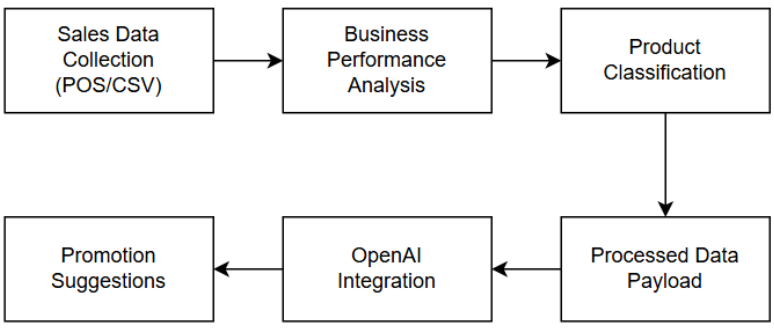
AI-Driven Promotion Suggestion Generation

Executive Summary

This technical document outlines the implementation of our AI-driven promotion suggestion system, which leverages sales data, business context, and feedback history to generate targeted promotional recommendations. The system employs data analysis techniques to classify products and assess their performance.

System Architecture



Key Components

1. **Data Collection Module**: Integrates with Square POS and provides CSV upload capabilities
2. **Performance Analysis Engine**: Analyzes sales data to identify trends and product performance
3. **Product Classification Module**: Categorizes products based on performance metrics
4. **AI Integration Layer**: Interfaces with OpenAI for suggestion generation
5. **Suggestion Management System**: Stores, displays, and tracks suggestion lifecycle

Product Performance Analysis

Data Processing Algorithm

The system employs a multi-step algorithm to analyze product performance:

```
def _get_products_performance(self, business, days=30):
    """
    Analyzes sales data to classify products based on performance and sales
trends.
    """
    # Get recent sales data within timeframe
    start_date = datetime.now(timezone('UTC')) - timedelta(days)
    end_date = datetime.now(timezone('UTC'))
    sales_data = SalesDataPoint.objects.filter(
        business_id=business.id,
        date__range=[start_date, end_date]
    )

    # Aggregate by product: revenue and units sold
    grouped = sales_data.values('product_name') \
        .annotate(
            total_revenue=Sum('revenue'),
            total_units=Sum('units_sold')
        )

    # Determine top and bottom percentiles
    total = len(grouped)
    top_10_percent = max(int(total * 0.1), 1)
    bottom_10_percent = max(int(total * 0.1), 1)

    # Sort products by total revenue
    sorted_products = sorted(
        grouped,
        key=lambda x: x['total_revenue'],
        reverse=True
    )

    # Extract product names for trend analysis
    product_names = [product['product_name'] for product in sorted_products]

    # Map product sales data by date for trend analysis
    product_data_map = {
        name: sales_data.filter(product_name=name).order_by('-date')
        for name in product_names
    }

    # Calculate sales trend for each product
    product_trends = {
        name: self._calculate_trend(product_data_map[name])
        for name in product_data_map
    }

    # Assign performance category and trend
    for i, product in enumerate(sorted_products):
        trend = product_trends[product['product_name']]
```

```
        # Classify based on percentile
        if i < top_10_percent:
            product['category'] = 'top_10_percent'
        elif i >= total - bottom_10_percent:
            product['category'] = 'bottom_10_percent'
        else:
            product['category'] = 'average'

        # Add trend classification
        product['trend'] = trend
```

Trend Analysis with Exponential Moving Average

For trend detection, we implemented an Exponential Moving Average (EMA) algorithm to give more weight to recent sales data:

```
def _calculate_trend(self, product_data, days=14, smoothing_factor=0.1,
threshold=0.05):
    """
    Calculates sales trend using Exponential Moving Average (EMA).

    Parameters:
    smoothing_factor: Weight for recent data (0-1)
    threshold: Maximum difference to consider "flat" trend
    """
    smoothing_factor = Decimal(smoothing_factor)
    revenues = product_data.order_by('-date').values_list('revenue', flat=True)
[:days]

    if len(revenues) < days:
        return 'flat'

    # Calculate Exponential Moving Average
    ema = revenues[0]
    for revenue in revenues[1:]:
        ema = (smoothing_factor * revenue) + ((1 - smoothing_factor) * ema)

    # Classify trend based on comparison with current revenue
    if abs(revenues[0] - ema) <= threshold:
        return 'flat'
    elif revenues[0] > ema:
        return 'upward'
    else:
        return 'downward'
```

Algorithm Selection Rationale

We evaluated several algorithms for trend analysis:

Algorithm	Advantages	Disadvantages	Selected
Simple Moving Average	Easy to implement, intuitive	Equal weight to all data points, lags behind rapid changes	No
Linear Regression	Good for consistent trends	Sensitive to outliers, computationally intensive	No
Exponential Moving Average	Prioritizes recent data, handles volatility well, low computation cost	Requires tuning of smoothing factor	Yes
ARIMA	Comprehensive time-series forecasting	Complex implementation, requires larger datasets, computationally expensive	No

EMA was selected because:

1. It gives higher weight to recent sales data, which is more relevant for marketing decisions
2. It's computationally efficient, allowing quick analysis even with large datasets
3. It handles the volatility typical in restaurant sales data
4. It requires minimal historical data to produce meaningful results
5. It's straightforward to implement and maintain

AI Integration for Suggestion Generation

Input Payload Structure

The system constructs a structured payload for the OpenAI API containing:

1. **Product Performance Data**:
    - Product name, total revenue, units sold
    - Performance category (top 10%, bottom 10%, average)
    - Sales trend (upward, downward, flat)
    - Date range of analysis
2. **Business Context**:
    - Business name and type
    - Target customer demographics
    - Business atmosphere/vibe
3. **Feedback History** (covered briefly, detailed in Document 3):
    - Previous suggestion feedback for continuous improvement

```
ai_input_payload = {
    "products_performance": {
        "start_date": overall_start_date,
```

```
        "end_date": overall_end_date,
        "products": sorted_products
    },
    "context_data": {
        "name": business.name,
        "type": business.category,
        "target_customers": business.target_customers,
        "vibe": business.vibe
    },
    "feedback_history": feedback_context
}
```

AI Prompt Engineering

The system uses a carefully crafted prompt template to guide the model in generating effective promotion suggestions:

```
prompt = f"""
Based on the provided business context, product performance (from {start_date} to
{end_date}), and general consumer insights, generate 3 distinct promotion
suggestions for each of the top 10 best-selling and bottom 10 low-performing
products (exclude average-performing ones entirely).

Each suggestion must follow this structure:
[
    {{
        "product_name": [<product_name_1>, <product_name_2>, ...],  # List ALL
products mentioned in the description
        "category": ["discount", "bundle", "social"],  // Select ALL applicable
categories (not just one).
        "title": <suggestion_title>,
        "description": <suggestion_description>
            # Clearly state which product is best-selling or low-performing
            # Include actual confirmed numbers: original prices, discounted rates,
new prices, bundle totals, sold quantities during n-days, happy hour time etc.
            # Explain why the promotion works for this product (e.g., bundling a
low-performer with a top-seller to increase exposure)
            # Support reasoning with general customer insights (e.g., popular
combinations, taste profiles, typical behavior)
    }}
]

After generating all suggestions, return only a single JSON array containing the
top 5 most promising promotions (out of all generated) that are most likely to
increase revenue, based on product performance, customer behavior, and general
insights.

Business Information:
- Business Name: {context_data['name']}
- Business Type: {context_data['type']}
- Target Customers: {context_data['target_customers']}
```

```
- Business Vibe: {context_data['vibe']}

Products Performance: {products_performance}
{feedback_context}
"""
```

Model Selection

We chose the OpenAI GPT-4o Mini model for several reasons:

1. **Cost-effectiveness**: Balances performance with operational costs
2. **Structured output capability**: Reliable JSON format generation
3. **Context length**: Sufficient for our product data and business context
4. **Reasoning depth**: Sophisticated enough to generate nuanced marketing suggestions
5. **Prompt consistency**: Reliable adherence to our structured format requirements

Frontend Implementation

The promotion suggestions are presented to users through an interactive card-based UI that:

1. Highlights product performance categories with color-coding
2. Provides detailed pricing information from Square integration
3. Allows users to create promotions directly from suggestions
4. Enables feedback collection for continuous improvement

Conclusion

The AI-driven promotion suggestion system represents a sophisticated integration of data analysis and artificial intelligence to provide actionable marketing insights. By categorizing products based on performance and analyzing sales trends, the system generates targeted promotion suggestions that help businesses optimize their marketing strategy, increase sales, and improve overall business performance.

The combination of robust backend analysis, thoughtful machine learning algorithm selection, and intuitive frontend presentation creates a powerful tool for data-driven marketing decisions in the food and beverage industry.