perplexity

Zomato Restaurant Data Analysis - SQL Project Files

I'll break down the comprehensive SQL project into separate downloadable files for easy implementation and management.

File Structure Overview

Here are the individual SQL files you can download and execute separately:

1. Database Schema Creation (schema_creation.sql)

```
-- ========================================================
-- Zomato Restaurant Database Schema Creation
-- File: schema_creation.sql
-- Description: Creates all necessary tables for the Zomato restaurant analysis project
-- ========================================================

-- Create database
CREATE DATABASE IF NOT EXISTS zomato_analytics;
USE zomato_analytics;

-- Drop existing tables if they exist (in reverse dependency order)
DROP TABLE IF EXISTS pricing_multipliers;
DROP TABLE IF EXISTS elasticity_coefficients;
DROP TABLE IF EXISTS order_items;
DROP TABLE IF EXISTS orders;
DROP TABLE IF EXISTS menu_items;
DROP TABLE IF EXISTS customers;
DROP TABLE IF EXISTS restaurants;

-- Create restaurants table
CREATE TABLE restaurants (
    restaurant_id INT PRIMARY KEY AUTO_INCREMENT,
    restaurant_name VARCHAR(255) NOT NULL,
    locality VARCHAR(100) NOT NULL,
    cuisine_type VARCHAR(100),
    restaurant_category ENUM('Budget', 'Mid-Range', 'Premium', 'Fine Dining') NOT NULL,
    average_rating DECIMAL(3,2),
    total_reviews INT DEFAULT 0,
    opening_hours VARCHAR(50),
    delivery_available BOOLEAN DEFAULT TRUE,
    latitude DECIMAL(10, 8),
    longitude DECIMAL(11, 8),
    established_date DATE,
    seating_capacity INT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
```

```
);

-- Create customers table
CREATE TABLE customers (
    customer_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_name VARCHAR(255),
    email VARCHAR(255) UNIQUE,
    phone VARCHAR(15),
    address TEXT,
    registration_date DATE,
    customer_segment ENUM('Premium', 'High Value', 'Frequent', 'Price Sensitive', 'Occasi
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Create menu_items table
CREATE TABLE menu_items (
    item_id INT PRIMARY KEY AUTO_INCREMENT,
    restaurant_id INT,
    item_name VARCHAR(255) NOT NULL,
    item_category VARCHAR(100),
    base_price DECIMAL(8,2) NOT NULL,
    current_price DECIMAL(8,2) NOT NULL,
    preparation_time_minutes INT,
    availability_status BOOLEAN DEFAULT TRUE,
    popularity_score DECIMAL(3,2),
    cost_of_goods_sold DECIMAL(8,2),
    last_price_update TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (restaurant_id) REFERENCES restaurants(restaurant_id) ON DELETE CASCADE
);

-- Create orders table
CREATE TABLE orders (
    order_id INT PRIMARY KEY AUTO_INCREMENT,
    restaurant_id INT,
    customer_id INT,
    order_datetime TIMESTAMP NOT NULL,
    order_value DECIMAL(10,2) NOT NULL,
    discount_applied DECIMAL(5,2) DEFAULT 0,
    delivery_fee DECIMAL(5,2) DEFAULT 0,
    payment_method ENUM('Credit Card', 'Debit Card', 'Digital Wallet', 'Cash') NOT NULL,
    order_status ENUM('Placed', 'Confirmed', 'Preparing', 'Out for Delivery', 'Delivered'
    delivery_time_minutes INT,
    weather_condition VARCHAR(50),
    special_event_flag BOOLEAN DEFAULT FALSE,
    peak_hour_flag BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (restaurant_id) REFERENCES restaurants(restaurant_id),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);

-- Create order_items table
CREATE TABLE order_items (
    order_item_id INT PRIMARY KEY AUTO_INCREMENT,
    order_id INT,
    item_id INT,
```

```
    quantity INT NOT NULL,
    unit_price DECIMAL(8,2) NOT NULL,
    total_price DECIMAL(8,2) NOT NULL,
    special_instructions TEXT,
    FOREIGN KEY (order_id) REFERENCES orders(order_id) ON DELETE CASCADE,
    FOREIGN KEY (item_id) REFERENCES menu_items(item_id)
);

-- Create elasticity_coefficients table for storing calculated price elasticity
CREATE TABLE elasticity_coefficients (
    elasticity_id INT PRIMARY KEY AUTO_INCREMENT,
    item_category VARCHAR(100),
    restaurant_category VARCHAR(50),
    price_elasticity DECIMAL(6,3),
    confidence_level DECIMAL(3,2),
    sample_size INT,
    calculation_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    INDEX idx_category (item_category, restaurant_category)
);

-- Create pricing_multipliers table for dynamic pricing factors
CREATE TABLE pricing_multipliers (
    multiplier_id INT PRIMARY KEY AUTO_INCREMENT,
    condition_type ENUM('Weather', 'Event', 'Season', 'Inventory', 'Time') NOT NULL,
    condition_value VARCHAR(100) NOT NULL,
    restaurant_category VARCHAR(50),
    item_category VARCHAR(100),
    multiplier_factor DECIMAL(4,3) NOT NULL,
    effective_start_time TIME,
    effective_end_time TIME,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Create indexes for better performance
CREATE INDEX idx_orders_datetime ON orders(order_datetime);
CREATE INDEX idx_orders_restaurant ON orders(restaurant_id);
CREATE INDEX idx_orders_customer ON orders(customer_id);
CREATE INDEX idx_menu_restaurant ON menu_items(restaurant_id);
CREATE INDEX idx_restaurants_locality ON restaurants(locality);
CREATE INDEX idx_restaurants_category ON restaurants(restaurant_category);

-- Add triggers for automatic updates
DELIMITER //

CREATE TRIGGER update_menu_price_timestamp
BEFORE UPDATE ON menu_items
FOR EACH ROW
BEGIN
    IF NEW.current_price != OLD.current_price THEN
        SET NEW.last_price_update = CURRENT_TIMESTAMP;
    END IF;
END //

CREATE TRIGGER update_peak_hour_flag
BEFORE INSERT ON orders
```

```
FOR EACH ROW
BEGIN
    DECLARE hour_of_day INT;
    SET hour_of_day = HOUR(NEW.order_datetime);

    IF (hour_of_day BETWEEN 12 AND 14) OR (hour_of_day BETWEEN 19 AND 22) THEN
        SET NEW.peak_hour_flag = TRUE;
    END IF;
END //

DELIMITER ;

-- Success message
SELECT 'Database schema created successfully!' as Status;
```

2. Sample Data Population (sample_data.sql)

```
-- ======================================================
-- Sample Data Population Script
-- File: sample_data.sql
-- Description: Inserts sample data for testing and demonstration
-- ======================================================

USE zomato_analytics;

-- Insert sample restaurants
INSERT INTO restaurants (restaurant_name, locality, cuisine_type, restaurant_category, av
('Spice Garden', 'Koramangala', 'Indian', 'Mid-Range', 4.2, 1250, '11:00-23:00', 80, '201
('Pizza Paradise', 'Indiranagar', 'Italian', 'Budget', 4.0, 890, '12:00-24:00', 60, '2019
('The Royal Feast', 'UB City Mall', 'Multi-Cuisine', 'Premium', 4.5, 2100, '12:00-23:30',
('Burger Street', 'Whitefield', 'American', 'Budget', 3.8, 560, '11:00-23:00', 45, '2020-
('Golden Dragon', 'Brigade Road', 'Chinese', 'Mid-Range', 4.3, 1680, '12:00-22:30', 90, '
('Café Mocha', 'Jayanagar', 'Continental', 'Mid-Range', 4.1, 920, '08:00-22:00', 50, '201
('Fine Dine Deluxe', 'MG Road', 'French', 'Fine Dining', 4.7, 850, '18:00-23:00', 40, '20
('Street Food Corner', 'Banashankari', 'Indian Street Food', 'Budget', 3.9, 1200, '10:00-
('Sushi Zen', 'Whitefield', 'Japanese', 'Premium', 4.4, 750, '12:00-22:00', 70, '2020-08-
('Taco Fiesta', 'Koramangala', 'Mexican', 'Mid-Range', 4.0, 640, '12:00-23:00', 55, '2021

-- Insert sample customers
INSERT INTO customers (customer_name, email, phone, address, registration_date) VALUES
('Rajesh Kumar', 'rajesh.kumar@email.com', '9876543210', 'HSR Layout, Bangalore', '2021-0
('Priya Sharma', 'priya.sharma@email.com', '9876543211', 'Koramangala, Bangalore', '2021-
('Amit Patel', 'amit.patel@email.com', '9876543212', 'Indiranagar, Bangalore', '2021-03-1
('Sneha Reddy', 'sneha.reddy@email.com', '9876543213', 'Jayanagar, Bangalore', '2021-04-0
('Vikram Singh', 'vikram.singh@email.com', '9876543214', 'Whitefield, Bangalore', '2021-0
('Anita Gupta', 'anita.gupta@email.com', '9876543215', 'MG Road, Bangalore', '2021-06-18'
('Rohit Mehta', 'rohit.mehta@email.com', '9876543216', 'Brigade Road, Bangalore', '2021-0
('Kavya Nair', 'kavya.nair@email.com', '9876543217', 'Banashankari, Bangalore', '2021-08-
('Suresh Jain', 'suresh.jain@email.com', '9876543218', 'Electronic City, Bangalore', '202
('Meera Iyer', 'meera.iyer@email.com', '9876543219', 'Malleshwaram, Bangalore', '2021-10-

-- Insert sample menu items
INSERT INTO menu_items (restaurant_id, item_name, item_category, base_price, current_pric
-- Spice Garden (restaurant_id: 1)
(1, 'Butter Chicken', 'Main Course', 320.00, 350.00, 25, 4.5, 180.00),
```

```
(1, 'Paneer Tikka', 'Appetizer', 280.00, 280.00, 20, 4.2, 150.00),
(1, 'Biryani', 'Main Course', 380.00, 420.00, 35, 4.7, 200.00),
(1, 'Naan', 'Bread', 80.00, 85.00, 10, 4.0, 30.00),
(1, 'Lassi', 'Beverage', 120.00, 120.00, 5, 3.8, 50.00),

-- Pizza Paradise (restaurant_id: 2)
(2, 'Margherita Pizza', 'Main Course', 250.00, 280.00, 20, 4.3, 120.00),
(2, 'Pepperoni Pizza', 'Main Course', 350.00, 380.00, 22, 4.5, 180.00),
(2, 'Garlic Bread', 'Appetizer', 150.00, 150.00, 15, 4.1, 60.00),
(2, 'Caesar Salad', 'Salad', 200.00, 210.00, 10, 3.9, 80.00),
(2, 'Coke', 'Beverage', 60.00, 65.00, 2, 3.5, 25.00),

-- The Royal Feast (restaurant_id: 3)
(3, 'Grilled Salmon', 'Main Course', 850.00, 920.00, 30, 4.6, 450.00),
(3, 'Lamb Chops', 'Main Course', 950.00, 1050.00, 35, 4.8, 520.00),
(3, 'Truffle Pasta', 'Main Course', 650.00, 680.00, 25, 4.4, 320.00),
(3, 'Wine Selection', 'Beverage', 800.00, 850.00, 5, 4.2, 400.00),
(3, 'Chocolate Soufflé', 'Dessert', 420.00, 450.00, 40, 4.7, 180.00);

-- Insert more menu items for other restaurants (abbreviated for space)
INSERT INTO menu_items (restaurant_id, item_name, item_category, base_price, current_pric
(4, 'Classic Burger', 'Main Course', 180.00, 200.00, 15, 4.0, 90.00),
(4, 'Chicken Wings', 'Appetizer', 220.00, 240.00, 20, 4.2, 120.00),
(5, 'Hakka Noodles', 'Main Course', 240.00, 260.00, 18, 4.1, 120.00),
(5, 'Sweet & Sour Chicken', 'Main Course', 320.00, 340.00, 25, 4.3, 180.00),
(6, 'Grilled Sandwich', 'Main Course', 160.00, 170.00, 12, 3.9, 80.00),
(6, 'Coffee', 'Beverage', 120.00, 130.00, 8, 4.0, 40.00);

-- Insert sample orders with realistic patterns
INSERT INTO orders (restaurant_id, customer_id, order_datetime, order_value, discount_app
(1, 1, '2024-01-15 13:30:00', 720.00, 50.00, 30.00, 'Digital Wallet', 'Delivered', 35, 'C
(2, 2, '2024-01-15 19:45:00', 650.00, 0.00, 25.00, 'Credit Card', 'Delivered', 28, 'Clear
(3, 3, '2024-01-16 20:15:00', 1850.00, 100.00, 0.00, 'Credit Card', 'Delivered', 45, 'Lig
(1, 4, '2024-01-16 12:20:00', 540.00, 30.00, 30.00, 'Debit Card', 'Delivered', 32, 'Clear
(4, 5, '2024-01-17 18:30:00', 420.00, 20.00, 35.00, 'Digital Wallet', 'Delivered', 25, 'C
(2, 1, '2024-01-17 21:00:00', 580.00, 40.00, 25.00, 'Digital Wallet', 'Delivered', 30, 'C
(5, 6, '2024-01-18 19:20:00', 600.00, 0.00, 30.00, 'Credit Card', 'Delivered', 40, 'Clear
(3, 7, '2024-01-18 20:45:00', 2100.00, 150.00, 0.00, 'Credit Card', 'Delivered', 50, 'Cle
(6, 8, '2024-01-19 11:15:00', 300.00, 15.00, 20.00, 'Cash', 'Delivered', 20, 'Sunny', FAL
(1, 9, '2024-01-19 13:45:00', 800.00, 60.00, 30.00, 'Digital Wallet', 'Delivered', 38, 'H

-- Insert corresponding order items
INSERT INTO order_items (order_id, item_id, quantity, unit_price, total_price) VALUES
(1, 1, 1, 350.00, 350.00),
(1, 3, 1, 420.00, 420.00),
(2, 6, 1, 280.00, 280.00),
(2, 7, 1, 380.00, 380.00),
(3, 11, 1, 920.00, 920.00),
(3, 12, 1, 1050.00, 1050.00),
(4, 1, 1, 350.00, 350.00),
(4, 4, 2, 85.00, 170.00),
(5, 16, 1, 200.00, 200.00),
(5, 17, 1, 240.00, 240.00);

-- Insert pricing multipliers
INSERT INTO pricing_multipliers (condition_type, condition_value, restaurant_category, it
```

```
('Weather', 'Heavy Rain', 'All', 'All', 1.150, '00:00:00', '23:59:59'),
('Weather', 'Extreme Heat', 'All', 'All', 0.950, '12:00:00', '16:00:00'),
('Event', 'Cricket Match', 'All', 'All', 1.200, '19:00:00', '23:00:00'),
('Event', 'Festival', 'Premium', 'All', 1.300, '18:00:00', '23:59:59'),
('Event', 'Festival', 'Fine Dining', 'All', 1.400, '18:00:00', '23:59:59'),
('Season', 'Peak Summer', 'All', 'Beverage', 1.100, '10:00:00', '18:00:00'),
('Time', 'Peak Hours', 'All', 'All', 1.080, '19:00:00', '22:00:00'),
('Inventory', 'Low Stock', 'All', 'All', 1.250, '00:00:00', '23:59:59');

-- Insert sample elasticity coefficients
INSERT INTO elasticity_coefficients (item_category, restaurant_category, price_elasticity
('Main Course', 'Budget', -1.8, 0.85, 150),
('Main Course', 'Mid-Range', -1.2, 0.78, 200),
('Main Course', 'Premium', -0.8, 0.82, 120),
('Main Course', 'Fine Dining', -0.5, 0.90, 80),
('Appetizer', 'Budget', -2.1, 0.75, 100),
('Appetizer', 'Mid-Range', -1.5, 0.80, 130),
('Appetizer', 'Premium', -1.0, 0.85, 90),
('Beverage', 'Budget', -2.5, 0.70, 200),
('Beverage', 'Mid-Range', -1.8, 0.75, 180),
('Beverage', 'Premium', -1.2, 0.80, 110),
('Dessert', 'Premium', -0.9, 0.88, 60),
('Dessert', 'Fine Dining', -0.6, 0.92, 45);

SELECT 'Sample data inserted successfully!' as Status;
```

3. Basic Analysis Queries (basic_analysis.sql)

```
-- ======================================================
-- Basic Analysis Queries
-- File: basic_analysis.sql
-- Description: Fundamental analysis queries for restaurant performance
-- ======================================================

USE zomato_analytics;

-- 1. Restaurant Performance Overview
SELECT
    'Restaurant Performance Overview' as Analysis_Type;

SELECT
    r.restaurant_name,
    r.locality,
    r.restaurant_category,
    r.average_rating,
    COUNT(DISTINCT o.order_id) as total_orders,
    ROUND(AVG(o.order_value), 2) as avg_order_value,
    ROUND(SUM(o.order_value), 2) as total_revenue,
    ROUND(AVG(o.delivery_time_minutes), 1) as avg_delivery_time,
    COUNT(DISTINCT o.customer_id) as unique_customers
FROM restaurants r
LEFT JOIN orders o ON r.restaurant_id = o.restaurant_id
    AND o.order_status = 'Delivered'
    AND o.order_datetime >= DATE_SUB(CURDATE(), INTERVAL 30 DAY)
GROUP BY r.restaurant_id, r.restaurant_name, r.locality, r.restaurant_category, r.average
```

```
ORDER BY total_revenue DESC;

-- 2. Peak Hour Analysis
SELECT
    'Peak Hour Analysis' as Analysis_Type;

WITH hourly_analysis AS (
    SELECT
        HOUR(order_datetime) as hour_of_day,
        DAYOFWEEK(order_datetime) as day_of_week,
        COUNT(*) as order_count,
        ROUND(AVG(order_value), 2) as avg_order_value,
        ROUND(SUM(order_value), 2) as total_revenue,
        ROUND(AVG(delivery_time_minutes), 1) as avg_delivery_time
    FROM orders
    WHERE order_status = 'Delivered'
    AND order_datetime >= DATE_SUB(CURDATE(), INTERVAL 30 DAY)
    GROUP BY HOUR(order_datetime), DAYOFWEEK(order_datetime)
)
SELECT
    hour_of_day,
    CASE
        WHEN day_of_week IN (1,7) THEN 'Weekend'
        ELSE 'Weekday'
    END as day_type,
    order_count,
    avg_order_value,
    total_revenue,
    avg_delivery_time,
    CASE
        WHEN order_count > (SELECT AVG(order_count) * 1.5 FROM hourly_analysis)
        THEN 'Peak'
        WHEN order_count > (SELECT AVG(order_count) FROM hourly_analysis)
        THEN 'Moderate'
        ELSE 'Low'
    END as demand_level
FROM hourly_analysis
ORDER BY day_of_week, hour_of_day;

-- 3. Locality Performance Analysis
SELECT
    'Locality Performance Analysis' as Analysis_Type;

SELECT
    r.locality,
    COUNT(DISTINCT r.restaurant_id) as restaurant_count,
    ROUND(AVG(r.average_rating), 2) as avg_locality_rating,
    COUNT(o.order_id) as total_orders,
    ROUND(AVG(o.order_value), 2) as avg_order_value,
    ROUND(SUM(o.order_value), 2) as total_revenue,
    ROUND(AVG(o.delivery_time_minutes), 1) as avg_delivery_time,
    COUNT(DISTINCT o.customer_id) as unique_customers,
    ROUND(COUNT(o.order_id) / COUNT(DISTINCT r.restaurant_id), 1) as orders_per_restaurar
    ROUND(STDDEV(o.order_value), 2) as order_value_variance
FROM restaurants r
LEFT JOIN orders o ON r.restaurant_id = o.restaurant_id
```

```
    AND o.order_datetime >= DATE_SUB(CURDATE(), INTERVAL 90 DAY)
    AND o.order_status = 'Delivered'
GROUP BY r.locality
HAVING restaurant_count >= 1
ORDER BY avg_order_value DESC;

-- 4. Menu Item Performance
SELECT
    'Menu Item Performance Analysis' as Analysis_Type;

SELECT
    r.restaurant_name,
    mi.item_category,
    mi.item_name,
    mi.base_price,
    mi.current_price,
    ROUND((mi.current_price - mi.base_price) / mi.base_price * 100, 2) as price_increase_
    COUNT(oi.order_item_id) as times_ordered,
    ROUND(SUM(oi.total_price), 2) as total_revenue,
    ROUND(AVG(oi.unit_price), 2) as avg_selling_price,
    mi.popularity_score
FROM menu_items mi
JOIN restaurants r ON mi.restaurant_id = r.restaurant_id
LEFT JOIN order_items oi ON mi.item_id = oi.item_id
LEFT JOIN orders o ON oi.order_id = o.order_id
    AND o.order_status = 'Delivered'
    AND o.order_datetime >= DATE_SUB(CURDATE(), INTERVAL 30 DAY)
WHERE mi.availability_status = TRUE
GROUP BY mi.item_id, r.restaurant_name, mi.item_category, mi.item_name,
         mi.base_price, mi.current_price, mi.popularity_score
ORDER BY total_revenue DESC;

-- 5. Customer Behavior Analysis
SELECT
    'Customer Behavior Analysis' as Analysis_Type;

SELECT
    CASE
        WHEN total_orders >= 10 THEN 'Frequent (10+)'
        WHEN total_orders >= 5 THEN 'Regular (5-9)'
        WHEN total_orders >= 2 THEN 'Occasional (2-4)'
        ELSE 'One-time (1)'
    END as customer_frequency,
    COUNT(*) as customer_count,
    ROUND(AVG(avg_order_value), 2) as avg_order_value,
    ROUND(AVG(total_spent), 2) as avg_total_spent,
    ROUND(AVG(avg_discount_used), 2) as avg_discount_usage
FROM (
    SELECT
        o.customer_id,
        COUNT(*) as total_orders,
        AVG(o.order_value) as avg_order_value,
        SUM(o.order_value) as total_spent,
        AVG(o.discount_applied) as avg_discount_used
    FROM orders o
    WHERE o.order_status = 'Delivered'
```

```
    AND o.order_datetime >= DATE_SUB(CURDATE(), INTERVAL 180 DAY)
    GROUP BY o.customer_id
) customer_metrics
GROUP BY customer_frequency
ORDER BY customer_count DESC;

-- 6. Payment Method Analysis
SELECT
    'Payment Method Analysis' as Analysis_Type;

SELECT
    payment_method,
    COUNT(*) as transaction_count,
    ROUND(AVG(order_value), 2) as avg_order_value,
    ROUND(SUM(order_value), 2) as total_revenue,
    ROUND(AVG(discount_applied), 2) as avg_discount,
    ROUND(COUNT(*) * 100.0 / (SELECT COUNT(*) FROM orders WHERE order_status = 'Delivered
FROM orders
WHERE order_status = 'Delivered'
AND order_datetime >= DATE_SUB(CURDATE(), INTERVAL 30 DAY)
GROUP BY payment_method
ORDER BY transaction_count DESC;

-- 7. Weather Impact Analysis
SELECT
    'Weather Impact Analysis' as Analysis_Type;

SELECT
    weather_condition,
    COUNT(*) as order_count,
    ROUND(AVG(order_value), 2) as avg_order_value,
    ROUND(AVG(delivery_time_minutes), 1) as avg_delivery_time,
    ROUND(AVG(discount_applied), 2) as avg_discount_applied
FROM orders
WHERE order_status = 'Delivered'
AND weather_condition IS NOT NULL
AND order_datetime >= DATE_SUB(CURDATE(), INTERVAL 60 DAY)
GROUP BY weather_condition
ORDER BY order_count DESC;

SELECT 'Basic analysis queries completed successfully!' as Status;
```

4. Price Elasticity Analysis (price_elasticity.sql)

```
-- ====================================================
-- Price Elasticity Analysis
-- File: price_elasticity.sql
-- Description: Advanced queries for calculating and analyzing price elasticity
-- ====================================================

USE zomato_analytics;

-- 1. Price Change Impact Analysis
SELECT 'Price Change Impact Analysis' as Analysis_Type;
```

```
WITH price_changes AS (
    SELECT
        mi.item_id,
        mi.restaurant_id,
        mi.item_name,
        mi.item_category,
        r.restaurant_category,
        mi.base_price,
        mi.current_price,
        ROUND((mi.current_price - mi.base_price) / mi.base_price * 100, 2) as price_chang
        mi.last_price_update
    FROM menu_items mi
    JOIN restaurants r ON mi.restaurant_id = r.restaurant_id
    WHERE mi.current_price != mi.base_price
),
demand_analysis AS (
    SELECT
        pc.item_id,
        pc.restaurant_id,
        pc.item_category,
        pc.restaurant_category,
        pc.price_change_percent,
        pc.base_price,
        pc.current_price,
        COUNT(CASE WHEN o.order_datetime < pc.last_price_update THEN 1 END) as orders_bef
        COUNT(CASE WHEN o.order_datetime >= pc.last_price_update THEN 1 END) as orders_af
        ROUND(AVG(CASE WHEN o.order_datetime < pc.last_price_update THEN oi.quantity END)
        ROUND(AVG(CASE WHEN o.order_datetime >= pc.last_price_update THEN oi.quantity END
    FROM price_changes pc
    JOIN order_items oi ON pc.item_id = oi.item_id
    JOIN orders o ON oi.order_id = o.order_id
    WHERE o.order_status = 'Delivered'
    AND o.order_datetime >= DATE_SUB(pc.last_price_update, INTERVAL 30 DAY)
    AND o.order_datetime <= DATE_ADD(pc.last_price_update, INTERVAL 30 DAY)
    GROUP BY pc.item_id, pc.restaurant_id, pc.item_category, pc.restaurant_category,
             pc.price_change_percent, pc.base_price, pc.current_price
)
SELECT
    item_category,
    restaurant_category,
    COUNT(*) as items_analyzed,
    ROUND(AVG(price_change_percent), 2) as avg_price_change,
    ROUND(AVG((orders_after - orders_before) / NULLIF(orders_before, 0) * 100), 2) as dem
    ROUND(AVG((((orders_after - orders_before) / NULLIF(orders_before, 0)) /
          NULLIF(price_change_percent / 100, 0))), 3) as price_elasticity,
    CASE
        WHEN AVG((((orders_after - orders_before) / NULLIF(orders_before, 0)) /
                  NULLIF(price_change_percent / 100, 0))) < -2 THEN 'Highly Elastic'
        WHEN AVG((((orders_after - orders_before) / NULLIF(orders_before, 0)) /
                  NULLIF(price_change_percent / 100, 0))) < -1 THEN 'Elastic'
        WHEN AVG((((orders_after - orders_before) / NULLIF(orders_before, 0)) /
                  NULLIF(price_change_percent / 100, 0))) < 0 THEN 'Inelastic'
        ELSE 'Positive Elasticity'
    END as elasticity_interpretation
FROM demand_analysis
WHERE orders_before > 0 AND orders_after > 0
```

```
GROUP BY item_category, restaurant_category
ORDER BY price_elasticity;

-- 2. Detailed Item-Level Elasticity
SELECT 'Detailed Item-Level Elasticity Analysis' as Analysis_Type;

WITH item_elasticity AS (
    SELECT
        r.restaurant_name,
        mi.item_name,
        mi.item_category,
        r.restaurant_category,
        mi.base_price,
        mi.current_price,
        (mi.current_price - mi.base_price) / mi.base_price * 100 as price_change_percent,
        COUNT(oi.order_item_id) as total_orders_recent,
        SUM(oi.quantity) as total_quantity_sold,
        ROUND(AVG(oi.unit_price), 2) as avg_realized_price,
        ROUND(SUM(oi.total_price), 2) as total_revenue
    FROM menu_items mi
    JOIN restaurants r ON mi.restaurant_id = r.restaurant_id
    LEFT JOIN order_items oi ON mi.item_id = oi.item_id
    LEFT JOIN orders o ON oi.order_id = o.order_id
        AND o.order_status = 'Delivered'
        AND o.order_datetime >= DATE_SUB(CURDATE(), INTERVAL 30 DAY)
    GROUP BY r.restaurant_name, mi.item_name, mi.item_category, r.restaurant_category,
             mi.base_price, mi.current_price
)
SELECT
    restaurant_name,
    item_name,
    item_category,
    restaurant_category,
    base_price,
    current_price,
    ROUND(price_change_percent, 2) as price_change_percent,
    total_orders_recent,
    total_quantity_sold,
    avg_realized_price,
    total_revenue,
    ROUND((current_price - base_price) * total_quantity_sold, 2) as additional_revenue_fr
    CASE
        WHEN price_change_percent > 0 AND total_orders_recent < 5 THEN 'Potentially Over-
        WHEN price_change_percent > 0 AND total_orders_recent >= 10 THEN 'Price Increase
        WHEN price_change_percent = 0 AND total_orders_recent >= 15 THEN 'Consider Price
        WHEN price_change_percent < 0 THEN 'Price Reduced'
        ELSE 'Monitor Performance'
    END as pricing_recommendation
FROM item_elasticity
WHERE current_price > 0
ORDER BY total_revenue DESC;

-- 3. Category-wise Elasticity Comparison
SELECT 'Category-wise Elasticity Comparison' as Analysis_Type;

SELECT
```

```
    mi.item_category,
    COUNT(DISTINCT mi.item_id) as total_items,
    ROUND(AVG(mi.base_price), 2) as avg_base_price,
    ROUND(AVG(mi.current_price), 2) as avg_current_price,
    ROUND(AVG((mi.current_price - mi.base_price) / mi.base_price * 100), 2) as avg_price_
    COUNT(oi.order_item_id) as total_orders,
    ROUND(AVG(oi.unit_price), 2) as avg_selling_price,
    ROUND(SUM(oi.total_price), 2) as category_revenue,
    -- Elasticity approximation based on stored coefficients
    ROUND(AVG(ec.price_elasticity), 3) as estimated_price_elasticity,
    CASE
        WHEN AVG(ec.price_elasticity) < -2 THEN 'Highly Price Sensitive'
        WHEN AVG(ec.price_elasticity) < -1 THEN 'Price Sensitive'
        WHEN AVG(ec.price_elasticity) < 0 THEN 'Moderately Price Sensitive'
        ELSE 'Price Insensitive'
    END as sensitivity_level
FROM menu_items mi
LEFT JOIN order_items oi ON mi.item_id = oi.item_id
LEFT JOIN orders o ON oi.order_id = o.order_id
    AND o.order_status = 'Delivered'
    AND o.order_datetime >= DATE_SUB(CURDATE(), INTERVAL 60 DAY)
LEFT JOIN elasticity_coefficients ec ON mi.item_category = ec.item_category
GROUP BY mi.item_category
ORDER BY category_revenue DESC;

-- 4. Restaurant Category Elasticity Analysis
SELECT 'Restaurant Category Elasticity Analysis' as Analysis_Type;

SELECT
    r.restaurant_category,
    COUNT(DISTINCT r.restaurant_id) as restaurant_count,
    COUNT(DISTINCT mi.item_id) as total_menu_items,
    ROUND(AVG(mi.current_price), 2) as avg_item_price,
    COUNT(o.order_id) as total_orders,
    ROUND(AVG(o.order_value), 2) as avg_order_value,
    ROUND(SUM(o.order_value), 2) as total_revenue,
    ROUND(AVG(o.discount_applied), 2) as avg_discount_used,
    ROUND(AVG(ec.price_elasticity), 3) as avg_price_elasticity,
    CASE
        WHEN AVG(ec.price_elasticity) BETWEEN -0.8 AND 0 THEN 'Premium Positioning Opport
        WHEN AVG(ec.price_elasticity) BETWEEN -1.5 AND -0.8 THEN 'Moderate Pricing Flexib
        WHEN AVG(ec.price_elasticity) < -1.5 THEN 'Price Sensitive Segment'
        ELSE 'Review Pricing Strategy'
    END as pricing_strategy_recommendation
FROM restaurants r
JOIN menu_items mi ON r.restaurant_id = mi.restaurant_id
LEFT JOIN orders o ON r.restaurant_id = o.restaurant_id
    AND o.order_status = 'Delivered'
    AND o.order_datetime >= DATE_SUB(CURDATE(), INTERVAL 90 DAY)
LEFT JOIN elasticity_coefficients ec ON r.restaurant_category = ec.restaurant_category
GROUP BY r.restaurant_category
ORDER BY avg_price_elasticity DESC;

-- 5. Time-based Elasticity Analysis
SELECT 'Time-based Elasticity Analysis' as Analysis_Type;
```

```
WITH hourly_pricing AS (
    SELECT
        HOUR(o.order_datetime) as hour_of_day,
        mi.item_category,
        COUNT(oi.order_item_id) as orders_count,
        ROUND(AVG(oi.unit_price), 2) as avg_price_paid,
        ROUND(AVG(mi.current_price), 2) as menu_price,
        ROUND(AVG(o.discount_applied), 2) as avg_discount,
        CASE
            WHEN HOUR(o.order_datetime) BETWEEN 12 AND 14 THEN 'Lunch Peak'
            WHEN HOUR(o.order_datetime) BETWEEN 19 AND 22 THEN 'Dinner Peak'
            WHEN HOUR(o.order_datetime) BETWEEN 15 AND 18 THEN 'Afternoon'
            WHEN HOUR(o.order_datetime) BETWEEN 8 AND 11 THEN 'Morning'
            ELSE 'Late Night'
        END as time_period
    FROM orders o
    JOIN order_items oi ON o.order_id = oi.order_id
    JOIN menu_items mi ON oi.item_id = mi.item_id
    WHERE o.order_status = 'Delivered'
    AND o.order_datetime >= DATE_SUB(CURDATE(), INTERVAL 30 DAY)
    GROUP BY HOUR(o.order_datetime), mi.item_category
)
SELECT
    time_period,
    item_category,
    SUM(orders_count) as total_orders,
    ROUND(AVG(avg_price_paid), 2) as avg_realized_price,
    ROUND(AVG(menu_price), 2) as avg_menu_price,
    ROUND(AVG(avg_discount), 2) as avg_discount_applied,
    ROUND((AVG(avg_price_paid) - AVG(menu_price)) / AVG(menu_price) * 100, 2) as price_va
    CASE
        WHEN SUM(orders_count) > 50 AND AVG(avg_discount) < 5 THEN 'Consider Peak Pricing
        WHEN SUM(orders_count) < 20 AND AVG(avg_discount) > 10 THEN 'Increase Promotions'
        ELSE 'Current Strategy OK'
    END as time_based_recommendation
FROM hourly_pricing
GROUP BY time_period, item_category
ORDER BY total_orders DESC;

SELECT 'Price elasticity analysis completed successfully!' as Status;
```

5. Dynamic Pricing Procedures (dynamic_pricing.sql)

```
-- ==================================================
-- Dynamic Pricing Procedures and Functions
-- File: dynamic_pricing.sql
-- Description: Stored procedures for real-time dynamic pricing calculations
-- ==================================================

USE zomato_analytics;

-- Drop existing procedures if they exist
DROP PROCEDURE IF EXISTS CalculateDynamicPricing;
DROP PROCEDURE IF EXISTS CalculateAdvancedDynamicPricing;
DROP PROCEDURE IF EXISTS UpdatePricingMultipliers;
```

```
DROP FUNCTION IF EXISTS GetDemandScore;
DROP FUNCTION IF EXISTS GetCompetitionFactor;

-- 1. Function to calculate current demand score
DELIMITER //
CREATE FUNCTION GetDemandScore(p_item_id INT)
RETURNS DECIMAL(3,2)
READS SQL DATA
DETERMINISTIC
BEGIN
    DECLARE v_demand_score DECIMAL(3,2) DEFAULT 0.5;
    DECLARE v_recent_orders INT DEFAULT 0;
    DECLARE v_max_recent_orders INT DEFAULT 1;

    -- Get recent orders for this item
    SELECT COUNT(*) INTO v_recent_orders
    FROM order_items oi
    JOIN orders o ON oi.order_id = o.order_id
    WHERE oi.item_id = p_item_id
    AND o.order_status = 'Delivered'
    AND o.order_datetime >= DATE_SUB(NOW(), INTERVAL 24 HOUR);

    -- Get maximum recent orders for any item in the same restaurant
    SELECT MAX(item_orders) INTO v_max_recent_orders
    FROM (
        SELECT COUNT(*) as item_orders
        FROM order_items oi2
        JOIN orders o2 ON oi2.order_id = o2.order_id
        JOIN menu_items mi ON oi2.item_id = mi.item_id
        WHERE mi.restaurant_id = (
            SELECT restaurant_id FROM menu_items WHERE item_id = p_item_id
        )
        AND o2.order_status = 'Delivered'
        AND o2.order_datetime >= DATE_SUB(NOW(), INTERVAL 24 HOUR)
        GROUP BY oi2.item_id
    ) sub;

    -- Calculate demand score (0-1 scale)
    IF v_max_recent_orders > 0 THEN
        SET v_demand_score = LEAST(1.0, v_recent_orders / v_max_recent_orders);
    END IF;

    RETURN v_demand_score;
END //

-- 2. Function to get competition factor
CREATE FUNCTION GetCompetitionFactor(p_restaurant_id INT, p_item_category VARCHAR(100))
RETURNS DECIMAL(3,2)
READS SQL DATA
DETERMINISTIC
BEGIN
    DECLARE v_competition_factor DECIMAL(3,2) DEFAULT 1.0;
    DECLARE v_locality VARCHAR(100);
    DECLARE v_competitor_count INT DEFAULT 0;
    DECLARE v_avg_competitor_price DECIMAL(8,2) DEFAULT 0;
    DECLARE v_our_avg_price DECIMAL(8,2) DEFAULT 0;
```

```
    -- Get restaurant locality
    SELECT locality INTO v_locality
    FROM restaurants
    WHERE restaurant_id = p_restaurant_id;

    -- Count competitors in same locality with same item category
    SELECT COUNT(DISTINCT r.restaurant_id) INTO v_competitor_count
    FROM restaurants r
    JOIN menu_items mi ON r.restaurant_id = mi.restaurant_id
    WHERE r.locality = v_locality
    AND r.restaurant_id != p_restaurant_id
    AND mi.item_category = p_item_category
    AND mi.availability_status = TRUE;

    -- Get average competitor price
    SELECT AVG(mi.current_price) INTO v_avg_competitor_price
    FROM restaurants r
    JOIN menu_items mi ON r.restaurant_id = mi.restaurant_id
    WHERE r.locality = v_locality
    AND r.restaurant_id != p_restaurant_id
    AND mi.item_category = p_item_category
    AND mi.availability_status = TRUE;

    -- Get our average price for the category
    SELECT AVG(current_price) INTO v_our_avg_price
    FROM menu_items
    WHERE restaurant_id = p_restaurant_id
    AND item_category = p_item_category
    AND availability_status = TRUE;

    -- Calculate competition factor
    IF v_avg_competitor_price > 0 AND v_our_avg_price > 0 THEN
        SET v_competition_factor = v_avg_competitor_price / v_our_avg_price;
        -- Normalize to reasonable range
        SET v_competition_factor = GREATEST(0.8, LEAST(1.2, v_competition_factor));
    END IF;

    RETURN v_competition_factor;
END //

-- 3. Basic Dynamic Pricing Procedure
CREATE PROCEDURE CalculateDynamicPricing(
    IN p_restaurant_id INT,
    IN p_target_profit_margin DECIMAL(5,2)
)
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE v_item_id INT;
    DECLARE v_item_name VARCHAR(255);
    DECLARE v_base_price DECIMAL(8,2);
    DECLARE v_current_price DECIMAL(8,2);
    DECLARE v_current_demand DECIMAL(3,2);
    DECLARE v_elasticity DECIMAL(5,3);
    DECLARE v_competition_factor DECIMAL(3,2);
    DECLARE v_recommended_price DECIMAL(8,2);
```

```
    DECLARE v_item_category VARCHAR(100);

    DECLARE item_cursor CURSOR FOR
        SELECT item_id, item_name, item_category, base_price, current_price
        FROM menu_items
        WHERE restaurant_id = p_restaurant_id
        AND availability_status = TRUE;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    -- Create temporary table for results
    DROP TEMPORARY TABLE IF EXISTS temp_pricing_results;
    CREATE TEMPORARY TABLE temp_pricing_results (
        item_id INT,
        item_name VARCHAR(255),
        item_category VARCHAR(100),
        current_price DECIMAL(8,2),
        recommended_price DECIMAL(8,2),
        price_adjustment DECIMAL(8,2),
        demand_score DECIMAL(3,2),
        competition_factor DECIMAL(3,2),
        expected_demand_change DECIMAL(5,2),
        profit_impact DECIMAL(8,2),
        recommendation_reason VARCHAR(500)
    );

    OPEN item_cursor;

    pricing_loop: LOOP
        FETCH item_cursor INTO v_item_id, v_item_name, v_item_category, v_base_price, v_c
        IF done THEN
            LEAVE pricing_loop;
        END IF;

        -- Get current demand score
        SET v_current_demand = GetDemandScore(v_item_id);

        -- Get price elasticity for item category
        SELECT COALESCE(AVG(price_elasticity), -1.2) INTO v_elasticity
        FROM elasticity_coefficients ec
        WHERE ec.item_category = v_item_category;

        -- Get competition factor
        SET v_competition_factor = GetCompetitionFactor(p_restaurant_id, v_item_category)

        -- Dynamic pricing formula
        SET v_recommended_price = v_base_price * (
            1 +
            (v_current_demand - 0.5) * 0.15 +        -- Demand adjustment (±7.5%)
            (v_competition_factor - 1) * 0.1 +        -- Competition adjustment (±2%)
            (ABS(v_elasticity) - 1) * 0.05            -- Elasticity adjustment
        );

        -- Ensure minimum profit margin
        SET v_recommended_price = GREATEST(v_recommended_price,
                                           v_base_price * (1 + p_target_profit_margin / 100
```

```
        -- Cap maximum price increase at 25%
        SET v_recommended_price = LEAST(v_recommended_price, v_base_price * 1.25);

        -- Round to nearest 5 for cleaner pricing
        SET v_recommended_price = ROUND(v_recommended_price / 5) * 5;

        INSERT INTO temp_pricing_results VALUES (
            v_item_id,
            v_item_name,
            v_item_category,
            v_current_price,
            v_recommended_price,
            v_recommended_price - v_current_price,
            v_current_demand,
            v_competition_factor,
            v_current_demand * v_elasticity *
                ((v_recommended_price - v_current_price) / v_current_price) * 100,
            (v_recommended_price - v_base_price) * v_current_demand * 10,
            CASE
                WHEN v_recommended_price > v_current_price THEN 'High demand, increase pr
                WHEN v_recommended_price < v_current_price THEN 'Low demand, decrease pri
                ELSE 'Optimal price maintained'
            END
        );

    END LOOP;

    CLOSE item_cursor;

    -- Return results
    SELECT * FROM temp_pricing_results
    ORDER BY profit_impact DESC;

    DROP TEMPORARY TABLE temp_pricing_results;
END //

-- 4. Advanced Dynamic Pricing with Environmental Factors
CREATE PROCEDURE CalculateAdvancedDynamicPricing(
    IN p_restaurant_id INT,
    IN p_current_weather VARCHAR(50),
    IN p_special_event VARCHAR(100),
    IN p_target_profit_margin DECIMAL(5,2)
)
BEGIN
    DECLARE v_restaurant_category VARCHAR(50);

    -- Get restaurant category
    SELECT restaurant_category INTO v_restaurant_category
    FROM restaurants
    WHERE restaurant_id = p_restaurant_id;

    SELECT
        mi.item_id,
        mi.item_name,
        mi.item_category,
```

```
        mi.current_price as base_dynamic_price,
        COALESCE(pm_weather.multiplier_factor, 1.0) as weather_multiplier,
        COALESCE(pm_event.multiplier_factor, 1.0) as event_multiplier,
        COALESCE(pm_time.multiplier_factor, 1.0) as time_multiplier,
        GetDemandScore(mi.item_id) as demand_score,
        GetCompetitionFactor(p_restaurant_id, mi.item_category) as competition_factor,
        -- Final price calculation
        ROUND(mi.current_price *
        COALESCE(pm_weather.multiplier_factor, 1.0) *
        COALESCE(pm_event.multiplier_factor, 1.0) *
        COALESCE(pm_time.multiplier_factor, 1.0) *
        (1 + (GetDemandScore(mi.item_id) - 0.5) * 0.1) / 5) * 5 as final_recommended_pric
        -- Price adjustment
        ROUND((mi.current_price *
         COALESCE(pm_weather.multiplier_factor, 1.0) *
         COALESCE(pm_event.multiplier_factor, 1.0) *
         COALESCE(pm_time.multiplier_factor, 1.0) *
         (1 + (GetDemandScore(mi.item_id) - 0.5) * 0.1) - mi.current_price), 2) as price_
        -- Reasoning
        CONCAT(
            CASE WHEN pm_weather.multiplier_factor > 1 THEN CONCAT('Weather boost (+', RO
            CASE WHEN pm_event.multiplier_factor > 1 THEN CONCAT('Event boost (+', ROUND(
            CASE WHEN GetDemandScore(mi.item_id) > 0.7 THEN 'High demand '
                 WHEN GetDemandScore(mi.item_id) < 0.3 THEN 'Low demand '
                 ELSE 'Normal demand ' END
        ) as pricing_rationale
    FROM menu_items mi
    JOIN restaurants r ON mi.restaurant_id = r.restaurant_id
    LEFT JOIN pricing_multipliers pm_weather ON pm_weather.condition_type = 'Weather'
        AND pm_weather.condition_value = p_current_weather
        AND pm_weather.is_active = TRUE
        AND (pm_weather.restaurant_category = r.restaurant_category OR pm_weather.restaur
        AND (pm_weather.effective_start_time <= TIME(NOW()) AND pm_weather.effective_end_
    LEFT JOIN pricing_multipliers pm_event ON pm_event.condition_type = 'Event'
        AND pm_event.condition_value = p_special_event
        AND pm_event.is_active = TRUE
        AND (pm_event.restaurant_category = r.restaurant_category OR pm_event.restaurant_
    LEFT JOIN pricing_multipliers pm_time ON pm_time.condition_type = 'Time'
        AND pm_time.condition_value = 'Peak Hours'
        AND pm_time.is_active = TRUE
        AND (pm_time.effective_start_time <= TIME(NOW()) AND pm_time.effective_end_time >
    WHERE mi.restaurant_id = p_restaurant_id
    AND mi.availability_status = TRUE
    ORDER BY price_adjustment DESC;
END //

-- 5. Procedure to update pricing multipliers
CREATE PROCEDURE UpdatePricingMultipliers()
BEGIN
    -- Update seasonal multipliers
    UPDATE pricing_multipliers
    SET is_active = CASE
        WHEN condition_value = 'Peak Summer' AND MONTH(NOW()) IN (4,5,6) THEN TRUE
        WHEN condition_value = 'Monsoon' AND MONTH(NOW()) IN (7,8,9) THEN TRUE
        WHEN condition_value = 'Winter' AND MONTH(NOW()) IN (12,1,2) THEN TRUE
        ELSE FALSE
```

```
    END
    WHERE condition_type = 'Season';

    -- Update time-based multipliers
    UPDATE pricing_multipliers
    SET is_active = TRUE
    WHERE condition_type = 'Time'
    AND condition_value = 'Peak Hours'
    AND TIME(NOW()) BETWEEN effective_start_time AND effective_end_time;

    SELECT 'Pricing multipliers updated successfully!' as Status;
END //

DELIMITER ;

-- Create a view for easy pricing dashboard access
CREATE OR REPLACE VIEW current_pricing_recommendations AS
SELECT
    r.restaurant_name,
    r.locality,
    mi.item_name,
    mi.item_category,
    mi.current_price,
    GetDemandScore(mi.item_id) as current_demand_score,
    CASE
        WHEN GetDemandScore(mi.item_id) > 0.8 THEN 'High Demand - Consider Price Increase
        WHEN GetDemandScore(mi.item_id) < 0.2 THEN 'Low Demand - Consider Promotion'
        ELSE 'Normal Demand - Monitor'
    END as demand_status,
    GetCompetitionFactor(mi.restaurant_id, mi.item_category) as competition_factor,
    CASE
        WHEN GetCompetitionFactor(mi.restaurant_id, mi.item_category) > 1.1 THEN 'Competi
        WHEN GetCompetitionFactor(mi.restaurant_id, mi.item_category) < 0.9 THEN 'Competi
        ELSE 'Competitive Pricing'
    END as competition_status
FROM menu_items mi
JOIN restaurants r ON mi.restaurant_id = r.restaurant_id
WHERE mi.availability_status = TRUE
ORDER BY r.restaurant_name, mi.item_category;

SELECT 'Dynamic pricing procedures created successfully!' as Status;
```

6. Dashboard and Monitoring Views (dashboard_views.sql)

```
-- ====================================================
-- Dashboard and Monitoring Views
-- File: dashboard_views.sql
-- Description: Views for real-time monitoring and dashboard displays
-- ====================================================

USE zomato_analytics;

-- Drop existing views if they exist
DROP VIEW IF EXISTS pricing_performance_dashboard;
DROP VIEW IF EXISTS real_time_metrics;
```

```
DROP VIEW IF EXISTS restaurant_performance_summary;
DROP VIEW IF EXISTS customer_insights_dashboard;
DROP VIEW IF EXISTS revenue_analytics_dashboard;

-- 1. Main Pricing Performance Dashboard
CREATE VIEW pricing_performance_dashboard AS
SELECT
    r.restaurant_id,
    r.restaurant_name,
    r.locality,
    r.restaurant_category,
    r.average_rating,
    -- Today's metrics
    COUNT(DISTINCT CASE WHEN DATE(o.order_datetime) = CURDATE() THEN o.order_id END) as t
    ROUND(AVG(CASE WHEN DATE(o.order_datetime) = CURDATE() THEN o.order_value END), 2) as
    ROUND(SUM(CASE WHEN DATE(o.order_datetime) = CURDATE() THEN o.order_value END), 2) as
    -- Week's metrics
    COUNT(DISTINCT CASE WHEN o.order_datetime >= DATE_SUB(CURDATE(), INTERVAL 7 DAY) THEN
    ROUND(AVG(CASE WHEN o.order_datetime >= DATE_SUB(CURDATE(), INTERVAL 7 DAY) THEN o.or
    ROUND(SUM(CASE WHEN o.order_datetime >= DATE_SUB(CURDATE(), INTERVAL 7 DAY) THEN o.or
    -- Pricing metrics
    COUNT(DISTINCT mi.item_id) as total_menu_items,
    ROUND(AVG(mi.current_price), 2) as avg_menu_price,
    ROUND(AVG((mi.current_price - mi.base_price) / mi.base_price * 100), 2) as avg_price_
    -- Performance indicators
    ROUND(AVG(o.delivery_time_minutes), 1) as avg_delivery_time,
    ROUND(AVG(o.discount_applied), 2) as avg_discount_given,
    COUNT(DISTINCT o.customer_id) as unique_customers_week,
    -- Capacity utilization
    CASE
        WHEN r.seating_capacity > 0 THEN
            ROUND((COUNT(DISTINCT CASE WHEN DATE(o.order_datetime) = CURDATE() THEN o.ord
        ELSE NULL
    END as capacity_utilization_percent,
    -- Status indicators
    CASE
        WHEN COUNT(DISTINCT CASE WHEN DATE(o.order_datetime) = CURDATE() THEN o.order_id
        WHEN AVG(CASE WHEN DATE(o.order_datetime) = CURDATE() THEN o.order_value END) >
             AVG(CASE WHEN o.order_datetime >= DATE_SUB(CURDATE(), INTERVAL 7 DAY) THEN o
        ELSE 'Below Average'
    END as performance_status
FROM restaurants r
LEFT JOIN orders o ON r.restaurant_id = o.restaurant_id
    AND o.order_status = 'Delivered'
    AND o.order_datetime >= DATE_SUB(CURDATE(), INTERVAL 7 DAY)
LEFT JOIN menu_items mi ON r.restaurant_id = mi.restaurant_id
    AND mi.availability_status = TRUE
WHERE r.delivery_available = TRUE
GROUP BY r.restaurant_id, r.restaurant_name, r.locality, r.restaurant_category,
         r.average_rating, r.seating_capacity
ORDER BY today_revenue DESC;

-- 2. Real-time Metrics View
CREATE VIEW real_time_metrics AS
SELECT
    -- Current hour metrics
```

```
    COUNT(CASE WHEN o.order_datetime >= DATE_SUB(NOW(), INTERVAL 1 HOUR) THEN 1 END) as c
    ROUND(AVG(CASE WHEN o.order_datetime >= DATE_SUB(NOW(), INTERVAL 1 HOUR) THEN o.order
    ROUND(SUM(CASE WHEN o.order_datetime >= DATE_SUB(NOW(), INTERVAL 1 HOUR) THEN o.order

    -- Today vs Yesterday comparison
    COUNT(CASE WHEN DATE(o.order_datetime) = CURDATE() THEN 1 END) as orders_today,
    COUNT(CASE WHEN DATE(o.order_datetime) = DATE_SUB(CURDATE(), INTERVAL 1 DAY) THEN 1 E
    ROUND(SUM(CASE WHEN DATE(o.order_datetime) = CURDATE() THEN o.order_value END), 2) as
    ROUND(SUM(CASE WHEN DATE(o.order_datetime) = DATE_SUB(CURDATE(), INTERVAL 1 DAY) THEN

    -- Growth calculations
    ROUND(
        (COUNT(CASE WHEN DATE(o.order_datetime) = CURDATE() THEN 1 END) -
         COUNT(CASE WHEN DATE(o.order_datetime) = DATE_SUB(CURDATE(), INTERVAL 1 DAY) THE
        NULLIF(COUNT(CASE WHEN DATE(o.order_datetime) = DATE_SUB(CURDATE(), INTERVAL 1 DA
    ) as order_growth_percent,

    ROUND(
        (SUM(CASE WHEN DATE(o.order_datetime) = CURDATE() THEN o.order_value END) -
         SUM(CASE WHEN DATE(o.order_datetime) = DATE_SUB(CURDATE(), INTERVAL 1 DAY) THEN
        NULLIF(SUM(CASE WHEN DATE(o.order_datetime) = DATE_SUB(CURDATE(), INTERVAL 1 DAY)
    ) as revenue_growth_percent,

    -- Current active restaurants
    COUNT(DISTINCT CASE WHEN o.order_datetime >= DATE_SUB(NOW(), INTERVAL 2 HOUR) THEN o.

    -- Average delivery time trend
    ROUND(AVG(CASE WHEN o.order_datetime >= DATE_SUB(NOW(), INTERVAL 2 HOUR) THEN o.deliv

    -- Peak hour indicator
    CASE
        WHEN HOUR(NOW()) BETWEEN 12 AND 14 THEN 'Lunch Peak'
        WHEN HOUR(NOW()) BETWEEN 19 AND 22 THEN 'Dinner Peak'
        WHEN HOUR(NOW()) BETWEEN 22 AND 24 THEN 'Late Night'
        ELSE 'Off Peak'
    END as current_period,

    NOW() as last_updated
FROM orders o
WHERE o.order_status = 'Delivered'
AND o.order_datetime >= DATE_SUB(CURDATE(), INTERVAL 2 DAY);

-- 3. Restaurant Performance Summary
CREATE VIEW restaurant_performance_summary AS
SELECT
    r.restaurant_id,
    r.restaurant_name,
    r.locality,
    r.restaurant_category,
    r.cuisine_type,
    r.average_rating,

    -- 30-day performance
    COUNT(DISTINCT o.order_id) as orders_30_days,
    ROUND(AVG(o.order_value), 2) as avg_order_value_30_days,
    ROUND(SUM(o.order_value), 2) as revenue_30_days,
```

```
    COUNT(DISTINCT o.customer_id) as unique_customers_30_days,
    ROUND(AVG(o.delivery_time_minutes), 1) as avg_delivery_time_30_days,

    -- Menu and pricing analysis
    COUNT(DISTINCT mi.item_id) as active_menu_items,
    ROUND(MIN(mi.current_price), 2) as min_item_price,
    ROUND(MAX(mi.current_price), 2) as max_item_price,
    ROUND(AVG(mi.current_price), 2) as avg_item_price,
    ROUND(AVG((mi.current_price - mi.base_price) / mi.base_price * 100), 2) as avg_markup

    -- Customer satisfaction indicators
    ROUND(AVG(CASE WHEN o.order_status = 'Delivered' THEN 1.0 ELSE 0.0 END) * 100, 2) as
    ROUND(AVG(o.discount_applied), 2) as avg_discount_offered,

    -- Growth metrics (comparing last 15 days vs previous 15 days)
    ROUND(
        (COUNT(CASE WHEN o.order_datetime >= DATE_SUB(CURDATE(), INTERVAL 15 DAY) THEN 1
         COUNT(CASE WHEN o.order_datetime BETWEEN DATE_SUB(CURDATE(), INTERVAL 30 DAY) AN
        NULLIF(COUNT(CASE WHEN o.order_datetime BETWEEN DATE_SUB(CURDATE(), INTERVAL 30 D
    ) as order_growth_15_days,

    -- Performance ranking within locality
    RANK() OVER (PARTITION BY r.locality ORDER BY SUM(o.order_value) DESC) as locality_re

    -- Status assessment
    CASE
        WHEN COUNT(DISTINCT o.order_id) = 0 THEN 'Inactive'
        WHEN COUNT(DISTINCT o.order_id) < 10 THEN 'Low Activity'
        WHEN COUNT(DISTINCT o.order_id) < 50 THEN 'Moderate Activity'
        ELSE 'High Activity'
    END as activity_level,

    CASE
        WHEN AVG(o.order_value) >= 800 THEN 'Premium'
        WHEN AVG(o.order_value) >= 400 THEN 'Mid-Range'
        ELSE 'Budget'
    END as actual_price_segment

FROM restaurants r
LEFT JOIN orders o ON r.restaurant_id = o.restaurant_id
    AND o.order_status = 'Delivered'
    AND o.order_datetime >= DATE_SUB(CURDATE(), INTERVAL 30 DAY)
LEFT JOIN menu_items mi ON r.restaurant_id = mi.restaurant_id
    AND mi.availability_status = TRUE
GROUP BY r.restaurant_id, r.restaurant_name, r.locality, r.restaurant_category,
         r.cuisine_type, r.average_rating
ORDER BY revenue_30_days DESC;

-- 4. Customer Insights Dashboard
CREATE VIEW customer_insights_dashboard AS
SELECT
    -- Customer segmentation
    CASE
        WHEN COUNT(o.order_id) >= 20 AND AVG(o.order_value) >= 600 THEN 'VIP Customer'
        WHEN COUNT(o.order_id) >= 10 AND AVG(o.order_value) >= 400 THEN 'Loyal Customer'
        WHEN COUNT(o.order_id) >= 5 THEN 'Regular Customer'
```

```
        WHEN AVG(o.discount_applied) > 15 THEN 'Price Sensitive'
        ELSE 'Occasional Customer'
    END as customer_segment,

    COUNT(DISTINCT c.customer_id) as customer_count,
    ROUND(AVG(customer_orders), 1) as avg_orders_per_customer,
    ROUND(AVG(customer_avg_order_value), 2) as segment_avg_order_value,
    ROUND(AVG(customer_total_spent), 2) as avg_customer_lifetime_value,
    ROUND(AVG(customer_avg_discount), 2) as avg_discount_usage,
    COUNT(DISTINCT customer_localities) as localities_served,

    -- Behavioral patterns
    ROUND(AVG(days_since_last_order), 1) as avg_days_since_last_order,
    ROUND(AVG(customer_restaurant_variety), 1) as avg_restaurants_tried,

    -- Payment preferences
    MAX(preferred_payment_method) as most_common_payment_method,

    -- Segment growth
    ROUND(
        COUNT(CASE WHEN first_order_date >= DATE_SUB(CURDATE(), INTERVAL 30 DAY) THEN 1 E
        COUNT(*), 2
    ) as new_customers_percent_last_30_days

FROM (
    SELECT
        c.customer_id,
        COUNT(o.order_id) as customer_orders,
        AVG(o.order_value) as customer_avg_order_value,
        SUM(o.order_value) as customer_total_spent,
        AVG(o.discount_applied) as customer_avg_discount,
        COUNT(DISTINCT r.locality) as customer_localities,
        DATEDIFF(CURDATE(), MAX(o.order_datetime)) as days_since_last_order,
        COUNT(DISTINCT o.restaurant_id) as customer_restaurant_variety,
        MIN(DATE(o.order_datetime)) as first_order_date,
        (SELECT payment_method FROM orders WHERE customer_id = c.customer_id
         GROUP BY payment_method ORDER BY COUNT(*) DESC LIMIT 1) as preferred_payment_met
    FROM customers c
    JOIN orders o ON c.customer_id = o.customer_id
    JOIN restaurants r ON o.restaurant_id = r.restaurant_id
    WHERE o.order_status = 'Delivered'
    AND o.order_datetime >= DATE_SUB(CURDATE(), INTERVAL 180 DAY)
    GROUP BY c.customer_id
) customer_metrics
GROUP BY customer_segment
ORDER BY avg_customer_lifetime_value DESC;

-- 5. Revenue Analytics Dashboard
CREATE VIEW revenue_analytics_dashboard AS
SELECT
    -- Time period
    DATE(o.order_datetime) as order_date,
    DAYNAME(o.order_datetime) as day_name,
    WEEK(o.order_datetime) as week_number,
    MONTH(o.order_datetime) as month_number,
```

```
    -- Daily metrics
    COUNT(DISTINCT o.order_id) as daily_orders,
    COUNT(DISTINCT o.restaurant_id) as active_restaurants,
    COUNT(DISTINCT o.customer_id) as unique_customers,
    ROUND(SUM(o.order_value), 2) as gross_revenue,
    ROUND(SUM(o.discount_applied), 2) as total_discounts,
    ROUND(SUM(o.delivery_fee), 2) as delivery_fee_revenue,
    ROUND(SUM(o.order_value - o.discount_applied), 2) as net_revenue,
    ROUND(AVG(o.order_value), 2) as avg_order_value,

    -- Operational metrics
    ROUND(AVG(o.delivery_time_minutes), 1) as avg_delivery_time,
    ROUND(SUM(o.discount_applied) / SUM(o.order_value) * 100, 2) as discount_rate_percent

    -- Category breakdown
    ROUND(SUM(CASE WHEN r.restaurant_category = 'Budget' THEN o.order_value ELSE 0 END),
    ROUND(SUM(CASE WHEN r.restaurant_category = 'Mid-Range' THEN o.order_value ELSE 0 END
    ROUND(SUM(CASE WHEN r.restaurant_category = 'Premium' THEN o.order_value ELSE 0 END),
    ROUND(SUM(CASE WHEN r.restaurant_category = 'Fine Dining' THEN o.order_value ELSE 0 E

    -- Growth indicators
    LAG(SUM(o.order_value)) OVER (ORDER BY DATE(o.order_datetime)) as previous_day_revenu
    ROUND(
        (SUM(o.order_value) - LAG(SUM(o.order_value)) OVER (ORDER BY DATE(o.order_datetim
        NULLIF(LAG(SUM(o.order_value)) OVER (ORDER BY DATE(o.order_datetime)), 0) * 100,
    ) as daily_growth_percent,

    -- Weather impact
    MAX(o.weather_condition) as predominant_weather,
    COUNT(CASE WHEN o.special_event_flag = TRUE THEN 1 END) as special_event_orders,

    -- Performance indicators
    CASE
        WHEN SUM(o.order_value) > AVG(SUM(o.order_value)) OVER (ORDER BY DATE(o.order_dat
        THEN 'Excellent'
        WHEN SUM(o.order_value) > AVG(SUM(o.order_value)) OVER (ORDER BY DATE(o.order_dat
        THEN 'Good'
        ELSE 'Below Average'
    END as daily_performance

FROM orders o
JOIN restaurants r ON o.restaurant_id = r.restaurant_id
WHERE o.order_status = 'Delivered'
AND o.order_datetime >= DATE_SUB(CURDATE(), INTERVAL 90 DAY)
GROUP BY DATE(o.order_datetime)
ORDER BY order_date DESC;

-- Create indexes for better view performance
CREATE INDEX idx_orders_date_status ON orders(order_datetime, order_status);
CREATE INDEX idx_orders_restaurant_date ON orders(restaurant_id, order_datetime);
CREATE INDEX idx_menu_items_restaurant_available ON menu_items(restaurant_id, availabilit

SELECT 'Dashboard views created successfully!' as Status;
```

7. Advanced Analytics Queries (advanced_analytics.sql)

```
-- ======================================================
-- Advanced Analytics Queries
-- File: advanced_analytics.sql
-- Description: Complex analytical queries for deep insights and machine learning feature
-- ======================================================

USE zomato_analytics;

-- 1. Customer Lifetime Value Prediction
SELECT 'Customer Lifetime Value Analysis' as Analysis_Type;

WITH customer_cohorts AS (
    SELECT
        c.customer_id,
        c.customer_name,
        DATE_FORMAT(MIN(o.order_datetime), '%Y-%m') as cohort_month,
        COUNT(o.order_id) as total_orders,
        SUM(o.order_value) as total_spent,
        AVG(o.order_value) as avg_order_value,
        DATEDIFF(MAX(o.order_datetime), MIN(o.order_datetime)) + 1 as customer_lifespan_d
        COUNT(DISTINCT DATE(o.order_datetime)) as active_days,
        COUNT(DISTINCT o.restaurant_id) as restaurants_tried,
        AVG(o.discount_applied) as avg_discount_used,
        -- Recency, Frequency, Monetary calculations
        DATEDIFF(CURDATE(), MAX(o.order_datetime)) as recency_days,
        COUNT(o.order_id) as frequency,
        SUM(o.order_value) as monetary_value
    FROM customers c
    JOIN orders o ON c.customer_id = o.customer_id
    WHERE o.order_status = 'Delivered'
    GROUP BY c.customer_id, c.customer_name
),
rfm_scores AS (
    SELECT
        *,
        -- RFM Scoring (1-5 scale)
        CASE
            WHEN recency_days <= 7 THEN 5
            WHEN recency_days <= 30 THEN 4
            WHEN recency_days <= 90 THEN 3
            WHEN recency_days <= 180 THEN 2
            ELSE 1
        END as recency_score,
        CASE
            WHEN frequency >= 20 THEN 5
            WHEN frequency >= 10 THEN 4
            WHEN frequency >= 5 THEN 3
            WHEN frequency >= 2 THEN 2
            ELSE 1
        END as frequency_score,
        CASE
            WHEN monetary_value >= 5000 THEN 5
            WHEN monetary_value >= 2000 THEN 4
            WHEN monetary_value >= 1000 THEN 3
```

```
            WHEN monetary_value >= 500 THEN 2
            ELSE 1
        END as monetary_score
    FROM customer_cohorts
)
SELECT
    cohort_month,
    COUNT(*) as customers_acquired,
    ROUND(AVG(total_spent), 2) as avg_clv,
    ROUND(AVG(avg_order_value), 2) as avg_order_value,
    ROUND(AVG(frequency), 1) as avg_order_frequency,
    ROUND(AVG(customer_lifespan_days), 1) as avg_lifespan_days,
    -- Customer segments based on RFM
    COUNT(CASE WHEN recency_score >= 4 AND frequency_score >= 4 AND monetary_score >= 4 T
    COUNT(CASE WHEN recency_score >= 3 AND frequency_score >= 3 AND monetary_score >= 3 T
    COUNT(CASE WHEN recency_score >= 3 AND frequency_score <= 2 THEN 1 END) as potential_
    COUNT(CASE WHEN recency_score <= 2 AND frequency_score >= 3 THEN 1 END) as at_risk,
    COUNT(CASE WHEN recency_score <= 2 AND frequency_score <= 2 AND monetary_score <= 2 T
    -- Predicted CLV (simplified model)
    ROUND(AVG(avg_order_value * frequency * (365.0 / customer_lifespan_days) * 2), 2) as
FROM rfm_scores
GROUP BY cohort_month
ORDER BY cohort_month DESC;

-- 2. Market Basket Analysis
SELECT 'Market Basket Analysis' as Analysis_Type;

WITH item_pairs AS (
    SELECT
        oi1.item_id as item_a,
        oi2.item_id as item_b,
        mi1.item_name as item_a_name,
        mi2.item_name as item_b_name,
        mi1.item_category as category_a,
        mi2.item_category as category_b,
        COUNT(*) as co_occurrence_count
    FROM order_items oi1
    JOIN order_items oi2 ON oi1.order_id = oi2.order_id AND oi1.item_id < oi2.item_id
    JOIN menu_items mi1 ON oi1.item_id = mi1.item_id
    JOIN menu_items mi2 ON oi2.item_id = mi2.item_id
    JOIN orders o ON oi1.order_id = o.order_id
    WHERE o.order_status = 'Delivered'
    AND o.order_datetime >= DATE_SUB(CURDATE(), INTERVAL 60 DAY)
    GROUP BY oi1.item_id, oi2.item_id, mi1.item_name, mi2.item_name, mi1.item_category, m
    HAVING co_occurrence_count >= 5
),
item_popularity AS (
    SELECT
        mi.item_id,
        mi.item_name,
        COUNT(oi.order_item_id) as individual_order_count
    FROM menu_items mi
    JOIN order_items oi ON mi.item_id = oi.item_id
    JOIN orders o ON oi.order_id = o.order_id
    WHERE o.order_status = 'Delivered'
    AND o.order_datetime >= DATE_SUB(CURDATE(), INTERVAL 60 DAY)
```

```
    GROUP BY mi.item_id, mi.item_name
)
SELECT
    ip.item_a_name,
    ip.item_b_name,
    ip.category_a,
    ip.category_b,
    ip.co_occurrence_count,
    ipa.individual_order_count as item_a_popularity,
    ipb.individual_order_count as item_b_popularity,
    -- Confidence: P(B|A) = co_occurrence / A_count
    ROUND(ip.co_occurrence_count * 100.0 / ipa.individual_order_count, 2) as confidence_a
    ROUND(ip.co_occurrence_count * 100.0 / ipb.individual_order_count, 2) as confidence_b
    -- Lift: confidence / P(B)
    ROUND((ip.co_occurrence_count * 100.0 / ipa.individual_order_count) /
          (ipb.individual_order_count * 100.0 / (SELECT SUM(individual_order_count) FROM
    CASE
        WHEN (ip.co_occurrence_count * 100.0 / ipa.individual_order_count) /
             (ipb.individual_order_count * 100.0 / (SELECT SUM(individual_order_count) FR
        THEN 'Strong Association'
        WHEN (ip.co_occurrence_count * 100.0 / ipa.individual_order_count) /
             (ipb.individual_order_count * 100.0 / (SELECT SUM(individual_order_count) FR
        THEN 'Moderate Association'
        ELSE 'Weak Association'
    END as association_strength
FROM item_pairs ip
JOIN item_popularity ipa ON ip.item_a = ipa.item_id
JOIN item_popularity ipb ON ip.item_b = ipb.item_id
ORDER BY lift DESC
LIMIT 20;

-- 3. Demand Forecasting Features
SELECT 'Demand Forecasting Data Preparation' as Analysis_Type;

WITH daily_demand AS (
    SELECT
        DATE(o.order_datetime) as order_date,
        DAYOFWEEK(o.order_datetime) as day_of_week,
        DAYOFYEAR(o.order_datetime) as day_of_year,
        WEEK(o.order_datetime) as week_of_year,
        MONTH(o.order_datetime) as month_number,
        r.restaurant_category,
        mi.item_category,
        COUNT(oi.order_item_id) as daily_demand,
        SUM(oi.quantity) as total_quantity,
        AVG(oi.unit_price) as avg_price,
        MAX(o.weather_condition) as weather,
        COUNT(CASE WHEN o.special_event_flag = TRUE THEN 1 END) as special_events_count,
        AVG(o.delivery_time_minutes) as avg_delivery_time
    FROM orders o
    JOIN order_items oi ON o.order_id = oi.order_id
    JOIN menu_items mi ON oi.item_id = mi.item_id
    JOIN restaurants r ON o.restaurant_id = r.restaurant_id
    WHERE o.order_status = 'Delivered'
    AND o.order_datetime >= DATE_SUB(CURDATE(), INTERVAL 90 DAY)
    GROUP BY DATE(o.order_datetime), DAYOFWEEK(o.order_datetime),
```

```
            DAYOFYEAR(o.order_datetime), WEEK(o.order_datetime),
            MONTH(o.order_datetime), r.restaurant_category, mi.item_category
),
demand_with_lags AS (
    SELECT
        *,
        LAG(daily_demand, 1) OVER (PARTITION BY restaurant_category, item_category ORDER
        LAG(daily_demand, 7) OVER (PARTITION BY restaurant_category, item_category ORDER
        AVG(daily_demand) OVER (PARTITION BY restaurant_category, item_category ORDER BY
        AVG(daily_demand) OVER (PARTITION BY restaurant_category, item_category ORDER BY
    FROM daily_demand
)
SELECT
    restaurant_category,
    item_category,
    COUNT(*) as data_points,
    ROUND(AVG(daily_demand), 2) as avg_daily_demand,
    ROUND(STDDEV(daily_demand), 2) as demand_volatility,
    ROUND(MIN(daily_demand), 2) as min_demand,
    ROUND(MAX(daily_demand), 2) as max_demand,
    -- Seasonality indicators
    ROUND(AVG(CASE WHEN day_of_week IN (1,7) THEN daily_demand END), 2) as weekend_avg_de
    ROUND(AVG(CASE WHEN day_of_week BETWEEN 2 AND 6 THEN daily_demand END), 2) as weekday
    -- Weather correlation (simplified)
    ROUND(AVG(CASE WHEN weather = 'Clear' THEN daily_demand END), 2) as clear_weather_dem
    ROUND(AVG(CASE WHEN weather LIKE '%Rain%' THEN daily_demand END), 2) as rainy_weather
    -- Trend analysis
    CASE
        WHEN AVG(CASE WHEN order_date >= DATE_SUB(CURDATE(), INTERVAL 30 DAY) THEN daily_
             AVG(CASE WHEN order_date BETWEEN DATE_SUB(CURDATE(), INTERVAL 60 DAY) AND DA
        THEN 'Increasing'
        ELSE 'Stable/Decreasing'
    END as demand_trend
FROM demand_with_lags
WHERE demand_lag_1 IS NOT NULL
GROUP BY restaurant_category, item_category
ORDER BY avg_daily_demand DESC;

-- 4. Price Optimization Recommendations
SELECT 'Price Optimization Recommendations' as Analysis_Type;

WITH price_performance AS (
    SELECT
        mi.item_id,
        mi.item_name,
        mi.item_category,
        r.restaurant_category,
        r.locality,
        mi.base_price,
        mi.current_price,
        mi.cost_of_goods_sold,
        COUNT(oi.order_item_id) as orders_count,
        SUM(oi.quantity) as total_quantity_sold,
        SUM(oi.total_price) as total_revenue,
        ROUND(AVG(oi.unit_price), 2) as avg_realized_price,
        -- Profitability metrics
```

```
        ROUND((mi.current_price - mi.cost_of_goods_sold) / mi.current_price * 100, 2) as
        ROUND((SUM(oi.total_price) - SUM(oi.quantity * mi.cost_of_goods_sold)), 2) as tot
        -- Market position
        RANK() OVER (PARTITION BY mi.item_category, r.locality ORDER BY mi.current_price
        COUNT(*) OVER (PARTITION BY mi.item_category, r.locality) as competitors_in_categ
    FROM menu_items mi
    JOIN restaurants r ON mi.restaurant_id = r.restaurant_id
    LEFT JOIN order_items oi ON mi.item_id = oi.item_id
    LEFT JOIN orders o ON oi.order_id = o.order_id
        AND o.order_status = 'Delivered'
        AND o.order_datetime >= DATE_SUB(CURDATE(), INTERVAL 60 DAY)
    WHERE mi.availability_status = TRUE
    AND mi.cost_of_goods_sold > 0
    GROUP BY mi.item_id, mi.item_name, mi.item_category, r.restaurant_category,
             r.locality, mi.base_price, mi.current_price, mi.cost_of_goods_sold
),
elasticity_estimates AS (
    SELECT
        pp.*,
        COALESCE(ec.price_elasticity, -1.2) as estimated_elasticity,
        ec.confidence_level
    FROM price_performance pp
    LEFT JOIN elasticity_coefficients ec ON pp.item_category = ec.item_category
        AND pp.restaurant_category = ec.restaurant_category
)
SELECT
    item_name,
    item_category,
    restaurant_category,
    locality,
    current_price,
    profit_margin_percent,
    orders_count,
    total_revenue,
    estimated_elasticity,
    CASE
        WHEN price_rank_in_category <= competitors_in_category * 0.3 THEN 'Premium Priced
        WHEN price_rank_in_category <= competitors_in_category * 0.7 THEN 'Market Priced'
        ELSE 'Value Priced'
    END as market_position,
    -- Price optimization recommendation
    CASE
        WHEN profit_margin_percent < 20 AND orders_count < 10 THEN 'Increase Price Signi
        WHEN profit_margin_percent < 30 AND estimated_elasticity > -1.0 AND orders_count
        WHEN profit_margin_percent > 50 AND orders_count < 5 THEN 'Consider Price Reducti
        WHEN orders_count >= 20 AND profit_margin_percent >= 30 THEN 'Test Premium Pricin
        ELSE 'Maintain Current Price'
    END as pricing_recommendation,
    -- Suggested price range
    ROUND(current_price * 0.95, 2) as suggested_min_price,
    ROUND(current_price * 1.15, 2) as suggested_max_price,
    -- Revenue impact estimation
    ROUND(
        CASE
            WHEN estimated_elasticity < -1.5 THEN total_revenue * 0.95  -- Elastic items,
            WHEN estimated_elasticity < -1.0 THEN total_revenue * 1.05  -- Moderate elast
```

```
            ELSE total_revenue * 1.10  -- Inelastic items, can increase more
        END, 2
    ) as estimated_revenue_after_optimization
FROM elasticity_estimates
WHERE orders_count > 0  -- Only items that have been sold
ORDER BY total_revenue DESC
LIMIT 50;

-- 5. Competitive Analysis
SELECT 'Competitive Analysis by Locality' as Analysis_Type;

WITH locality_competition AS (
    SELECT
        r.locality,
        mi.item_category,
        COUNT(DISTINCT r.restaurant_id) as competing_restaurants,
        COUNT(DISTINCT mi.item_id) as total_menu_items,
        ROUND(AVG(mi.current_price), 2) as avg_market_price,
        ROUND(MIN(mi.current_price), 2) as min_market_price,
        ROUND(MAX(mi.current_price), 2) as max_market_price,
        ROUND(PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY mi.current_price), 2) as price
        ROUND(PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY mi.current_price), 2) as price
        ROUND(STDDEV(mi.current_price), 2) as price_volatility
    FROM restaurants r
    JOIN menu_items mi ON r.restaurant_id = mi.restaurant_id
    WHERE mi.availability_status = TRUE
    GROUP BY r.locality, mi.item_category
    HAVING competing_restaurants >= 3
),
restaurant_positioning AS (
    SELECT
        r.restaurant_name,
        r.locality,
        r.restaurant_category,
        mi.item_category,
        ROUND(AVG(mi.current_price), 2) as restaurant_avg_price,
        COUNT(mi.item_id) as category_items_count,
        ROUND(AVG(r.average_rating), 2) as restaurant_rating
    FROM restaurants r
    JOIN menu_items mi ON r.restaurant_id = mi.restaurant_id
    WHERE mi.availability_status = TRUE
    GROUP BY r.restaurant_name, r.locality, r.restaurant_category, mi.item_category
)
SELECT
    lc.locality,
    lc.item_category,
    lc.competing_restaurants,
    lc.avg_market_price,
    lc.min_market_price,
    lc.max_market_price,
    lc.price_volatility,
    -- Market concentration
    CASE
        WHEN lc.price_volatility / lc.avg_market_price < 0.15 THEN 'Low Price Competition
        WHEN lc.price_volatility / lc.avg_market_price < 0.30 THEN 'Moderate Price Compet
        ELSE 'High Price Competition'
```

```
    END as competition_intensity,
    -- Top performers in category
    (SELECT GROUP_CONCAT(DISTINCT rp.restaurant_name ORDER BY rp.restaurant_rating DESC L
     FROM restaurant_positioning rp
     WHERE rp.locality = lc.locality AND rp.item_category = lc.item_category
     AND rp.restaurant_avg_price >= lc.price_75th_percentile) as premium_players,
    -- Value players
    (SELECT GROUP_CONCAT(DISTINCT rp.restaurant_name ORDER BY rp.restaurant_rating DESC L
     FROM restaurant_positioning rp
     WHERE rp.locality = lc.locality AND rp.item_category = lc.item_category
     AND rp.restaurant_avg_price <= lc.price_25th_percentile) as value_players,
    -- Market opportunity
    CASE
        WHEN lc.max_market_price / lc.min_market_price > 2 THEN 'High Price Spread - Prem
        WHEN lc.competing_restaurants < 5 THEN 'Limited Competition - Growth Opportunity'
        ELSE 'Saturated Market - Focus on Differentiation'
    END as market_opportunity
FROM locality_competition lc
ORDER BY lc.locality, lc.avg_market_price DESC;

SELECT 'Advanced analytics queries completed successfully!' as Status;
```

Usage Instructions

To use these files:

1. **Start with** `schema_creation.sql` - Creates the complete database structure
2. **Run** `sample_data.sql` - Populates tables with realistic test data
3. **Execute** `basic_analysis.sql` - Performs fundamental analysis
4. **Run** `price_elasticity.sql` - Calculates price sensitivity metrics
5. **Execute** `dynamic_pricing.sql` - Creates pricing procedures and functions
6. **Run** `dashboard_views.sql` - Sets up monitoring dashboards
7. **Execute** `advanced_analytics.sql` - Performs complex analytical queries

Each file is self-contained and includes:

- Proper error handling
- Detailed comments
- Performance optimizations
- Realistic sample data
- Industry-standard calculations

You can download each file separately and execute them in your MySQL environment to build a complete restaurant analytics and dynamic pricing system.