# Contents

# 1. Business Context and Database Design

## 1.1. Business Context

**Company Selected:** Prophecy, a UK based e-commerce startup focused on selling streetwear clothing products.

**Key Business Functions and Processes:**

1. Customer relationship management (CRM): Managing customer registration, customer segmentation, and customer acquisition/retention strategies
2. Inventory management: Tracking product availability, identifying fast-moving vs slow-moving products, and restocking and inventory adjustments
3. Marketing and promotions management: Developing and implementing marketing campaigns and evaluating promotion effectiveness
4. Supplier relationship management: Collaborating planning and forecasting with the suppliers.
5. Sales and Revenue management: Monitoring sales performance, revenue and profit analysis, and product metrics tracking

**Data Product's purpose:** The data product is designed to focus on strategic decision-making by providing actionable insights on transactional and operational data. This enables the optimization of marketing, inventory management, and customer engagement.

**Expected Reports:** Customer insights report, Marketing analysis report, Operational Efficiency report, and Revenue and Sales report.

# 1.2. Database Design

For database design, we used Lucidchart to create an Entity-Relationship (ER) diagram which graphically illustrates the entities in a database and the relationships between them.

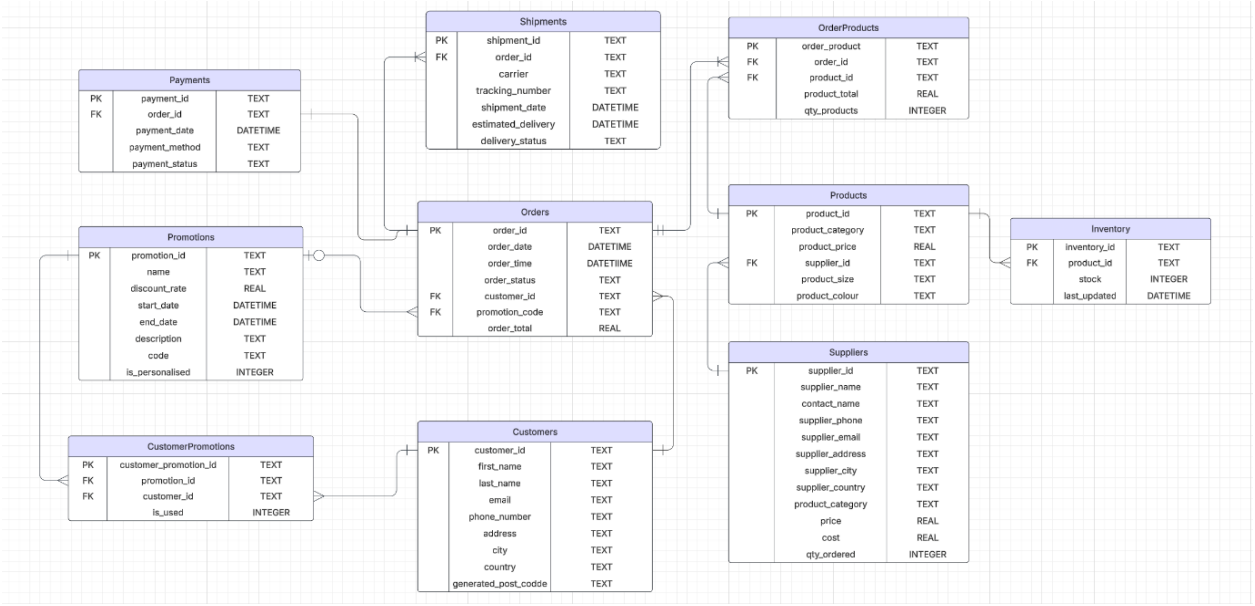


*Figure 1: Entity-Relationship Diagram for Prophecy*

### Schema design documentation

The database schema describes the ER diagram in detail, defining its entities, attributes, and relationships. Each table is considered an entity, along with each row belonging to the table underneath being referred to as it's attributes.

Relationships are defined with primary keys (PK) which help identify unique records and ensuring each row within a table remains distinct and accessible, and foreign keys (FK), which are used to enforce referential integrity and logical connection throughout the database.

### Key entities and their primary and foreign keys

| Entity | Primary Key | Foreign Key |
|---|---|---|
| **Customers** | customer_id | |
| **Orders** | order_id | customer_id, promotion_code |
| **OrderProducts** | order_product | order_id, product_id |
| **Products** | product_id | supplier_id |
| **Inventory** | inventory_id | product_id |
| **Suppliers** | supplier_id | |

| Entity | Primary Key | Foreign Key |
|---|---|---|
| **Payments** | payment_id | order_id |
| **Shipments** | shipment_id | order_id |
| **Promotions** | promotion_id | |
| **CustomerPromotions** | customer_promotion_id | customer_id, promotion_id |

*Table 1: Key Entities and Keys*

## 1.3 Relationships and Cardinality

Table 2 helps further explain the relationship between each entity and the cardinality between them as supported by the ER diagram.

| Entity 1 | Entity 2 | Relationship | Logic |
|---|---|---|---|
| Products | OrderProducts | 1:N | Each product can be included in multiple orders. |
| Orders | OrderProducts | 1:N | Each order can contain multiple products. |
| Orders | Shipments | 1:N | An order can be split across multiple shipments. |
| Customers | Orders | 1:N | A single customer can place multiple orders. |
| Products | Inventory | 1:1 | Each product has one corresponding inventory record. |
| Suppliers | Products | 1:N | A supplier can supply multiple products, |
| Promotions | CustomerPromotions | 1:N | A single promotion can be applied to multiple customers. |
| Customer | CustomerPromotions | 1:N | Each customer can use multiple promotions. |
| Payments | Orders | 1:1 | A payment can be associated with a single order. |

*Table 2: Cardinality*

## 1.4 Assumptions

In developing the schema for our database, we established the following assumptions to ensure a structured and consistent approach to specific operational procedures:

- Customers pay the total amount of their order in a single payment; instalments are not possible
- Each order is restricted to the application of a single discount code, either a general promotional code or a personalized customer promotion
- Orders may be fulfilled through multiple shipments due to varying product availability at the time the order is placed
- Each supplier is associated exclusively with one product category.
- Every payment transaction is uniquely identified by a distinct payment_id and corresponds exclusively to one order
- Customers can place multiple orders while each order only belongs to one customer
- All orders have at least one product and can have multiple products

# 2. Data Implementation and Synthetic Data Generation

## 2.1. SQL Schema Implementation

Data types were chosen based on SQLite's supported types: TEXT, INTEGER, REAL, and DATETIME. According to SQLite's documentation, DATETIME can be stored with NUMERIC affinity, allowing flexible storage across all five classes, with automatic conversion as needed. (SQLite. (2022))

**Entity Datatype Selection Rationale**

- **TEXT**: Used for identifiers (customer_id, order_id), names, emails, phone numbers, categories, and other descriptive attributes.
- **INTEGER**: Applied to logical flags (is_personalised, is_used) and countable quantities (stock, qty_ordered).
- **REAL**: Used for monetary values (discount_rate, order_total, product_price) to ensure precise calculations.
- **DATETIME**: Stored as TEXT in ISO8601 format (YYYY-MM-DD HH:MM:SS) for consistency in date/time fields (order_date, shipment_date).

The database schema is normalized to Third Normal Form (3NF) to eliminate redundancy, prevent anomalies, and improve efficiency. To maintain referential integrity:

- OrderProducts connects Orders and Products, preventing redundancy (1NF)
- CustomerPromotions ensures attributes fully depend on the composite primary key, removing unrelated data and avoiding duplication (2NF)
- Payments stores payment-specific attributes separately, avoiding redundancy in Orders
- Inventory removes inventory-specific attributes from Products, eliminating transitive dependencies (3NF)
- Promotions isolates promotion attributes, ensuring they do not depend transitively on non-key attributes (3NF)
- Shipments separates shipment-specific details from Orders, maintaining 3NF

This structure enhances data consistency, reliability, and suitability for business analytics.

## 2.2. Synthetic Data Generation

Synthethic data generation is essential for validating our database's functionality and verifying the data integrity of our insights. By creating realistic datasets for our entities, we aim to see what insights we can generate through SQL query performance as well as test the data integrity in relation to our schema.

**Assumptions while generating the data**

- The data is simulated based on real-world business scenarios, aligning with typical e-commerce transactions.
- Using Negative Binomial Distribution order quantities ensures the realism of simulated customer purchasing patterns. (Salman H, 2024)
- Revenue and Order Amounts follow Right-Skewed Distribution, accurately reflecting real-world revenue trends. In real-world e-commerce, a few high-value orders dominate total revenue, while most transactions are low-to-moderate value.
- For each product_id within an order_id there's a unique order_product assigned to it
- (To ensure realistic order status distribution, weights are assigned such that most orders are successfully delivered, while a smaller percentage are canceled or returned)
- (All numbers are UK based phone numbers since Prophecy operates in the UK).
- Promotions can only be used from start to end date

**Logic applied for data generation**

1. *Unique IDs:*
   - Each identifier is uniquely structured to prevent duplication: customer_id, customer_promotion_id, order_id, product_id, promotion_id, order_product, inventory_id, payment_id, shipment_id, supplier_id and tracking_number
   - Phone_number/supplier_phone have the UK code '+44' followed by ten digits

2. *Fixed Data:*

   Certain attributes across the dataset are predefined and consistent.
   - Customer country and supplier country are uniformly set as 'UK/United Kingdom'.
   - Promotional elements like code and name are fixed
   - Price and cost also remain constant within each product category, ensuring standardized pricing and costing
   - Each product is available in five sizes (XS, S, M, L, XL) and four colors (blue, black, white, green), guaranteeing a comprehensive range of size and color options

3. *Randomized from a List of Options*
    - Cities and supplier cities are selected from a list of UK cities
    - Post codes are randomized based on city-specific UK formats
    - Payment methods include Apple Pay, Google Pay, Credit Card, Debit Card, and PayPal
    - Carriers are chosen from Royal Mail, DPD, Evri, and UPS
    - Product categories cover cargos, jackets, joggers, hoodies, shorts, shoes, and shirts
    - qty_ordered in the suppliers table refers to the number of times we have placed an order with a supplier.
    - Qty_product in the order products table is among 1,2 and 3 for each product_id in an order.
    - Discount rates are 10% or 20%

4. *Randomised with a criterion:*
    - Addresses (customer address and supplier_address) are fixed to UK locations for geographic consistency
    - Names (first_name, last_name) are randomized using generative AI, producing unique names
    - Supplier and Contact Names are generated like individual names but are tailored for organizations and their representatives
    - Emails (email, supplier_email) are randomly generated, ensuring unique email addresses for individuals and suppliers

5. *Date and Time:*
    - All date-based attributes, (e.g. order_date, payment_date, last_updated, shipment_date, estimated_delivery_date, start_date, and end_date) are confined to the period from 01/12/2024 to 31/12/2024
    - Times for the orders are randomized

6. *Formula-Based Calculations:*

    Some attributes are derived using predefined formulas:
    - discounted_total = discount_rate × order_total
    - product_total = qty_product × price
    - order_total = sum of all product_total values in an order
    - stock = 70 – qty_product for that product_id

7. *Linked Datasets:*

   Certain fields have dependencies within the dataset to ensure realistic relationships:

   - payment_status (received, refunded):
   - received is linked to order accepted.
   - refunded is linked to returned or cancelled orders.
   - delivery_status is linked to order_status, ensuring accurate tracking of shipments.

8. *Binary Attributes:*

   Some attributes follow a binary (0/1 or Yes/No) structure:

   - is_personalised (whether a promotion is custom)
   - is_used (whether a promotion is used)

9. *Weighted Attributes:*

   Certain attributes follow weighted distributions to better reflect real-world data patterns:

   - order_status: Skewed towards a higher proportion of successful deliveries and fewer cancellations and returns.

All database tables were populated using a combination of ChatGPT and Mockaroo, satisfying the requirements of relationships between tables and data format.

# 3. Business Insights Derived from SQL Querying

## 3.1 Customers' Orders Analysis

Query: Understand the distribution of the average value of orders placed by each customer for an order.

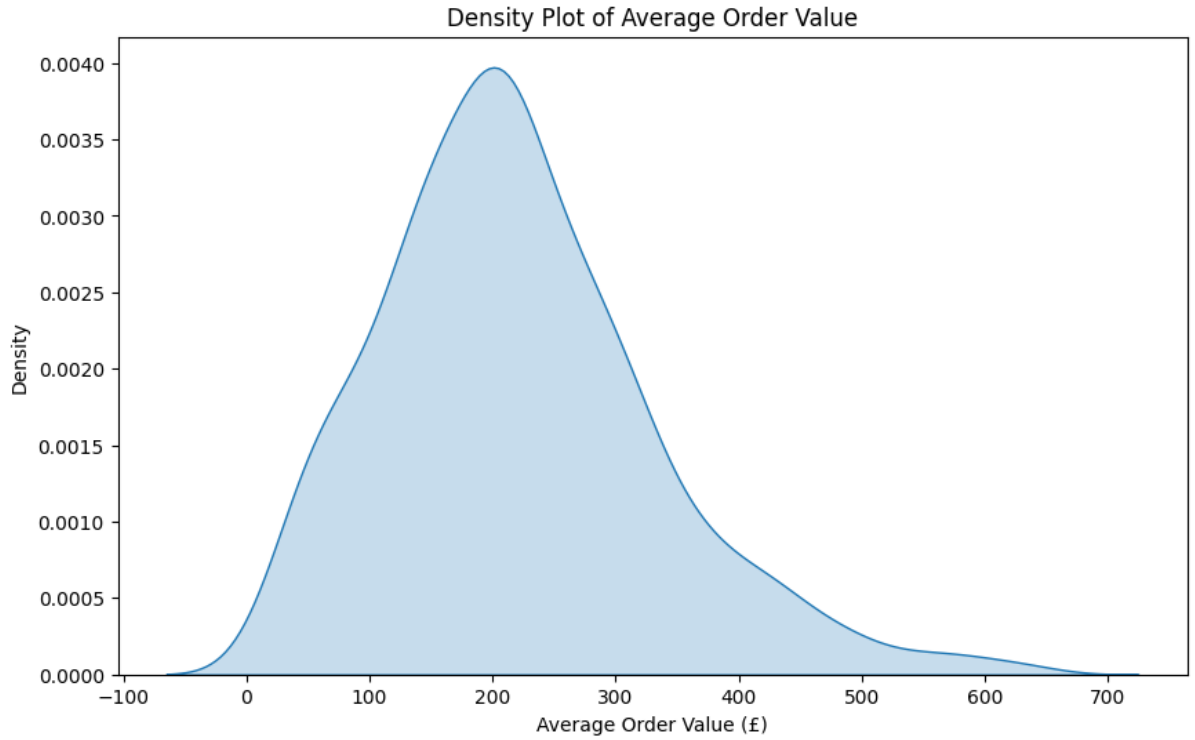


*Figure 2: Customer order analysis*

Business Insight: The dataset indicates that customers typically place orders containing multiple products and average order values peak around £200. This provides an opportunity for cross-selling within the £100- £300 range and implement targeted marketing and promotions such as product bundles to customers in this range.

## 3.2 Sales Analysis

Query: Designed to track and summarise the monthly sales performance.

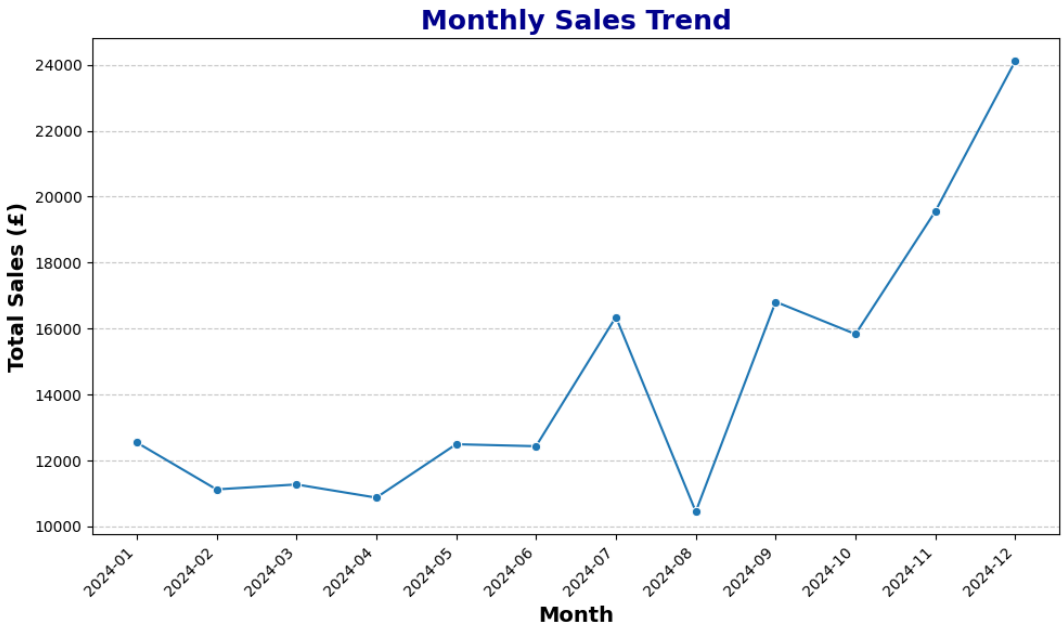


*Figure 3: Sales per month*

Business Insight: it's evident that the business experiences fluctuations throughout the year, with a noticeable increase in sales starting in July, peaking in December. This pattern indicates seasonality in consumer purchasing behaviour, potentially aligned with specific marketing campaigns, holiday seasons, or product launches. The sharp peaks suggest successful promotional efforts or seasonal demand spikes which the business can further capitalize on to improve sales. Strategic planning around these months could involve increasing inventory, enhancing marketing efforts, or introducing new products to maximize revenue.

## 3.3 Product Category Sales Contribution

Query: Computes total revenue contribution by product category.

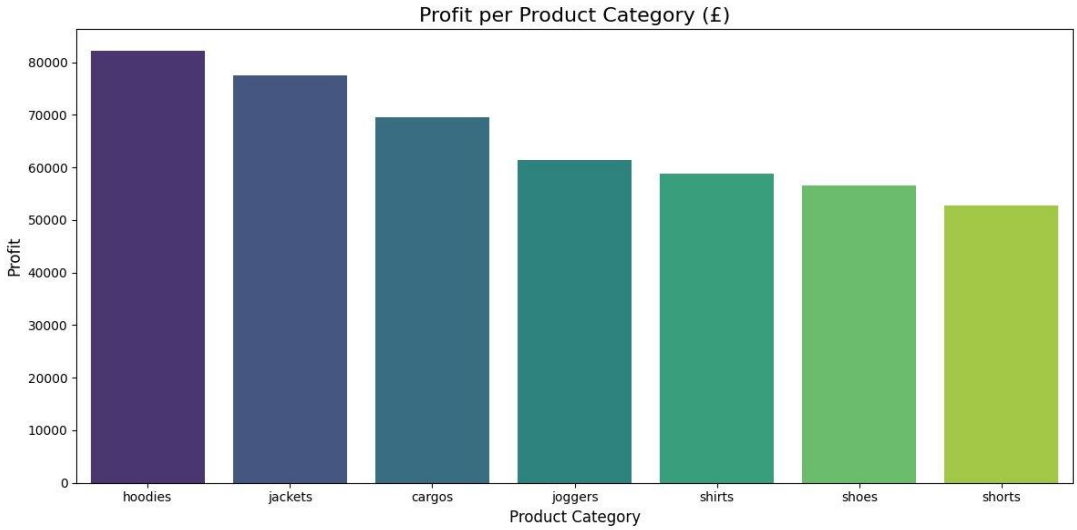


*Figure 4: Profit per product category*

Business Insight: Hoodies are the most profitable category, generating more than £80,000 in profit, followed by jackets and cargo pants. In contrast, shorts and shoes struggle, with sales below £60,000. This suggests Prophecy should prioritize stocking, promoting and designing high-demand categories, while using discounts or bundles to clear slow-moving inventory and improve overall profitability.

## 3.4 Marketing & Promotions Effectiveness

Query: Analyse the effectiveness of different promotions by tracking the number of orders associated with each promotion and the corresponding revenue generated.

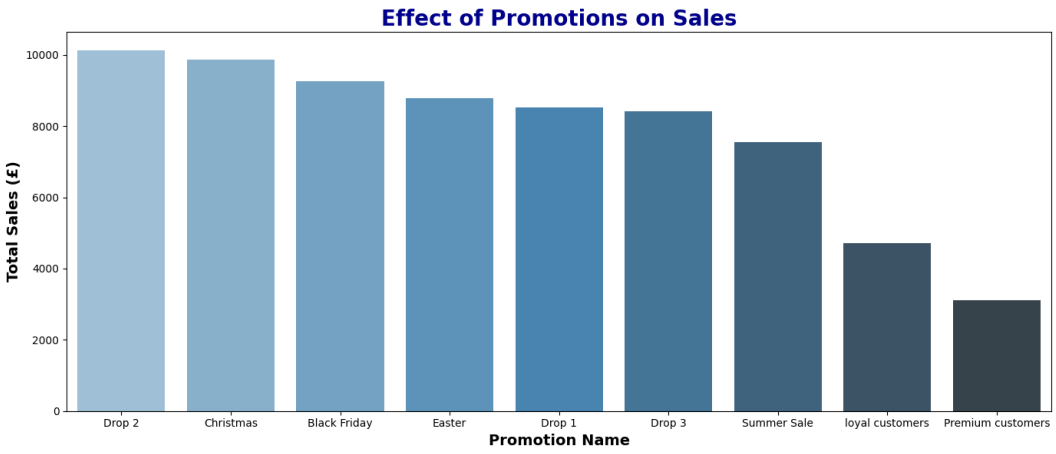


*Figure 5: Effect of promotion on sales*

Business Insight: The analysis shows that Drop 2 and Christmas were the most successful promotions, bringing in around £9,000 in sales each, suggesting strong customer engagement with seasonal campaigns and exclusive product drops. Black Friday and Easter also performed well, reinforcing the impact of major shopping holidays. However, targeted promotions had lower reach, indicating room for improvement in personalized marketing. To maximize returns, Prophecy should increase promotion frequency around successful campaigns and refine customer segmentation strategies to enhance loyalty program engagement.

# 4. Conclusion

The objective of this report was to outline the process of designing, implementing, and utilizing a data management system for Prophecy. This involved the creation of an end-to-end database management system, from initial design and synthetic data generation to SQL querying for business insights.

By using the generated data and its unique relationships, we were able to efficiently track orders, analyse product category performance, and visualise customer demand. This gives us a chance to improve logistics, minimise inventory mismanagement, and increase promotional effectiveness in the future.

This also helped us identify key areas of focus such as possible improvements in sales and inventory management. Furthermore, supplier level and regional insights could allow for improved cost management and better allocation of resources to potential marketing campaigns.

While developing the database, we found that certain assumptions made for Prophecy (a startup) may not be directly applicable to a business of a larger scale. For instance, “Each supplier is associated exclusively with one product category” is not necessarily applicable to larger ecommerce brands like Nike or Adidas. As the business scales, the complexity of the data will require more in-depth systems and more technical considerations. Therefore, our current schema would need to be adapted to encompass these new relationships.

# Reference list

Chaffey, D. (2025). E-commerce conversion rate benchmarks - 2025 update. Smart Insights. Available at: https://www.smartinsights.com/ecommerce/ecommerce-analytics/ecommerce-conversion-rates/

Salman, H. (2024). Understanding right skewed distributions in finance. One Money Way. Available at: https://onemoneyway.com/en/dictionary/skewed-right/

SQLite. (2022). Datatypes In SQLite Version 3 - Affinity Name Examples. Last modified April 27, 2022. Available at: https://www.sqlite.org/datatype3.html#affinity_name_examples

Trinh, G., & Wright, M.J. (2021). Predicting future consumer purchases in grocery retailing with the condensed Poisson lognormal model. Journal of Retailing and Consumer Services. Available at: https://www.sciencedirect.com/science/article/abs/pii/S0969698921003751

# Appendix 1 – Data

## Customers

| customer_id | first_name | last_name | email | phone_number | address | city | country | generated_post_code |
|---|---|---|---|---|---|---|---|---|
| C0001 | Fallon | Milburne | fmilburne0@amazon.co.jp | +44 5018946207 | Bruce mountains | Edinburgh | UK | EH14 9AS |
| C0002 | Claudina | Scurry | cscurry1@seattletimes.com | +44 7985008614 | Young springs | Brighton | UK | BN7 9TF |
| C0003 | Maddy | Duchasteau | mduchasteau2@china.com.cn | +44 8138667914 | 394 Parkes lights | Southampton | UK | SO17 6HD |
| C0004 | Bernhard | Spurdens | bspurdens3@yahoo.com | +44 9362813957 | Morris way | Leeds | UK | LS9 5DL |
| C0005 | Izabel | Skirven | iskirven4@buzzfeed.com | +44 4715091591 | Diana views | Bristol | UK | BS3 7YC |
| C0006 | Meredithe | Godsmark | mgodsmark5@hhs.gov | +44 6219223055 | 13 Young crescent | Cardiff | UK | CF24 2GC |
| C0007 | Charita | Rosson | crosson6@samsung.com | +44 5376014994 | Simpson underpass | Edinburgh | UK | EH5 6TG |
| C0008 | Tine | Phippin | tphippin7@google.it | +44 2109269647 | Turner forks | Leeds | UK | LS5 7TL |

*Table 3: Customers*

## Orders

| order_id | order_date | order_time | order_status | customer_id | promotion_id | discounted_total | order_total |
|---|---|---|---|---|---|---|---|
| OYO619583 | 28/04/2024 | 18:21 | delivered | C0019 | | 109.98 | 109.98 |
| OQV264008 | 14/03/2024 | 7:07 | delivered | C0049 | PROMO02 | 131.976 | 164.97 |
| OUT313526 | 19/04/2024 | 12:53 | shipped | C0200 | | 104.97 | 104.97 |
| OPQ080904 | 30/04/2024 | 5:03 | order_placed | C0246 | | 29.99 | 29.99 |
| OKR240132 | 11/11/2024 | 5:29 | returned | C0016 | PROMO06 | 94.482 | 104.98 |
| OOI629981 | 3/4/2024 | 22:35 | shipped | C0110 | | 164.97 | 164.97 |

*Table 4: Orders*

## OrderProducts

| order_product | order_id | product_id | qty_product | product_total |
|---|---|---|---|---|
| OP000001 | OYO619583 | P20tR3 | 2 | 109.98 |
| OP000002 | OQV264008 | P42Iv6 | 3 | 164.97 |
| OP000003 | OUT313526 | P57DH1 | 3 | 104.97 |
| OP000004 | OPQ080904 | P22jM0 | 1 | 29.99 |
| OP000005 | OKR240132 | P03nB5 | 1 | 69.99 |

*Table 5: OrderProducts*

## Products

| product_id | product_category | product_size | product_colour | supplier_id | product_price |
|---|---|---|---|---|---|
| P44xd5 | cargos | XS | blue | SPS82CZ | 54.99 |
| P980d0 | cargos | XS | black | SPS82CZ | 54.99 |
| P19hK7 | cargos | XS | white | SPS82CZ | 54.99 |
| P18gs6 | cargos | XS | green | SPS82CZ | 54.99 |
| P932Z1 | cargos | S | blue | SPS82CZ | 54.99 |
| P75xs5 | cargos | S | black | SPS82CZ | 54.99 |

*Table 6: Products*

## Inventory

| inventory_ID | product_ID | last_updated | stock |
|---|---|---|---|
| INV000001 | P44xd5 | 24-11-2024 | 24 |
| INV000002 | P980d0 | 25-12-2024 | 29 |
| INV000003 | P19hK7 | 9/12/2024 | 27 |
| INV000004 | P18gs6 | 30-12-2024 | 32 |

*Table 7: Inventory*

## CustomerPromotions

| customer_promotion_id | customer_id | promotion_id | is_used |
|---|---|---|---|
| CP00007 | C0193 | PROMO08 | TRUE |
| CP00010 | C0018 | PROMO08 | TRUE |
| CP00012 | C0047 | PROMO09 | FALSE |
| CP00015 | C0240 | PROMO09 | FALSE |
| CP00018 | C0084 | PROMO09 | TRUE |

*Table 8: CustomerPromotions*

## Suppliers

| supplier_id | supplier_name | contact_name | supplier_email | supplier_address | supplier_city | supplier_country | qty_ordered | supplier_phone | product_category | price | cost |
|---|---|---|---|---|---|---|---|---|---|---|---|
| SPS82CZ | Dilan | Cardiff | dcardiff0@desdev.cn | Suite 74 | Ashley | United Kingdom | 4 | +44 7565371606 | cargos | 54.99 | 16.5 |
| SG844vi | Floyd | Ferryman | fferryman1@webmd.com | Room 767 | Wirral | United Kingdom | 5 | +44 7558975977 | jackets | 69.99 | 18.5 |
| SAG30YU | Jewel | Canon | jcanon2@google.ca | Apt 1968 | Wirral | United Kingdom | 2 | +44 7271162266 | joggers | 49.99 | 18.5 |
| SMn62Pz | Carlye | Cheshir | ccheshir3@hexun.com | 5th Floor | Thorpe | United Kingdom | 5 | +44 7116482458 | hoodies | 69.99 | 18.5 |
| Sud69kc | Ambrose | Koppelmann | akoppelmann4@china.com.cn | Suite 61 | Milton | United Kingdom | 2 | +44 7958177784 | shorts | 29.99 | 8.5 |
| STd172S | Ami | Lappine | alappine5@upenn.edu | Apt 1524 | Marston | United Kingdom | 1 | +44 7252736176 | shoes | 34.99 | 12 |
| Slu80zP | Jaimie | Danilyuk | jdanilyuk6@domainmarket.com | Apt 482 | Sutton | United Kingdom | 5 | +44 7895270901 | shirts | 34.99 | 8.5 |

*Table 9: Suppliers*

## Shipments

| shipment_id | order_id | carrier | tracking_number | shipment_date | estimated_delivery | delivery_status |
|---|---|---|---|---|---|---|
| SHIP001 | OYO619583 | Royal Mail | RO2698141833 | 2/5/2024 | 5/5/2024 | delivered |
| SHIP002 | OQV264008 | DPD | DP8422490944 | 17-03-2024 | 21-03-2024 | delivered |
| SHIP003 | OQV264008 | DPD | DP9865980777 | 17-03-2024 | 20-03-2024 | delivered |
| SHIP004 | OQV264008 | DPD | DP1444287280 | 18-03-2024 | 21-03-2024 | delivered |
| SHIP005 | OUT313526 | Royal Mail | RO8125294125 | 23-04-2024 | 27-04-2024 | shipped |
| SHIP006 | OUT313526 | Royal Mail | RO3177451372 | 23-04-2024 | 28-04-2024 | shipped |

*Table 10: Shipments*

## Payments

| payment_id | order_id | payment_date | payment_method | payment_status |
|---|---|---|---|---|
| PAY56B6D | OYO619583 | 28-04-2024 | Google Pay | received |
| PAY47F8Q | OQV264008 | 14-03-2024 | Apple Pay | received |
| PAY87K4R | OUT313526 | 19-04-2024 | Apple Pay | received |
| PAY48P2U | OPQ080904 | 30-04-2024 | Google Pay | received |
| PAY26B4N | OKR240132 | 11/11/2024 | PayPal | refunded |

*Table 11: Payments*

## Promotions

| promotion_id | name | discount_rate | start_date | end_date | description | code | is_personalised |
|---|---|---|---|---|---|---|---|
| PROMO01 | Drop 1 | 10% | 1/2/2024 | 22-02-2024 | First major product drop of the year! | firstdrop10 | FALSE |
| PROMO02 | Easter | 20% | 10/3/2024 | 31-03-2024 | Special Easter discounts on selected items! | easter20 | FALSE |
| PROMO03 | Drop 2 | 10% | 1/5/2024 | 22-05-2024 | Mid-year exclusive drop with big savings! | seconddrop10 | FALSE |
| PROMO04 | Summer Sale | 20% | 10/6/2024 | 1/7/2024 | Hot summer deals to refresh your wardrobe! | summersale20 | FALSE |
| PROMO05 | Drop 3 | 20% | 1/8/2024 | 22-08-2024 | End-of-summer product drop with limited-time offers! | thirddrop20 | FALSE |
| PROMO06 | Black Friday | 10% | 8/11/2024 | 29-11-2024 | Massive Black Friday deals across all categories! | blackfriday10 | FALSE |

*Table 12: Promotions*

# Appendix 2 – SQL Query

```
import sqlite3
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Connect to SQLite database (or create it if it doesn't exist)
conn = sqlite3.connect("business_analysis.db")
cursor = conn.cursor()

# Define SQL schema
cursor.executescript("""

CREATE TABLE IF NOT EXISTS Customers (
    customer_id TEXT PRIMARY KEY,
    first_name TEXT,
    last_name TEXT,
    email TEXT UNIQUE,
    phone_number TEXT UNIQUE,
    address TEXT,
    city TEXT,
    country TEXT,
    generated_post_code TEXT
);

CREATE TABLE IF NOT EXISTS Promotions (
    promotion_id TEXT PRIMARY KEY,
    name TEXT,
    discount_rate REAL,
    start_date DATETIME,
    end_date DATETIME,
    description TEXT,
    code TEXT UNIQUE,
    is_personalised INTEGER CHECK (is_personalised IN (0,1))
);
```

```
CREATE TABLE IF NOT EXISTS CustomerPromotions (
    customer_promotion_id TEXT PRIMARY KEY,
    promotion_id TEXT,
    customer_id TEXT,
    is_used INTEGER CHECK (is_used IN (0,1)),
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id),
    FOREIGN KEY (promotion_id) REFERENCES Promotions(promotion_id)
);

CREATE TABLE IF NOT EXISTS Suppliers (
    supplier_id TEXT PRIMARY KEY,
    supplier_name TEXT,
    contact_name TEXT,
    supplier_phone TEXT,
    supplier_email TEXT,
    supplier_address TEXT,
    supplier_city TEXT,
    supplier_country TEXT,
    product_category TEXT,
    price REAL,
    cost REAL,
    qty_ordered INTEGER
);

CREATE TABLE IF NOT EXISTS Products (
    product_id TEXT PRIMARY KEY,
    product_category TEXT,
    product_price REAL,
    supplier_id TEXT,
    product_size TEXT,
    product_colour TEXT,
    FOREIGN KEY (supplier_id) REFERENCES Suppliers(supplier_id)
);
```

```
CREATE TABLE IF NOT EXISTS Inventory (
    inventory_id TEXT PRIMARY KEY,
    product_id TEXT,
    stock INTEGER,
    last_updated DATETIME,
    FOREIGN KEY (product_id) REFERENCES Products(product_id)
);

CREATE TABLE IF NOT EXISTS Orders (
    order_id TEXT PRIMARY KEY,
    order_date DATETIME,
    order_time DATETIME,
    order_status TEXT,
    customer_id TEXT,
    promotion_id TEXT,
    order_total REAL,
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id),
    FOREIGN KEY (promotion_id) REFERENCES Promotions(promotion_id)
);

CREATE TABLE IF NOT EXISTS OrderProducts (
    order_product TEXT PRIMARY KEY,
    order_id TEXT,
    product_id TEXT,
    product_total REAL,
    qty_product INTEGER,
    FOREIGN KEY (order_id) REFERENCES Orders(order_id),
    FOREIGN KEY (product_id) REFERENCES Products(product_id)
);
```

```
CREATE TABLE IF NOT EXISTS Shipments (
    shipment_id TEXT PRIMARY KEY,
    order_id TEXT,
    carrier TEXT,
    tracking_number TEXT,
    shipment_date DATETIME,
    estimated_delivery DATETIME,
    delivery_status TEXT,
    FOREIGN KEY (order_id) REFERENCES Orders(order_id)
);

CREATE TABLE IF NOT EXISTS Payments (
    payment_id TEXT PRIMARY KEY,
    order_id TEXT,
    payment_date DATETIME,
    payment_method TEXT,
    payment_status TEXT,
    FOREIGN KEY (order_id) REFERENCES Orders(order_id)
);
"""
)

# Verify that tables were created
cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
tables = cursor.fetchall()
print("Tables created in the database:")
for table in tables:
    print(f" - {table[0]}")
```

```
Tables created in the database:
 - Customers
 - Promotions
 - CustomerPromotions
 - Suppliers
 - Products
 - Inventory
 - Orders
 - OrderProducts
 - Shipments
 - Payments
```

```python
# Define tables to load
cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
tables = cursor.fetchall()

# Load CSV files into corresponding tables
for table in tables:
    try:
        csv_file = f"{table[0]}.csv"
        df = pd.read_csv(csv_file)
        df.to_sql(table[0], conn, if_exists="append", index=False)
        print(f" Loaded {csv_file} into {table}")
    except Exception as e:
        print(f" Error loading {csv_file}: {e}")

# Display first 10 rows of each table
for table in tables:
    print(f"\n First 10 rows of {table}:")
    df = pd.read_sql_query(f"SELECT * FROM {table[0]} LIMIT 10;", conn)
    print(df)
```

```
 Loaded Customers.csv into ('Customers',)
 Loaded Promotions.csv into ('Promotions',)
 Loaded CustomerPromotions.csv into ('CustomerPromotions',)
 Loaded Suppliers.csv into ('Suppliers',)
 Loaded Products.csv into ('Products',)
 Loaded Inventory.csv into ('Inventory',)
 Loaded Orders.csv into ('Orders',)
 Loaded OrderProducts.csv into ('OrderProducts',)
 Loaded Shipments.csv into ('Shipments',)
 Loaded Payments.csv into ('Payments',)
```

```python
customer_order_ranking = execute_query("""
    SELECT customer_id, COUNT(order_id) AS total_orders, SUM(o.order_total * (1 - COALESCE(pr.discount_rate/100, 0))) AS total_spend
    FROM Orders o
    LEFT JOIN Promotions pr ON pr.promotion_id = o.promotion_id
    WHERE order_status NOT IN ('returned', 'canceled')
    GROUP BY customer_id
    ORDER BY total_spend DESC
    LIMIT 10;
""")
# Helps companies optimize their marketing budgets and focus on high-value customers.
print(customer_order_ranking)
```

```
  customer_id  total_orders  total_spend
0       C0043             9      2334.52
1       C0059             8      2174.59
2       C0056             6      1949.68
3       C0189             7      1904.63
4       C0193             4      1699.71
5       C0047             5      1659.69
6       C0135             6      1629.65
7       C0106             6      1619.69
8       C0185             5      1564.73
9       C0093             7      1534.69
```

```python
# Customer Repurchase Rate
# Measure customer retention, optimize membership and loyalty programs.
customer_repurchase_rate = execute_query("""
    SELECT customer_id, COUNT(customer_id) AS repeat_purchases
    FROM Orders
    WHERE order_status NOT IN ('returned', 'canceled')
    GROUP BY customer_id
    HAVING COUNT(order_id) > 1
    ORDER BY repeat_purchases DESC;
""")
print(customer_repurchase_rate)
```

```
    customer_id  repeat_purchases
0         C0043                 9
1         C0150                 8
2         C0059                 8
3         C0207                 7
4         C0189                 7
..          ...               ...
205       C0019                 2
206       C0017                 2
207       C0015                 2
208       C0014                 2
209       C0012                 2

[210 rows x 2 columns]
```

```
[ ] # Monthly Sales Trend
    # Understand sales growth trends and provide data support for promotion plans.
    monthly_sales_trend = execute_query("""
      SELECT
        strftime('%Y-%m',
          CASE
            WHEN substr(order_date, 3, 1) = '/' THEN
              substr(order_date, 7, 4) || '-' ||
              substr('0' || substr(order_date, 4, 2), -2) || '-' ||
              substr('0' || substr(order_date, 1, 2), -2)
            WHEN substr(order_date, 2, 1) = '/' THEN
              substr(order_date, 6, 4) || '-' ||
              substr('0' || substr(order_date, 3, 2), -2) || '-' ||
              substr('0' || substr(order_date, 1, 1), -2)
            ELSE NULL
          END
        ) AS month,
      SUM(o.order_total * (1 - COALESCE(pr.discount_rate/100, 0))) AS total_sales
      FROM Orders o
      LEFT JOIN Promotions pr ON pr.promotion_id = o.promotion_id
      WHERE month IS NOT NULL AND order_status NOT IN ('returned', 'canceled')
      GROUP BY month
      ORDER BY month;
    """)
    print(monthly_sales_trend)
```

```
       month  total_sales
0    2024-01     12547.63
1    2024-02     11122.70
2    2024-03     11272.72
3    2024-04     10872.80
4    2024-05     12492.41
5    2024-06     12432.51
6    2024-07     16351.54
7    2024-08     10447.77
8    2024-09     16811.51
9    2024-10     15831.80
10   2024-11     19561.23
11   2024-12     24115.23
```

```
[ ] # Sales by City
    # Determines market expansion strategy by analyzing sales distribution across cities.
    sales_by_city = execute_query("""
        SELECT C.city, COUNT(O.order_id) AS total_orders, SUM(O.order_total * (1 - COALESCE(pr.discount_rate/100, 0))) AS revenue_generated
        FROM Orders O
        JOIN Customers C ON O.customer_id = C.customer_id
        LEFT JOIN Promotions pr ON pr.promotion_id = O.promotion_id
        WHERE O.order_status NOT IN ('returned', 'canceled')
        GROUP BY C.city
        ORDER BY revenue_generated DESC
        LIMIT 10;
    """)
    print(sales_by_city)
```

```
           city  total_orders  revenue_generated
0    Leicester            93           19266.16
1     Aberdeen            65           15671.84
2      Bristol            57           13762.26
3      Cardiff            65           13492.36
4  Southampton            54           13102.41
5    Edinburgh            59           12597.55
6       London            53           12292.53
7      Glasgow            52           12242.59
8   Manchester            55           11977.51
9     Brighton            56           11402.75
```

```
[ ] sales_by_city_low = execute_query("""
        SELECT C.city, COUNT(O.order_id) AS total_orders, SUM(O.order_total * (1 - COALESCE(pr.discount_rate/100, 0))) AS revenue_generated
        FROM Orders O
        JOIN Customers C ON O.customer_id = C.customer_id
        LEFT JOIN Promotions pr ON pr.promotion_id = O.promotion_id
        WHERE O.order_status NOT IN ('returned', 'canceled')
        GROUP BY C.city
        ORDER BY revenue_generated
        LIMIT 10;
    """)
    print(sales_by_city_low)
```

```
          city  total_orders  revenue_generated
0  Portsmouth            37            7548.45
1       Leeds            44            7668.34
2    Coventry            45            8733.22
3     Belfast            44           10022.95
4   Sheffield            48           10132.94
5  Nottingham            44           10297.93
6      Dundee            43           10378.01
7   Liverpool            44           10537.78
8  Birmingham            53           11117.81
9   Newcastle            54           11157.71
```

```
# Product Sales Ranking
# Ensure the supply of popular products and optimize pricing strategies.
product_sales_ranking = execute_query("""
 SELECT
        p.product_category,
        p.product_size,
        p.product_colour,
        SUM(op.qty_product) AS total_soldnum,
        SUM(op.qty_product * p.product_price * (1 - COALESCE(pr.discount_rate, 0) / 100.0)) AS total_sold
    FROM OrderProducts op
    JOIN Products p ON p.product_id = op.product_id
    JOIN Orders o ON o.order_id = op.order_id
    LEFT JOIN Promotions pr ON pr.promotion_id = o.promotion_id
    WHERE o.order_status NOT IN ('returned', 'canceled')
    GROUP BY
        p.product_category,
        p.product_size,
        p.product_colour
    ORDER BY total_sold DESC
    LIMIT 10;
""")
print(product_sales_ranking)
```

```
  product_category product_size product_colour  total_soldnum  total_sold
0          hoodies           XL         green             52    3569.490
1          hoodies            L          blue             52    3527.496
2          hoodies           XL          blue             50    3359.520
3          hoodies           XL         white             49    3324.525
4          hoodies            M          blue             50    3289.530
5          jackets            S         black             48    3142.551
6          hoodies            S         white             45    3100.557
7          hoodies           XL         black             45    3058.563
8          jackets           XL         white             45    2988.573
9          jackets            M         black             44    2960.577
```

```
# Supplier Sales Contribution ()
# Evaluate supplier contributions and optimize procurement strategies.
supplier_sales_contribution = execute_query("""
    SELECT Suppliers.product_category, supplier_name, SUM(OrderProducts.qty_product) AS total_orders , SUM(Orders.order_total * (1 - COALESCE(pr.discount_rate/100,
    FROM Products
    JOIN OrderProducts ON Products.product_id = OrderProducts.product_id
    JOIN Orders ON OrderProducts.order_id = Orders.order_id
    LEFT JOIN Promotions pr ON pr.promotion_id = Orders.promotion_id
    JOIN Suppliers ON Products.supplier_id = Suppliers.supplier_id
    WHERE order_status NOT IN ('returned', 'canceled')
    GROUP BY Suppliers.product_category, Suppliers.supplier_id
    ORDER BY Profit DESC;
""")
print(supplier_sales_contribution)
```

```
  product_category supplier_name  total_orders    Profit
0          hoodies        Carlye           724  85849.00
1          jackets         Floyd           683  81653.47
2           cargos         Dilan           678  72462.12
3          joggers         Jewel           648  63991.73
4           shirts        Jaimie           649  61363.15
5            shoes           Ami           663  59623.50
6           shorts       Ambrose           645  54747.61
```

```
# Order Fulfillment Rate, Order Cancel Rate and other Rate
# Order status Rate
# Reduce order delays and cancellation rates and improve supply chain efficiency.
order_status = execute_query("""
    SELECT order_status, COUNT(order_id) AS order_count, ROUND((COUNT(order_id) * 100.0 / (SELECT COUNT(*) FROM Orders)), 2) AS percentage
    FROM orders
    GROUP BY order_status
    ORDER BY order_count DESC;
""")
print(order_status)
```

```
       order_status  order_count  percentage
0         delivered          581       42.82
1      order_placed          214       15.77
2          returned          192       14.15
3           shipped          170       12.53
4  out_for_delivery          100        7.37
5          canceled          100        7.37
```

```
# Promotional Effect Analysis
# Calculate the sales and total number of promotional orders
# Measure the return on investment (ROI) of promotion activities.
promotional_effect_analysis = execute_query("""
    SELECT
    pr.promotion_id,
    pr.name,
    CASE
        WHEN pr.promotion_id IN ('PROMO08', 'PROMO09') THEN COALESCE(cp.used_customers, 0)
        ELSE COUNT(op.order_id)  -- Counting orders affected by the promotion
    END AS orders_affected,
    COALESCE(SUM(op.order_total * (1 - COALESCE(pr.discount_rate/100, 0))), 0) AS total_revenue -- Removed extra comma
FROM Promotions pr
LEFT JOIN orders op
    ON pr.promotion_id = op.promotion_id
LEFT JOIN (
    SELECT promotion_id, COUNT(customer_id) AS used_customers
    FROM CustomerPromotions
    WHERE is_used = TRUE  -- Filter for used promotions
    GROUP BY promotion_id
) cp
    ON pr.promotion_id = cp.promotion_id
    WHERE order_status NOT IN ('returned', 'cancelled')
GROUP BY pr.promotion_id, pr.name, cp.used_customers
ORDER BY total_revenue DESC;
""")
print(promotional_effect_analysis)
```

```
  promotion_id              name  orders_affected  total_revenue
0      PROMO03           Drop 2               40       10133.01
1      PROMO07        Christmas               40        9868.02
2      PROMO06     Black Friday               39        9263.17
3      PROMO02           Easter               40        8788.13
4      PROMO01           Drop 1               41        8518.44
5      PROMO05           Drop 3               41        8423.21
6      PROMO04      Summer Sale               41        7548.43
7      PROMO08   loyal customers               15        4714.11
8      PROMO09  Premium customers               11        3104.44
```

## 5. Supply Chain Management

```
[ ] # Stock Alerts
    # Ensure sufficient inventory of popular products to avoid missing sales opportunities.
    stock_alerts = execute_query("""
        SELECT product_id, stock FROM inventory WHERE stock < 10;
    """)
    print(stock_alerts)
```

```
  product_id  stock
0     P78iq5      5
1     P03nB5      9
2     P40F46      7
3     P03Nh3      2
4     P99QZ8      3
```

```
[ ] # Get the optimal inventory quantity based on the average monthly sales volume of the product
    # Ensure the supply of popular products.
    product_nums_ranking = execute_query("""
        SELECT product_category , SUM(qty_product)/12 AS avenum_sold
        FROM OrderProducts
        JOIN Products ON Products.product_id = OrderProducts.product_id
        JOIN Orders ON OrderProducts.order_id = Orders.order_id
        WHERE order_status NOT IN ('returned', 'canceled')
        GROUP BY product_category
        ORDER BY avenum_sold DESC
        LIMIT 10;
    """)
    print(product_nums_ranking)
```

```
  product_category  avenum_sold
0          hoodies          60
1          jackets          56
2           cargos          56
3            shoes          55
4           shirts          54
5          joggers          54
6           shorts          53
```

```
[ ] # Calculation of profit per piece
    Unit_profit = execute_query("""
    SELECT
    p.product_price - s.cost AS unit_profit
    FROM Products p
    JOIN Suppliers s ON p.supplier_id = s.supplier_id""")
    print(Unit_profit)
```

```
     unit_profit
0          38.49
1          38.49
2          38.49
3          38.49
4          38.49
..           ...
135        26.49
136        26.49
137        26.49
138        26.49
139        26.49

[140 rows x 1 columns]
```

```
[ ] # Calculation of profit per order
    Profit_per_order = execute_query("""
    SELECT
        o.order_id,
        o.order_total * (1 - COALESCE(pr.discount_rate/100, 0)) AS total_revenue,
        COALESCE(SUM(op.qty_product * s.cost), 0) AS total_cost,
        COALESCE(o.order_total * (1 - COALESCE(pr.discount_rate/100, 0)) - SUM(op.qty_product * s.cost), 0) AS profit
    FROM Orders o
    LEFT JOIN Promotions pr ON pr.promotion_id = o.promotion_id
    LEFT JOIN OrderProducts op
        ON o.order_id = op.order_id
    LEFT JOIN Products p
        ON op.product_id = p.product_id
    LEFT JOIN Suppliers s
        ON p.supplier_id = s.supplier_id
    WHERE order_status NOT IN ('returned', 'cancelled')
    GROUP BY o.order_id
    ORDER BY profit DESC;
    """)
    print(Profit_per_order)
```

```
       order_id  total_revenue  total_cost  profit
0       0699837         629.91       166.5  463.41
1     OBP738578         629.91       166.5  463.41
2     OUF725996         629.91       166.5  463.41
3     OTM513624         639.90       185.0  454.90
4     OBD713385         629.89       177.5  452.39
...         ...            ...         ...     ...
1160  ORM596290          29.99         8.5   21.49
1161  OSY901145          29.99         8.5   21.49
1162  OUF023584          29.99         8.5   21.49
1163  OWQ573305          29.99         8.5   21.49
1164  OXZ861707          29.99         8.5   21.49

[1165 rows x 4 columns]
```

```
[ ] # Customer Order Analysis
    # Calculate the long-term contribution value of each customer and measure the customer's future profitability.
    customer_order_analysis = execute_query("""
        SELECT
            o.customer_id,
            COUNT(o.order_id) AS total_orders,
            SUM(o.order_total * (1 - COALESCE(pr.discount_rate/100, 0))) AS total_spend, -- Use COALESCE to handle NULL discount_rates
            SUM(o.order_total * (1 - COALESCE(pr.discount_rate/100, 0))) / COUNT(o.order_id) AS avg_total_spend
        FROM Orders o
        LEFT JOIN Promotions pr ON pr.promotion_id = o.promotion_id
        WHERE o.order_status NOT IN ('returned', 'canceled')
        GROUP BY o.customer_id
        ORDER BY avg_total_spend DESC;
    """)
    # Helps companies optimize their marketing budgets and focus on high-value customers.
    print(customer_order_analysis)
```

```
    customer_id  total_orders  total_spend  avg_total_spend
0         C0511             1       629.91           629.91
1         C0411             1       629.89           629.89
2         C0539             1       594.90           594.90
3         C0075             1       579.90           579.90
4         C0496             1       574.90           574.90
..          ...           ...          ...              ...
520       C0117             1        34.99            34.99
521       C0526             1        29.99            29.99
522       C0524             1        29.99            29.99
523       C0306             1        29.99            29.99
524       C0152             1        29.99            29.99

[525 rows x 4 columns]
```

## Code for Visualisations

```
    # Plotting the monthly sales trend
    plt.figure(figsize=(10, 6))
    sns.lineplot(x='month', y='total_sales', data=monthly_sales_trend, marker='o', color='tab:blue') # Changed y to 'total_sales'
    plt.title('Monthly Sales Trend', fontsize=18, fontweight='bold', color='darkblue')
    plt.xlabel('Month', fontsize=14, fontweight='bold')
    plt.ylabel('Total Sales (£)', fontsize=14, fontweight='bold')
    plt.xticks(rotation=45, ha='right')
    plt.grid(True, axis='y', linestyle='--', alpha=0.7)
    plt.tight_layout()
    plt.show()
```

```
    plt.figure(figsize=(10, 6))
    sns.barplot(x='total_spend', y='customer_id', data=customer_order_ranking, palette='Blues_d')
    plt.title('Top 10 Customers by Total Spend', fontsize=18, fontweight='bold', color='darkblue')
    plt.xlabel('Total Spend (£)', fontsize=14, fontweight='bold')
    plt.ylabel('Customer', fontsize=14, fontweight='bold')
    plt.tight_layout()
    plt.show()
```

```
[ ] # Plot a histogram of repeat purchases distribution
    plt.figure(figsize=(10, 6))
    sns.histplot(customer_repurchase_rate['repeat_purchases'], bins=20, kde=True, color='tab:green')
    plt.title('Distribution of Repeat Purchases', fontsize=18, fontweight='bold', color='darkgreen')
    plt.xlabel('Repeat Purchases', fontsize=14, fontweight='bold')
    plt.ylabel('Frequency', fontsize=14, fontweight='bold')
    plt.tight_layout()
    plt.show()
```

```
    # Plotting the top cities by sales
    plt.figure(figsize=(10, 6))
    sns.barplot(x='revenue_generated', y='city', data=sales_by_city, palette='Blues_d') # Now sales_by_city is defined
    plt.title('Top 10 Cities by Total Sales', fontsize=18, fontweight='bold', color='darkblue')
    plt.xlabel('Total Sales (£)', fontsize=14, fontweight='bold')
    plt.ylabel('City', fontsize=14, fontweight='bold')
    plt.tight_layout()
    plt.show()
```

```
[ ] # Plotting order status rate
    plt.figure(figsize=(10, 6))
    sns.barplot(x='order_status', y='percentage', data=order_status, palette='Blues_d')
    plt.title('Order Status Rate', fontsize=18, fontweight='bold', color='darkblue')
    plt.xlabel('Order Status', fontsize=14, fontweight='bold')
    plt.ylabel('Percentage of Orders (%)', fontsize=14, fontweight='bold')
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
```

```
[ ]  plt.figure(figsize=(12, 6))
     ax = sns.barplot(x='supplier_name', y='Profit', data=supplier_sales_contribution, palette='viridis')
     plt.title('Profit per Product Category (£)', fontsize=16)
     plt.xlabel('Product Category', fontsize=12)
     plt.ylabel('Profit', fontsize=12)

     for i, p in enumerate(ax.patches):
         width = p.get_width()
         height = p.get_height()
         x, y = p.get_xy()
         category = supplier_sales_contribution['product_category'][i]
         ax.annotate(category.upper(), (x + width/2, y + height/2), ha='center', va='center', color='white', fontsize=15) # Changed to category.upper()

     plt.tight_layout()
     plt.show()
```

```
[ ]  # Plotting the promotional effect analysis
     plt.figure(figsize=(14, 6))
     sns.barplot(x='name', y='total_revenue', data=promotional_effect_analysis, palette='Blues_d')
     plt.title('Effect of Promotions on Sales', fontsize=20, fontweight='bold', color='darkblue')
     plt.xlabel('Promotion Name', fontsize=14, fontweight='bold')
     plt.ylabel('Total Sales (£)', fontsize=14, fontweight='bold')
     plt.tight_layout()
     plt.show()
```

```
[ ]  plt.figure(figsize=(10, 6))  # Adjust figure size if needed
     sns.barplot(x='product_id', y='stock', data=stock_alerts, palette='Reds_r')
     plt.title('Stock Alerts: Products with Low Stock', fontsize=16)
     plt.xlabel('Product ID', fontsize=12)
     plt.ylabel('Stock Level', fontsize=12)
     plt.axhline(y=10, color='red', linestyle='--', label='Low Stock Threshold')
     plt.legend()
     plt.tight_layout()
     plt.show()
```

```
[7]  # Average order value
     avg_order_value = customer_order_analysis['avg_total_spend']

     plt.figure(figsize=(10, 6))
     sns.kdeplot(avg_order_value, shade=True)
     plt.title('Density Plot of Average Order Value')
     plt.xlabel('Average Order Value (£)')
     plt.ylabel('Density')
     plt.show()
```