodo'],
                'cambio_yoy': overview_kpis['cambio_ingresos_yoy_pct'],
                'tendencia': 'positiva' if overview_kpis['cambio_ingresos_yoy_pct
            },
            'margen': {
```

```
                'porcentaje': financial_health['margen_porcentaje'],
                'total': overview_kpis['margen_total']
            },
            'transacciones': {
                'cantidad': overview_kpis['num_transacciones'],
                'ticket_promedio': overview_kpis['ticket_promedio'],
                'cambio_ticket_yoy': overview_kpis['cambio_ticket_yoy_pct']
            },
            'clientes': {
                'total': customer_health['summary']['total_clientes'],
                'activos': customer_health['summary']['clientes_activos'],
                'pct_activos': customer_health['summary']['pct_clientes_activos']
                'nuevos': customer_health['summary']['clientes_nuevos']
            }
        },

        # SECCIÓN 3: Productos
        'productos': {
            'total': product_performance['summary']['total_productos'],
            'activos': product_performance['summary']['productos_activos'],
            'top_10_productos': product_performance['top_products']['producto'].t
            'valor_top_10_pct': product_performance['summary']['pct_valor_top_10'
        },

        # SECCIÓN 4: Alertas Críticas
        'alertas': critical_alerts,

        # SECCIÓN 5: Oportunidades
        'oportunidades': opportunities,

        # SECCIÓN 6: Proyecciones
        'proyecciones': {
            'mensual': financial_health['proyeccion_mensual'],
            'anual': financial_health['proyeccion_anual']
        }
    }

    return dashboard

executive_dashboard = generate_executive_dashboard(
    overview_kpis, financial_health, product_performance,
    customer_health, critical_alerts, opportunities
)
```

Output Schema

```
{
    'fecha_generacion': date,
    'periodo_analisis': {
        'inicio': date,
        'fin': date,
        'dias': int
    },

    'salud_general': {
        'score': int,  # 0-100
        'categoria': str,  # Excelente/Bueno/Aceptable/Preocupante/Crítico
        'alertas_criticas': int,
        'oportunidades_identificadas': int
    },

    'kpis_principales': {
        'ingresos': {
            'valor': float,
            'cambio_yoy': float,
            'tendencia': str
        },
        'margen': {
            'porcentaje': float,
            'total': float
        },
        'transacciones': {...},
        'clientes': {...}
    },

    'productos': {...},
    'alertas': [...],  # Lista de alertas priorizadas
    'oportunidades': [...],  # Lista de oportunidades
    'proyecciones': {...}
}
```

Implementación Completa

```
def calculate_business_overview_model(df_enriched, product_model, customer_model,
                                      monthly_metrics, market_basket=None):
    """
    Genera dashboard ejecutivo completo del negocio

    Returns:
    --------
```

```
    dict con executive_dashboard y todos los componentes
    """
    print("Generando Business Overview Dashboard...")

    # 1. Overview KPIs
    print("  1/7 Calculando KPIs generales...")
    overview_kpis = calculate_overview_kpis(df_enriched, monthly_metrics)

    # 2. Financial Health
    print("  2/7 Analizando salud financiera...")
    financial_health = calculate_financial_health(overview_kpis, monthly_metrics)

    # 3. Product Performance
    print("  3/7 Analizando desempeño de productos...")
    product_performance = analyze_product_performance(product_model)

    # 4. Customer Health
    print("  4/7 Analizando salud de clientes...")
    customer_health = analyze_customer_health(customer_model)

    # 5. Critical Alerts
    print("  5/7 Generando alertas críticas...")
    critical_alerts = generate_critical_alerts(
        product_performance, customer_health,
        financial_health, overview_kpis
    )

    # 6. Opportunities
    print("  6/7 Identificando oportunidades...")
    opportunities = identify_opportunities(
        product_performance, customer_health,
        market_basket if market_basket else {}
    )

    # 7. Executive Dashboard
    print("  7/7 Consolidando dashboard ejecutivo...")
    executive_dashboard = generate_executive_dashboard(
        overview_kpis, financial_health, product_performance,
        customer_health, critical_alerts, opportunities
    )

    print("✓ Dashboard generado exitosamente")

    return {
        'executive_dashboard': executive_dashboard,
        'overview_kpis': overview_kpis,
        'financial_health': financial_health,
```

```
        'product_performance': product_performance,
        'customer_health': customer_health,
        'critical_alerts': critical_alerts,
        'opportunities': opportunities
    }

# Uso
business_overview = calculate_business_overview_model(
    df_enriched,
    product_model,
    customer_model,
    monthly_metrics,
    market_basket
)
```

Visualización del Dashboard

```
def print_executive_summary(dashboard):
    """Imprime resumen ejecutivo en formato legible"""

    print("=" * 80)
    print("DASHBOARD EJECUTIVO - RESUMEN DE NEGOCIO".center(80))
    print("=" * 80)
    print(f"\nPeríodo: {dashboard['periodo_analisis']['inicio']} a {dashboard['pe
    print(f"Generado: {dashboard['fecha_generacion']}")

    # Salud General
    print(f"\n{'SALUD GENERAL':=^80}")
    print(f"Score: {dashboard['salud_general']['score']}/100 - {dashboard['salud_
    print(f"Alertas Críticas: {dashboard['salud_general']['alertas_criticas']}")
    print(f"Oportunidades: {dashboard['salud_general']['oportunidades_identificad

    # KPIs
    print(f"\n{'KPIS PRINCIPALES':=^80}")
    kpis = dashboard['kpis_principales']
    print(f"Ingresos: ${kpis['ingresos']['valor']:,.0f} ({kpis['ingresos']['cambi
    print(f"Margen: {kpis['margen']['porcentaje']:.1f}% (${kpis['margen']['total'
    print(f"Transacciones: {kpis['transacciones']['cantidad']:,} (Ticket: ${kpis[
    print(f"Clientes: {kpis['clientes']['activos']}/{kpis['clientes']['total']} a

    # Alertas
    if dashboard['alertas']:
        print(f"\n{'ALERTAS CRÍTICAS':=^80}")
        for i, alert in enumerate(dashboard['alertas'][:5], 1):
            print(f"{i}. [{alert['prioridad']}] {alert['alerta']}")
```

```
            print(f"   → {alert['accion']}")

    # Oportunidades
    if dashboard['oportunidades']:
        print(f"\n{'TOP OPORTUNIDADES':=^80}")
        for i, opp in enumerate(dashboard['oportunidades'][:3], 1):
            print(f"{i}. [{opp['tipo']}] {opp['oportunidad']}")
            if 'valor_estimado' in opp:
                print(f"   Valor potencial: ${opp['valor_estimado']:,.0f}")
            print(f"   → {opp['accion']}")

    print("\n" + "=" * 80)

# Uso
print_executive_summary(business_overview['executive_dashboard'])
```

Dependencies

Input Requirements:

- ✅ Outputs de Modelos 1-5

External Libraries:

- pandas
- numpy

Notas

- Este modelo debe ejecutarse **después** de todos los otros modelos
- Frecuencia recomendada: **Semanal** para monitoreo, **Mensual** para reportes ejecutivos
- Personalizar umbrales de alertas según tipo de negocio
- Dashboard puede exportarse a PDF/HTML para compartir con stakeholders