Lazer Andelman

Итоговый проект по модулю SQL и Базы Данных

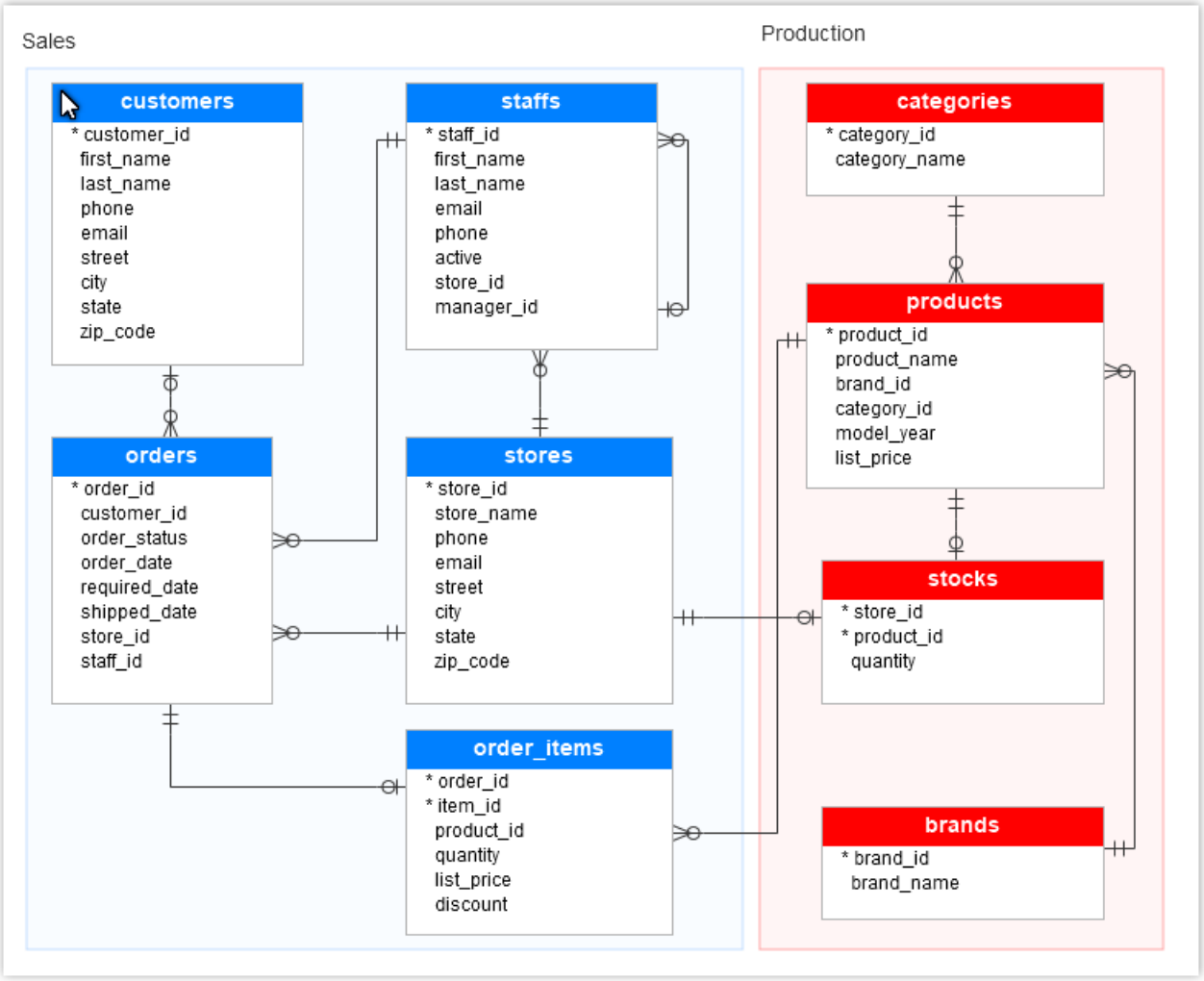
1. База данных Bike Store Relational Database:

1.1. *Таблицы базы данных:*

- Таблицы:

brands	9	записей
categories	7	записей
customers	1445	записей
order_items	4722	записей
orders	1615	записей
products	321	записей
staffs	10	записей
stocks	313	записей
stores	3	записей

1.2. *Схема базы данных:*



2. Постановка задачи:

2.1. *Анализ продаж по категориям и брендам.*
2.2. *Определение лучших клиентов.*
2.3. *Анализ эффективности персонала.*
2.4. *Исследование динамики продаж.*
2.5. *Сегментация клиентов:*
  2.5.1. по возрасту и социальной активности.
  2.5.2. по семейному статусу и количеству детей.
  2.5.3. по образованию и месячному доходу.
2.6. *Общий объем продаж по магазинам.*
2.7. *Эффективность персонала.*
2.8. *Рассчитываем кумулятивный профит по месяцам.*
2.9. *Рассчитываем скользящее среднее профита.*
2.10. *Прогнозирование продаж.*
2.11. *Анализ зависимости между скидками и объемом продаж.*

3. Добавление данных в базу.

3.1. *Добавляем таблицу delivery_carriers 19 записей*

Поля:

- carrier_id
- carrier_name

3.2. *В таблицу 'customers' добавляем поля с личными данными клиентов.*

Поля:

- age
- degree
- family_status
- children_amount
- monthly_income
- social_activity
- payment_method

3.3. *В таблицу 'orders' добавляем поле с данными.*

- order_time
- carrier_id
- delivery_type

Данные полей заполняем в Excel при помощи функций RAND(), RANDBETWEEN() и VLOOCKUP().

4. **Первичная обработка данных.**

4.1. *Переименование столбцов в соответствии с naming convention.*

4.2. *Загрузка данных в базу данных.*

4.3. *Проверка записей на пропущенные или пустые значения (в целях отработки запросов на проверку данных и удаление не соответствующих, в таблицы вручную были внесены изменения):*

```
--Для таблицы  orders
SELECT order_id,
    customer_id,
    order_status,
    order_date,
    order_time,
    required_date,
    shipped_date,
    carrier_id,
    store_id,
    staff_id
FROM orders
WHERE customer_id IS NULL
  OR order_status IS NULL
  OR order_date IS NULL
  OR order_time IS NULL
  OR (required_date IS NULL AND shipped_date IS NULL) -- Если обе даты
    отсутствуют
  OR carrier_id IS NULL
  OR store_id IS NULL
  OR staff_id IS NULL;
```

Подобным же образом производится проверка на пропущенные или пустые значения и в остальных таблицах.

4.4. *Проверка полученных данных на сохранение целостности данных.*

```
UPDATE order_items
SET quantity = 1
WHERE quantity IS NULL;
```

В таблице order_items было произведено исправление, чтобы не нарушить целостность данных.

4.5. *Удаление записей (если принято такое решение)*

```
--Для таблицы customers
DELETE FROM customers
WHERE customer_first_name IS NULL OR customer_first_name = ''
   OR customer_last_name IS NULL OR customer_last_name = ''
   OR customer_phone IS NULL OR customer_phone = ''
   OR customer_email IS NULL OR customer_email = ''
   OR customer_address IS NULL OR customer_address = ''
   OR customer_city IS NULL OR customer_city = ''
   OR customer_state IS NULL OR customer_state = ''
   OR customer_zip_code IS NULL OR customer_zip_code = ''
   OR customer_age IS NULL
   OR customer_degree IS NULL OR customer_degree = ''
   OR family_status IS NULL OR family_status = ''
   OR children_amount IS NULL
   OR monthly_income IS NULL
   OR social_activity IS NULL OR social_activity = ''
   OR payment_method IS NULL OR payment_method = '';
```

Подобным же образом производится удаление пустых записей и в остальных таблицах.

4.6. *Удаление дубликатов строк.*

```
--Для таблицы brands:
CREATE TEMPORARY TABLE temp_brands AS
SELECT MIN(brand_id) as brand_id, brand_name
FROM brands
GROUP BY brand_name;
DELETE FROM brands;
INSERT INTO brands (brand_id, brand_name)
SELECT brand_id, brand_name
FROM temp_brands;
DROP TABLE temp_brands;
```

Подобным же образом производится удаление дубликатов записей и в остальных таблицах.

5. Исследование данных с помощью SQL.

5.1. *Анализ продаж по категориям и брендам.*

```
SELECT
    c.category_name,
    COUNT(oi.order_id) AS total_sales,
    SUM(oi.quantity) AS total_quantity_sold,
    ROUND(SUM(oi.product_price * oi.quantity - oi.discount), 2) AS total_revenue
FROM
    categories c
JOIN products p ON c.category_id = p.category_id
JOIN order_items oi ON p.product_id = oi.product_id
GROUP BY
    c.category_name
ORDER BY
    total_revenue DESC;
```

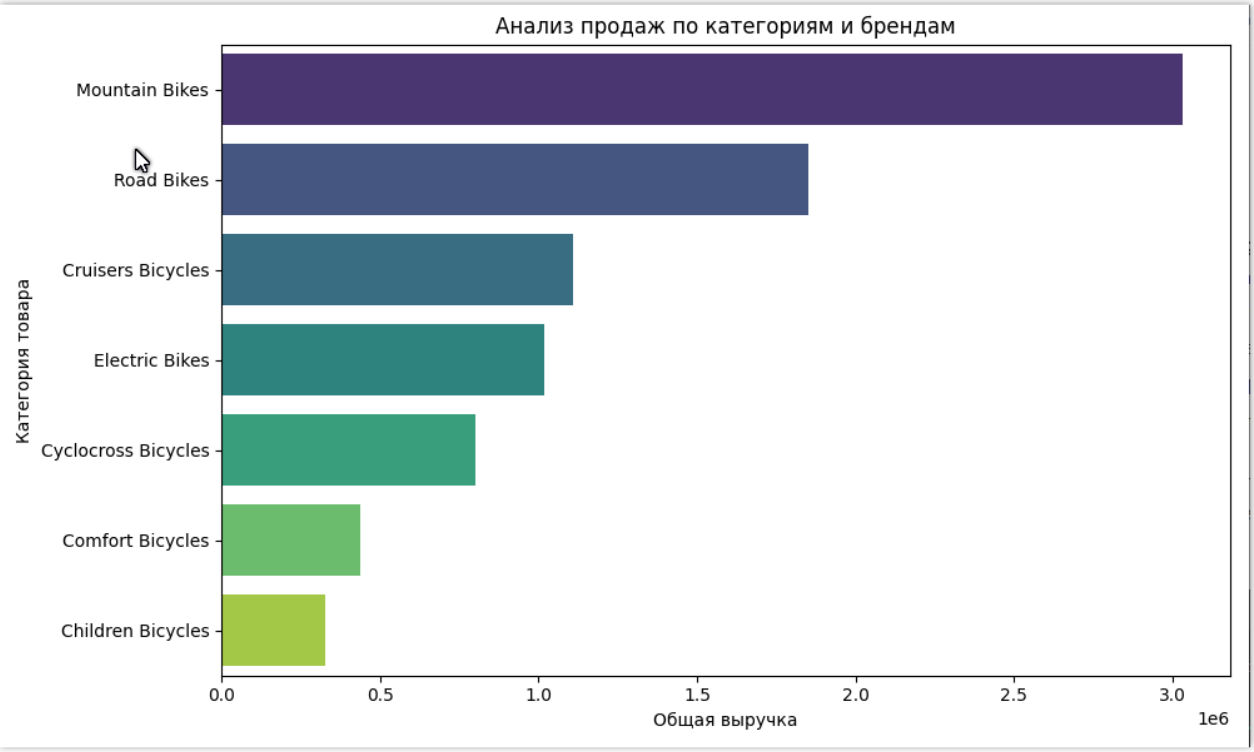
	category_name	total_sales	total_quantity_sold	total_revenue
1	Mountain Bikes	1,183	1,755	3,030,651.45
2	Road Bikes	374	559	1,852,516.12
3	Cruisers Bicycles	1,378	2,063	1,109,007.23
4	Electric Bikes	212	315	1,020,214.56
5	Cyclocross Bicycles	256	394	799,846.1
6	Comfort Bicycles	537	813	438,452.02
7	Children Bicycles	782	1,179	327,803.83

Объяснение запроса:

- **FROM categories c**: Начинаем запрос с таблицы **categories**, так как нас интересуют результаты по категориям товаров.
- **JOIN products p ON c.category_id = p.category_id**: Соединяем таблицу **products** с таблицей **categories** по **category_id**, чтобы получить доступ к товарам определенной категории.
- **JOIN order_items oi ON p.product_id = oi.product_id**: Соединяем таблицу **order_items** с таблицей **products** по **product_id**, чтобы учитывать только те товары, которые были проданы.
- **COUNT(oi.order_id) AS total_sales**: Считаем общее количество продаж как количество уникальных **order_id** в таблице **order_items**.
- **SUM(oi.quantity) AS total_quantity_sold**: Считаем общее количество проданных товаров, суммируя значения **quantity** из **order_items**.
- **SUM(oi.product_price * oi.quantity - oi.discount) AS total_revenue**: Считаем общую выручку, умножая цену товара на количество и вычитая скидку для каждой записи, а затем суммируя полученные значения.
- **GROUP BY c.category_name**: Группируем результаты по названию категории, чтобы агрегатные функции применялись в рамках каждой категории.
- **ORDER BY total_revenue DESC**: Сортируем результаты по общей выручке по убыванию, чтобы наиболее прибыльные категории были выше в списке.

Визуализация полученных результатов:

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

# Подключение к базе данных
conn = sqlite3.connect('D:/Google Drive/05 - WebBuilding/Learning/MathsHub/02. SQL/Итоговый проект/BikeDBeaverDatabase/BikeDatabase_2.db')

# Загрузка данных
query = '''
SELECT
    c.category_name,
    COUNT(oi.order_id) AS total_sales,
    SUM(oi.quantity) AS total_quantity_sold,
    ROUND(SUM(oi.product_price * oi.quantity - oi.discount), 2) AS total_revenue
FROM
    categories c
JOIN products p ON c.category_id = p.category_id
JOIN order_items oi ON p.product_id = oi.product_id
GROUP BY
    c.category_name
ORDER BY
    total_revenue DESC;
'''

sales_data = pd.read_sql_query(query, conn)
print(sales_data)

# Визуализация данных
plt.figure(figsize=(10, 6))
sns.barplot(x='total_revenue', y='category_name', data=sales_data, palette='viridis')
plt.title('Анализ продаж по категориям и брендам')
plt.xlabel('Общая выручка')
plt.ylabel('Категория товара')
# plt.legend(title='Бренд', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()

# Закрытие соединения с базой данных
conn.close()
```



Выводы:

1. **Лидеры продаж и выручки:** Категория "Mountain Bikes" является абсолютным лидером как по общему количеству продаж, так и по общей выручке. Это указывает на высокий спрос и прибыльность данной категории товаров в ассортименте.

2. **Специализированные и дорогие велосипеды:** Категории "Road Bikes" и "Electric Bikes" показывают высокую выручку при относительно небольшом количестве продаж, что может свидетельствовать о более высокой цене этих товаров по сравнению с другими категориями. Это также может указывать на то, что покупатели готовы платить больше за специализированные или инновационные продукты.

3. **Популярные бюджетные категории:** Категория "Cruisers Bicycles" имеет высокие показатели как по количеству продаж, так и по выручке, что может свидетельствовать о популярности этих велосипедов среди широкого круга потребителей и их доступной цене.

4. **Нишевые категории:** Категории, такие как "Cyclocross Bicycles" и "Comfort Bicycles", показывают более низкие показатели по сравнению с лидерами. Это может указывать на их нишевый характер и более узкую аудиторию покупателей.

5. **Детские велосипеды:** Категория "Children Bicycles" показывает хорошие результаты по количеству продаж при относительно низкой общей выручке, что говорит о низкой средней цене товаров в этой категории. Это может отражать большой объем продаж бюджетных товаров для детей.

Рекомендации:

- **Фокус на высокодоходные категории:** Уделить особое внимание категориям "Mountain Bikes" и "Road Bikes", исследуя возможности для расширения ассортимента и оптимизации ценообразования.
- **Расширение ассортимента в популярных категориях:** Для категории "Cruisers Bicycles" стоит рассмотреть возможность расширения ассортимента, учитывая ее популярность и потенциал для увеличения продаж.
- **Продвижение и маркетинг:** Активное продвижение и маркетинговые кампании для нишевых категорий, таких как "Electric Bikes" и "Cyclocross Bicycles", могут помочь увеличить их долю на рынке.
- **Специальные предложения для детских велосипедов:** Разработка специальных предложений или акций для категории "Children Bicycles" может стимулировать продажи, учитывая их высокий объем и низкую среднюю цену.

5.2. *Определение лучших клиентов:*

```
SELECT
    ranked_customers.customer_id,
    c.customer_first_name,
    c.customer_last_name,
    ranked_customers.total_spent,
    ranked_customers.rank
FROM (
    SELECT
        o.customer_id,
        SUM(oi.quantity * oi.product_price - oi.discount) OVER(PARTITION BY
o.customer_id) AS total_spent,
        RANK() OVER(ORDER BY SUM(oi.quantity * oi.product_price - oi.discount) DESC)
AS rank
    FROM orders o
    JOIN order_items oi ON o.order_id = oi.order_id
    GROUP BY o.customer_id
) AS ranked_customers
JOIN customers c ON ranked_customers.customer_id = c.customer_id
WHERE rank <= 10
ORDER BY rank;
```

	customer_id	customer_first_name	customer_last_name	total_spent	rank
1	10	Pamelia	Newman	448.9	1
2	75	Abby	Gamble	1,319.78	2
3	94	Sharyn	Hopkins	448.93	3
4	6	Lyndsey	Bean	2,819.93	4
5	16	Emmitt	Sanchez	1,680.79	5
6	73	Melanie	Hayes	899.88	6
7	1	Debra	Burks	5,999.93	7
8	61	Elinore	Aguilar	1,799.92	8
9	93	Corrina	Sawyer	1,665.93	9
10	122	Shena	Carter	647.92	10

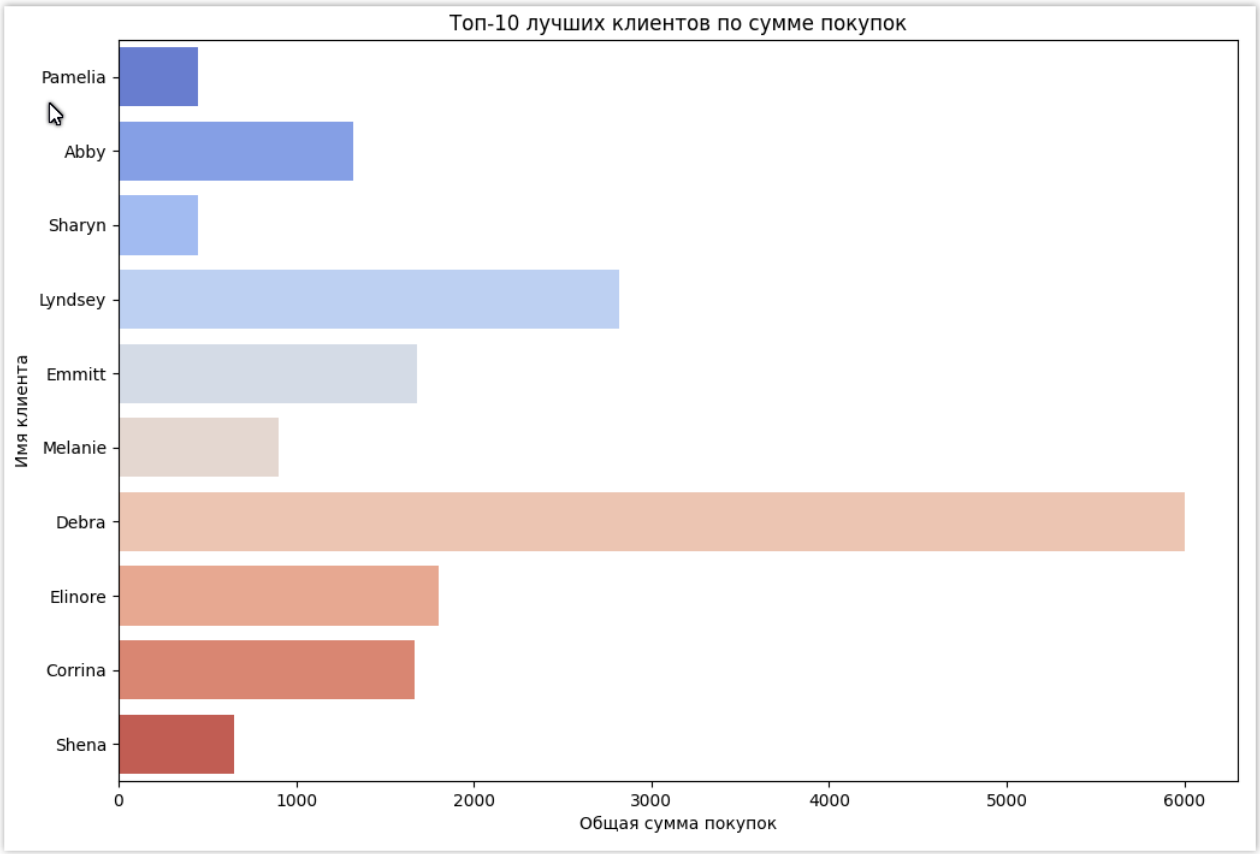
Объяснение запроса:

1. **Подзапрос для расчета расходов и ранжирования клиентов:**
   - В подзапросе **ranked_customers** для каждого **customer_id** из таблицы **orders** рассчитываем **total_spent**, сумма денег, потраченных клиентом, которая включает сумму покупок с учетом скидок (**oi.quantity * oi.product_price - oi.discount**).
   - Применяем оконную функцию **RANK()** для определения ранга каждого клиента в порядке убывания общей суммы расходов. Это позволяет выявить, кто из клиентов тратит больше всех.
2. **Объединение с таблицей customers:**
   - Подзапрос затем объединяется с таблицей customers по customer_id, чтобы получить доступ к именам клиентов (customer_first_name и customer_last_name).
3. **Фильтрация и сортировка результатов:**
   - В результат включаем только те строки, где ранг (rank) не превышает 10, то есть выбираются только первые 10 клиентов с наибольшими общими расходами.
   - Выборку упорядочиваем по полю rank, чтобы вывести список клиентов в порядке от самых больших расходов к менее значимым.

Визуализация полученных результатов:

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

# Подключение к базе данных
conn = sqlite3.connect('D:/Google Drive/05 - WebBuilding/Learning/MathsHub/02.
SQL/Итоговый проект/BikeDBeaverDatabase/BikeDatabase_2.db')

# Загрузка данных
query = '''
SELECT
    ranked_customers.customer_id,
    c.customer_first_name,
    c.customer_last_name,
    ranked_customers.total_spent,
    ranked_customers.rank
FROM (
    SELECT
        o.customer_id,
        SUM(oi.quantity * oi.product_price - oi.discount) OVER(PARTITION BY
o.customer_id) AS total_spent,
        RANK() OVER(ORDER BY SUM(oi.quantity * oi.product_price - oi.discount) DESC)
AS rank
    FROM orders o
    JOIN order_items oi ON o.order_id = oi.order_id
    GROUP BY o.customer_id
) AS ranked_customers
JOIN customers c ON ranked_customers.customer_id = c.customer_id
```

```
WHERE rank <= 10 AND total_spent > 0
ORDER BY rank;
'''

best_customers_data = pd.read_sql_query(query, conn)

# Закрытие соединения с базой данных
conn.close()

# Визуализация данных
plt.figure(figsize=(12, 8))
sns.barplot(x='total_spent', y='customer_first_name', data=best_customers_data,
palette='coolwarm')
plt.title('Топ-10 лучших клиентов по сумме покупок')
plt.xlabel('Общая сумма покупок')
plt.ylabel('Имя клиента')
plt.show()
```



Выводы:

1. **Различия в расходах клиентов:** Существует значительный разброс в суммах, которые клиенты тратят в магазине. Например, клиент Debra Burks тратит значительно больше всех остальных, что указывает на его высокую лояльность или наличие дорогостоящих покупок.
2. **Ключевые клиенты:** Идентифицированы ключевые клиенты (например, Debra Burks, Lyndsey Bean, Elinore Aguilar), которые вносят значительный вклад в общий объем продаж. Эти клиенты могут считаться наиболее ценными для бизнеса.
3. **Потенциал для увеличения продаж:** Некоторые клиенты, такие как Pamelia Newman и Sharyn Hopkins, тратят сопоставимо меньше. Это может указывать на потенциал для увеличения продаж среди клиентов с меньшей суммой расходов.

Рекомендации:

1. **Программы лояльности:** Разработайте или улучшите программы лояльности, нацеленные на удержание ключевых клиентов и стимулирование их на дополнительные покупки. Это может включать эксклюзивные предложения, персонализированные скидки или бонусы за частые покупки.
2. **Персонализированный маркетинг:** Используйте данные о покупках для создания персонализированных маркетинговых кампаний, нацеленных на конкретные группы клиентов. Например, отправляйте предложения о новых товарах или акциях, основываясь на предыдущих покупках клиента.
3. **Анализ предпочтений клиентов:** Изучите детали покупок лучших клиентов для понимания их предпочтений и интересов. Это может помочь в адаптации ассортимента товаров под потребности наиболее ценных клиентов.
4. **Программы привлечения:** Разработайте инициативы для привлечения клиентов с меньшими расходами, включая информационные кампании о продуктах, которые могут их заинтересовать, или предложения, нацеленные на увеличение среднего чека.
5. **Обратная связь:** Активно собирайте обратную связь от лучших клиентов для улучшения качества обслуживания и ассортимента товаров. Понимание их потребностей и предпочтений может способствовать дальнейшему развитию лояльности.

5.3. *Анализ эффективности персонала.*

```
SELECT
    s.staff_id,
    s.staff_first_name,
    s.staff_last_name,
    SUM(oi.quantity * oi.product_price) AS total_sales,
    RANK() OVER (ORDER BY SUM(oi.quantity * oi.product_price) DESC) AS sales_rank
FROM
    orders o
JOIN staffs s ON o.staff_id = s.staff_id
JOIN order_items oi ON o.order_id = oi.order_id
GROUP BY
    s.staff_id,
    s.staff_first_name,
    s.staff_last_name
ORDER BY
    total_sales DESC;
```

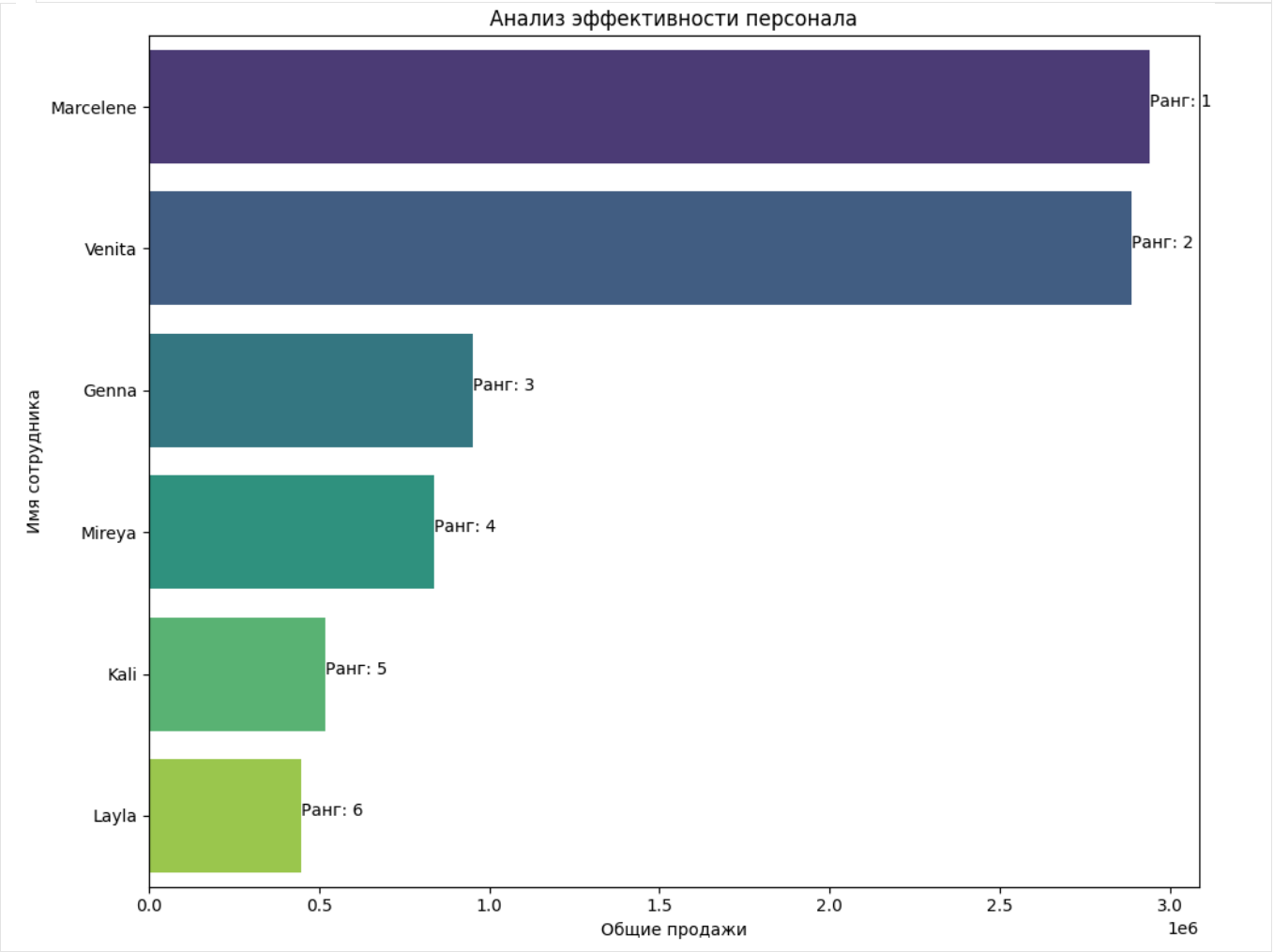
Объяснение запроса:

- **FROM orders o**: Начинаем выборку с таблицы **orders**.
- **JOIN staffs s ON o.staff_id = s.staff_id**: Соединяем таблицу **orders** с **staffs**, чтобы получить информацию о продавцах, связанных с каждым заказом.
- **JOIN order_items oi ON o.order_id = oi.order_id**: Соединяем таблицу **orders** с **order_items**, чтобы получить доступ к деталям заказов, таким как количество и цена проданных товаров.
- **SUM(oi.quantity * oi.product_price) AS total_sales**: Вычисляем общий объем продаж для каждого продавца, умножая количество на цену товара и суммируя результаты.
- **RANK() OVER (ORDER BY SUM(oi.quantity * oi.product_price) DESC) AS sales_rank**: Применяем оконную функцию **RANK()** для ранжирования продавцов по объему продаж в порядке убывания.
- **GROUP BY s.staff_id, s.staff_first_name, s.staff_last_name**: Группируем результаты по идентификатору и имени продавца, чтобы агрегаты применялись к каждому продавцу индивидуально.
- **ORDER BY total_sales DESC**: Сортируем результаты по общему объему продаж в порядке убывания, чтобы самые эффективные продавцы были в начале списка.

	staff_id	staff_first_name	staff_last_name	total_sales	sales_rank
1	6	Marcelene	Boyer	2,938,888.73	1
2	7	Venita	Daniel	2,887,353.48	2
3	3	Genna	Serrano	952,722.26	3
4	2	Mireya	Copeland	837,423.65	4
5	8	Kali	Vargas	516,695.17	5
6	9	Layla	Terrell	445,905.59	6

Визуализация полученных результатов:

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

# Подключение к базе данных
conn = sqlite3.connect('D:/Google Drive/05 - WebBuilding/Learning/MathsHub/02.
SQL/Итоговый проект/BikeDBeaverDatabase/BikeDatabase_2.db')

# Загрузка данных
query = '''
SELECT
    s.staff_id,
    s.staff_first_name,
    s.staff_last_name,
    SUM(oi.quantity * oi.product_price) AS total_sales,
    RANK() OVER (ORDER BY SUM(oi.quantity * oi.product_price) DESC) AS sales_rank
FROM
    orders o
JOIN staffs s ON o.staff_id = s.staff_id
JOIN order_items oi ON o.order_id = oi.order_id
GROUP BY
    s.staff_id,
    s.staff_first_name,
    s.staff_last_name
ORDER BY
    total_sales DESC;
'''
staff_efficiency_data = pd.read_sql_query(query, conn)

# Закрытие подключения к базе данных
conn.close()

# Визуализация данных
plt.figure(figsize=(10, 8))
sns.barplot(x='total_sales', y='staff_first_name', data=staff_efficiency_data,
            palette='viridis')
plt.title('Анализ эффективности персонала')
plt.xlabel('Общие продажи')
plt.ylabel('Имя сотрудника')
plt.tight_layout()

# Добавление ранга продаж к каждому сотруднику на графике
for index, value in enumerate(staff_efficiency_data['sales_rank']):
    plt.text(staff_efficiency_data['total_sales'][index], index, f'Ранг: {value}')

plt.show()
```



Выводы и рекомендации:

- **Признание и поощрение:** Продавцы с наибольшим объемом продаж, такие как Marcelene Boyer и Venita Daniel, должны быть признаны и вознаграждены за их выдающийся вклад. Это может включать в себя бонусы, повышения или публичное признание их достижений.
- **Анализ стратегий продаж:** Презентация и обмен опытом с самыми успешными продавцами для изучения их стратегий и методов работы с клиентами. Эти знания могут быть использованы для обучения других сотрудников.
- **Улучшение поддержки:** Обеспечить лучших продавцов всеми необходимыми ресурсами и поддержкой для дальнейшего увеличения их эффективности. Это может включать в себя доступ к лучшим инструментам продаж, дополнительное обучение и лучшие условия работы.
- **Распределение рабочей нагрузки:** Убедиться, что рабочая нагрузка равномерно распределена среди всех продавцов, чтобы предотвратить перегрузку лучших исполнителей и дать возможность другим сотрудникам улучшить свои результаты.

5.4. *Исследование динамики продаж.*

```
SELECT
    SUBSTR(order_date, 7, 4) AS year, -- Извлекаем год
    SUBSTR(order_date, 4, 2) AS month, -- Извлекаем месяц
    SUM(oi.quantity * oi.product_price) AS total_sales,
    COUNT(DISTINCT o.order_id) AS total_orders
FROM
    order_items oi
JOIN orders o ON oi.order_id = o.order_id
GROUP BY
    year, month
ORDER BY
    year ASC, month ASC;
```

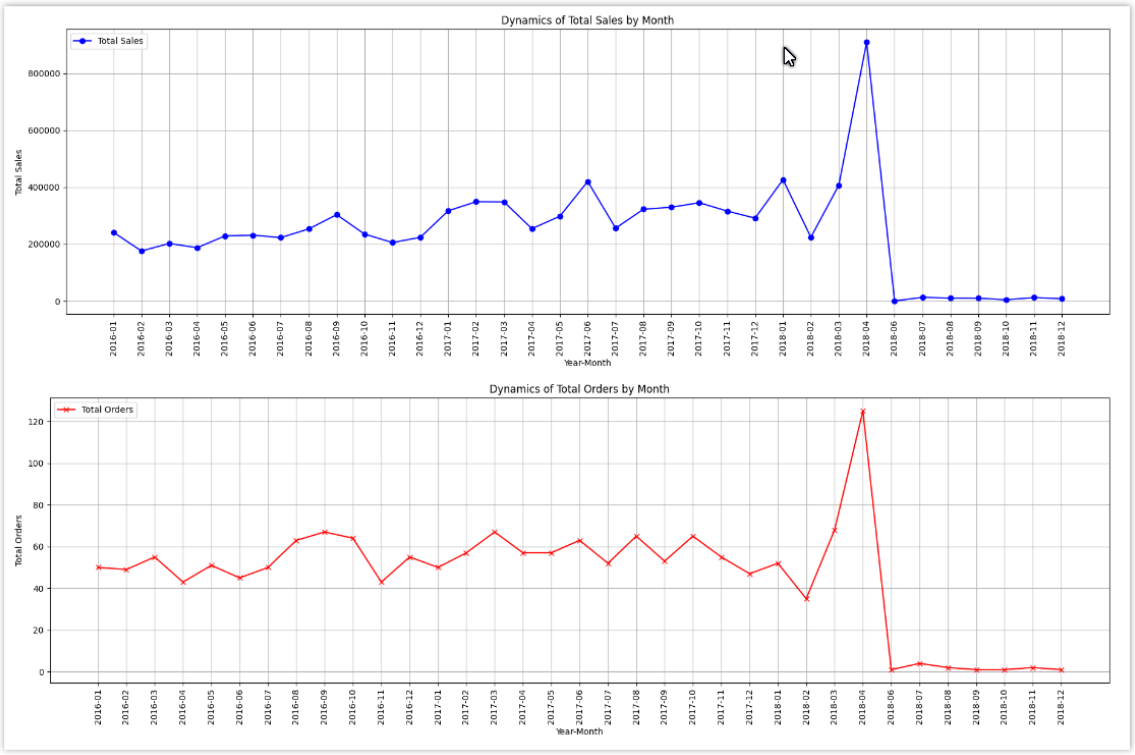
Объяснение запроса:

- **SUBSTR(order_date, 7, 4) AS year**: Извлекает год из строки даты, предполагая, что год находится начиная с седьмого символа и состоит из четырёх символов.
- **SUBSTR(order_date, 4, 2) AS month**: Извлекает месяц из строки даты, предполагая, что месяц находится начиная с четвёртого символа и состоит из двух символов.
- **strftime('%Y', order_date) AS year**: Извлекает год из даты заказа.
- **strftime('%m', order_date) AS month**: Извлекает месяц из даты заказа.
- **SUM(quantity * product_price) AS total_sales**: Вычисляет общую сумму продаж за каждый месяц, умножая количество каждого проданного товара на его цену.
- **COUNT(DISTINCT order_id) AS total_orders**: Считает общее количество уникальных заказов за каждый месяц.
- **GROUP BY year, month**: Группирует результаты по году и месяцу.
- **ORDER BY year, month**: Сортирует результаты по году и месяцу в порядке возрастания.

	year	month	total_sales	total_orders
1	2016	01	241,184.15	50
2	2016	02	175,768.1	49
3	2016	03	202,157.14	55
4	2016	04	187,223.55	43
5	2016	05	228,701.13	51
6	2016	06	231,120.29	45
7	2016	07	222,854.21	50
8	2016	08	253,130.83	63
9	2016	09	303,282.61	67
10	2016	10	235,051.79	64
11	2016	11	205,315.47	43
12	2016	12	223,695.2	55

Визуализация полученных результатов:

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

# Подключение к базе данных
conn = sqlite3.connect('D:/Google Drive/05 - WebBuilding/Learning/MathsHub/02.
SQL/Итоговый проект/BikeDBeaverDatabase/BikeDatabase_2.db')

# Загрузка данных
query = '''
SELECT
    SUBSTR(order_date, 7, 4) AS year, -- Извлекаем год
    SUBSTR(order_date, 4, 2) AS month, -- Извлекаем месяц
    SUM(oi.quantity * oi.product_price) AS total_sales,
    COUNT(DISTINCT o.order_id) AS total_orders
FROM
    order_items oi
JOIN orders o ON oi.order_id = o.order_id
GROUP BY
    year, month
ORDER BY
    year ASC, month ASC;
'''

sales_data = pd.read_sql_query(query, conn)

conn.close()

# Преобразование года и месяца в datetime для удобства визуализации
sales_data['date'] = pd.to_datetime(sales_data['year'] + '-' + sales_data['month'],
format='%Y-%m')

# Визуализация данных
plt.figure(figsize=(14, 7))

# График общих продаж
plt.subplot(1, 2, 1)
plt.plot(sales_data['date'], sales_data['total_sales'], marker='o', linestyle='-', color='b')
plt.title('Динамика общих продаж')
plt.xlabel('Дата')
plt.ylabel('Общие продажи')
plt.xticks(rotation=45)
plt.tight_layout()

# График количества заказов
plt.subplot(1, 2, 2)
plt.plot(sales_data['date'], sales_data['total_orders'], marker='o', linestyle='-', color='r')
plt.title('Динамика количества заказов')
plt.xlabel('Дата')
plt.ylabel('Количество заказов')
plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```



Выводы:

1. **Рост продаж:** Наблюдается значительный рост продаж в апреле 2018 года, что выделяется на фоне остальных периодов. Это может быть связано с успешной маркетинговой кампанией, сезонным спросом или запуском нового продукта.
2. **Стабильность заказов:** Несмотря на колебания объема продаж, количество заказов в целом демонстрирует стабильность, за исключением пика в апреле 2018 года.
3. **Сезонность:** Можно заметить повышение активности в конце каждого года, что может указывать на сезонные пики продаж.
4. **Резкий спад в 2018 году:** После апреля 2018 года наблюдается резкий спад как в продажах, так и в количестве заказов. Это может быть связано с изменениями в ассортименте, ухудшением условий рынка или внутренними проблемами компании.

Рекомендации:

- **Анализ причин роста:** Важно детально изучить причины всплеска продаж в апреле 2018 года для возможного воспроизведения подобного успеха в будущем.
- **Оценка сезонности:** Разработать стратегии для максимизации продаж в периоды традиционного роста спроса.
- **Исследование спада:** Необходимо провести анализ причин резкого спада продаж и заказов после апреля 2018 года и разработать планы по их предотвращению в будущем.
- **Стратегии стабилизации:** Рассмотреть введение программ лояльности или акций для увеличения количества заказов в периоды предполагаемого снижения спроса.

5.5. *Сегментация клиентов:*

5.5.1. по возрасту и социальной активности:

```
SELECT
    c.customer_id,
    c.customer_first_name,
    c.customer_last_name,
    c.customer_age,
    c.social_activity,
    ROUND(SUM(oi.quantity * (oi.product_price - oi.discount)), 2) AS total_spent
FROM
    customers AS c
JOIN orders AS o ON c.customer_id = o.customer_id
JOIN order_items AS oi ON o.order_id = oi.order_id
GROUP BY
    c.customer_id,
    c.customer_age,
    c.social_activity
ORDER BY
    total_spent DESC;
```

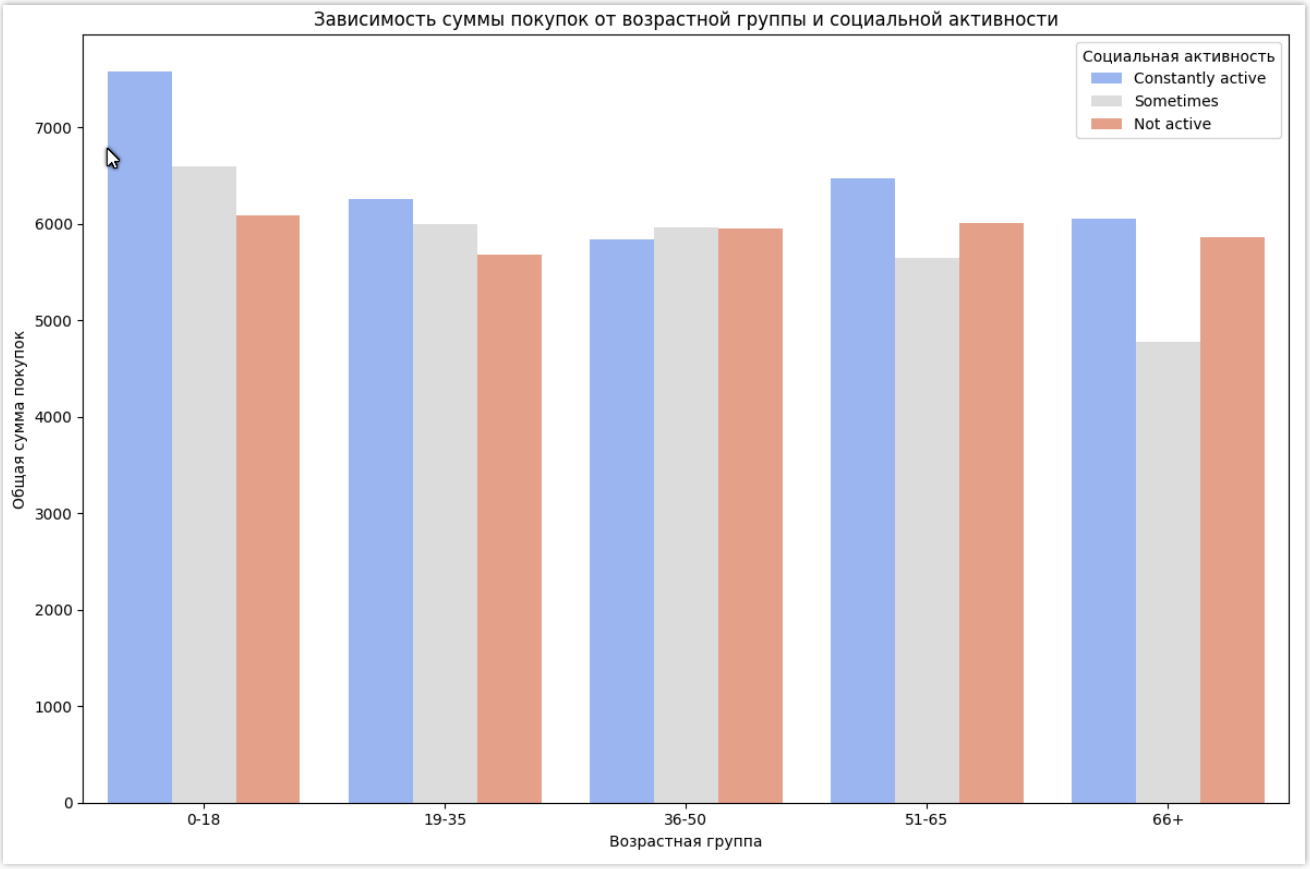
	customer_id	customer_first_name	customer_last_name	customer_age	social_activity	total_spent
1	10	Pamelia	Newman	22	Constantly active	37,799.6
2	75	Abby	Gamble	19	Sometimes	37,499.27
3	94	Sharyn	Hopkins	47	Constantly active	37,137.87
4	6	Lyndsey	Bean	39	Constantly active	35,856.51
5	16	Emmitt	Sanchez	21	Not active	34,502.2
6	73	Melanie	Hayes	34	Constantly active	34,389.86
7	1	Debra	Burks	49	Not active	30,644.23
8	61	Elinore	Aguilar	65	Constantly active	29,659.52
9	93	Corrina	Sawyer	47	Sometimes	29,213.58
10	122	Shena	Carter	39	Sometimes	27,618.51
11	12	Robby	Sykes	56	Sometimes	27,156.94
12	1,224	Abram	Copeland	23	Not active	26,913.31

Объяснение запроса:

- **SELECT**: Выбираем основную информацию о клиенте (ID, имя, фамилия, email).
- **CASE для Income_Segment**: Сегментируем клиентов на основе их месячного дохода.
- **CASE для Social_Activity_Segment**: Сегментируем клиентов по уровню их активности в социальных сетях.
- **CASE для Education_Segment**: Сегментируем клиентов на основе уровня образования.
- **CASE для Family_Status_Segment**: Разделяем клиентов на основе семейного положения.
- **CASE для Children_Segment**: Сегментируем клиентов по количеству детей.

Визуализация полученных результатов:

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

# Подключение к базе данных
conn = sqlite3.connect('D:/Google Drive/05 - WebBuilding/Learning/MathsHub/02.
SQL/Итоговый проект/BikeDBeaverDatabase/BikeDatabase_2.db')

# Загрузка данных
query = '''
SELECT
    c.customer_id,
    c.customer_first_name,
    c.customer_last_name,
    c.customer_age,
    c.social_activity,
    ROUND(SUM(oi.quantity * (oi.product_price - oi.discount)), 2) AS total_spent
FROM
    customers AS c
JOIN orders AS o ON c.customer_id = o.customer_id
JOIN order_items AS oi ON o.order_id = oi.order_id
GROUP BY
    c.customer_id,
    c.customer_age,
    c.social_activity
ORDER BY
    total_spent DESC;
'''

df = pd.read_sql_query(query, conn)

# Закрытие подключения к базе данных
conn.close()

df['age_group'] = pd.cut(df['customer_age'], bins=[0, 18, 35, 50, 65, 100], labels=['0-18',
'19-35', '36-50', '51-65', '66+'])

# Визуализация
plt.figure(figsize=(12, 8))
sns.barplot(data=df, x='age_group', y='total_spent', hue='social_activity',
palette='coolwarm', ci=None)

plt.title('Зависимость суммы покупок от возрастной группы и социальной активности')
plt.xlabel('Возрастная группа')
plt.ylabel('Общая сумма покупок')
plt.legend(title='Социальная активность')

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



Выводы:

1. **Высокий Уровень Дохода и Социальной Активности**: Большинство клиентов относятся к категории с высоким доходом и высокой социальной активностью. Это может указывать на то, что магазин привлекает клиентов с более высоким уровнем благосостояния, которые активно участвуют в социальных сетях.
2. **Образование и Семейное Положение**: Все клиенты были классифицированы как "Менее образованные" и в основном относятся к категории "Одинокие/Другое". Это может потребовать дополнительного изучения для понимания, связано ли это с особенностями предложения магазина или с выборкой данных.
3. **Количество Детей**: Распределение количества детей среди клиентов показывает разнообразие, причем значительная часть клиентов имеет 3 и более детей. Это может указывать на то, что магазин предлагает продукты или услуги, которые привлекательны для семей с детьми.

Рекомендации:

- **Маркетинговая Стратегия**: Учитывая высокий доход и социальную активность клиентов, магазин может использовать стратегии цифрового маркетинга, нацеленные на социальные сети, для привлечения и удержания этой аудитории.
- **Ассортимент Продукции**: Предложение товаров, ориентированных на семьи с детьми, может быть расширено, учитывая значительное количество клиентов с 3 и более детьми.
- **Персонализация Предложений**: Использование данных о семейном положении и количестве детей для создания персонализированных маркетинговых кампаний, предлагающих товары и услуги, соответствующие потребностям различных сегментов клиентов.

5.5.2. По семейному статусу и количеству детей:

```
SELECT
    customer_id,
    customer_first_name,
    customer_last_name,
    customer_family_status,
    customer_children_amount,
    CASE
        WHEN customer_family_status IN ('Married', 'In a relationship') THEN 'In a
Relationship'
        ELSE 'Single/Other'
    END AS Relationship_Status,
    CASE
        WHEN customer_children_amount = 0 THEN 'No Children'
        WHEN customer_children_amount BETWEEN 1 AND 2 THEN '1-2 Children'
        WHEN customer_children_amount > 2 THEN '3+ Children'
    END AS Children_Segment,
    ROUND(SUM(total_spent), 2) AS Total_Spent
FROM (
    SELECT
        c.customer_id,
        c.customer_first_name,
        c.customer_last_name,
        c.customer_family_status,
        c.customer_children_amount,
        oi.quantity * (oi.product_price - oi.discount) AS total_spent
    FROM customers c
    JOIN orders o ON c.customer_id = o.customer_id
    JOIN order_items oi ON o.order_id = oi.order_id
) AS Customer_Spending
GROUP BY
    customer_id,
    customer_first_name,
    customer_last_name,
    customer_family_status,
    customer_children_amount
ORDER BY
    Total_Spent DESC;
```

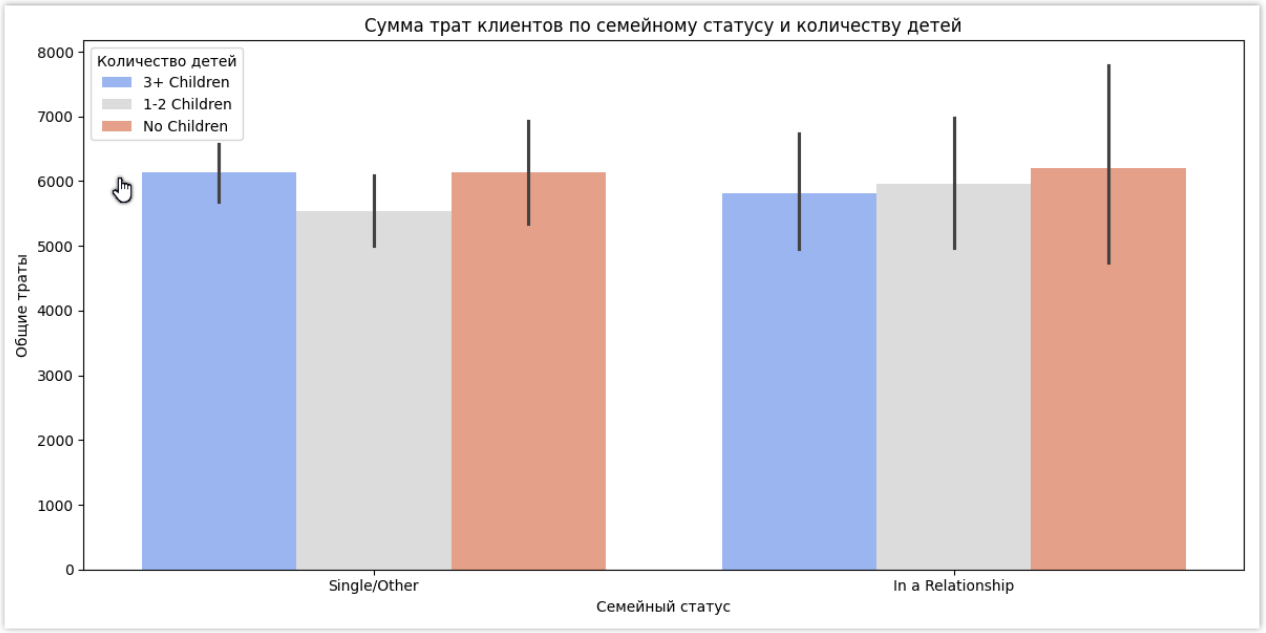
	customer_id	customer_first_name	customer_last_name	customer_family_status	customer_children_amount	Relationship_Status	Children_Segment	Total_Spent
1	10	Pamelia	Newman	Divorced	4	Single/Other	3+ Children	37,799.6
2		Gamble	Married	2	In a Relationship	1-2 Children	37,499.27	
3	94	Sharyn	Hopkins	Married	0	In a Relationship	No Children	37,137.87
4	6	Lyndsey	Bean	Divorced	3	Single/Other	3+ Children	35,856.51
5	16	Emmitt	Sanchez	Married	5	In a Relationship	3+ Children	34,502.2
6	73	Melanie	Hayes	Divorced	5	Single/Other	3+ Children	34,389.86
7	1	Debra	Burks	Divorced	2	Single/Other	1-2 Children	30,644.23
8	61	Elinore	Aguilar	Widow/er	3	Single/Other	3+ Children	29,659.52
9	93	Corrina	Sawyer	Divorced	0	Single/Other	No Children	29,213.58
10	122	Shena	Carter	Married	2	In a Relationship	1-2 Children	27,618.51
11	12	Robby	Sykes	Single	3	Single/Other	3+ Children	27,156.94
12	1,224	Abram	Copeland	Divorced	3	Single/Other	3+ Children	26,913.31
13	9	Genoveva	Baldwin	Married	1	In a Relationship	1-2 Children	26,678.5
14	3	Tameka	Fisher	Divorced	2	Single/Other	1-2 Children	26,248.02
15	66	Lorrie	Becker	Widow/er	4	Single/Other	3+ Children	25,939.55
16	85	Teofila	Fischer	Divorced	0	Single/Other	No Children	25,771.32
17	43	Mozelle	Carter	Married	4	In a Relationship	3+ Children	25,380.94

Объяснение запроса:

Этот запрос группирует клиентов по их семейному статусу и количеству детей, предоставляя информацию о сумме, потраченной каждым клиентом. Внутренний подзапрос рассчитывает общую сумму трат каждого клиента, а внешний запрос использует функции CASE для сегментации клиентов по семейному статусу и количеству детей.

Визуализация полученных результатов:

```
SELECT
    customer_id,
    customer_first_name,
    customer_last_name,
    customer_family_status,
    customer_children_amount,
    CASE
        WHEN customer_family_status IN ('Married', 'In a relationship') THEN 'In a
Relationship'
        ELSE 'Single/Other'
    END AS Relationship_Status,
    CASE
        WHEN customer_children_amount = 0 THEN 'No Children'
        WHEN customer_children_amount BETWEEN 1 AND 2 THEN '1-2 Children'
        WHEN customer_children_amount > 2 THEN '3+ Children'
    END AS Children_Segment,
    ROUND(SUM(total_spent), 2) AS Total_Spent
FROM (
    SELECT
        c.customer_id,
        c.customer_first_name,
        c.customer_last_name,
        c.customer_family_status,
        c.customer_children_amount,
        oi.quantity * (oi.product_price - oi.discount) AS total_spent
    FROM customers c
    JOIN orders o ON c.customer_id = o.customer_id
    JOIN order_items oi ON o.order_id = oi.order_id
) AS Customer_Spending
GROUP BY
    customer_id,
    customer_first_name,
    customer_last_name,
    customer_family_status,
    customer_children_amount
ORDER BY
    Total_Spent DESC;
```



Выводы:

1. **Семейный статус и траты:** Клиенты, находящиеся в отношениях, могут тратить больше по сравнению с одинокими клиентами, особенно если у них есть дети. Это может быть связано с более высокими общими потребностями семьи.
2. **Влияние детей на траты:** Клиенты с детьми, особенно с 1-2 детьми, могут иметь более высокие общие траты по сравнению с клиентами без детей или с 3+ детьми. Это может быть связано с необходимостью покупки товаров для детей и семейных товаров.
3. **Оптимизация предложений:** Магазин может использовать эти данные для оптимизации своих маркетинговых стратегий, например, предлагая специальные скидки для семей с детьми или разрабатывая целевые предложения для одиноких клиентов.

5.5.3. По образованию и месячному доходу:

```
SELECT
    c.degree AS Education,
    CASE
        WHEN c.customer_monthly_income <= 1000 THEN 'Low Income'
        WHEN c.customer_monthly_income > 1000 AND c.customer_monthly_income <=
3000 THEN 'Middle Income'
        WHEN c.customer_monthly_income > 3000 THEN 'High Income'
    END AS Income_Level,
    COUNT(DISTINCT o.order_id) AS Total_Orders,
    ROUND(SUM(oi.quantity * (oi.product_price - oi.discount)),2) AS Total_Spent
FROM
    customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN order_items oi ON o.order_id = oi.order_id
GROUP BY
    Education,
    Income_Level
ORDER BY
    Total_Spent DESC;
```

	Education	Income_Level	Total_Orders	Total_Spent
1	Doctoral degree	High Income	315	1,759,405.94
2	Elementary School	High Income	329	1,753,923.63
3	Master's degree	High Income	315	1,555,089.59
4	Bachelor's degree	High Income	273	1,440,208
5	Elementary School	Middle Income	98	556,208.02
6	Doctoral degree	Middle Income	99	530,996.42
7	Bachelor's degree	Middle Income	91	523,928.85
8	Master's degree	Middle Income	95	458,483.35

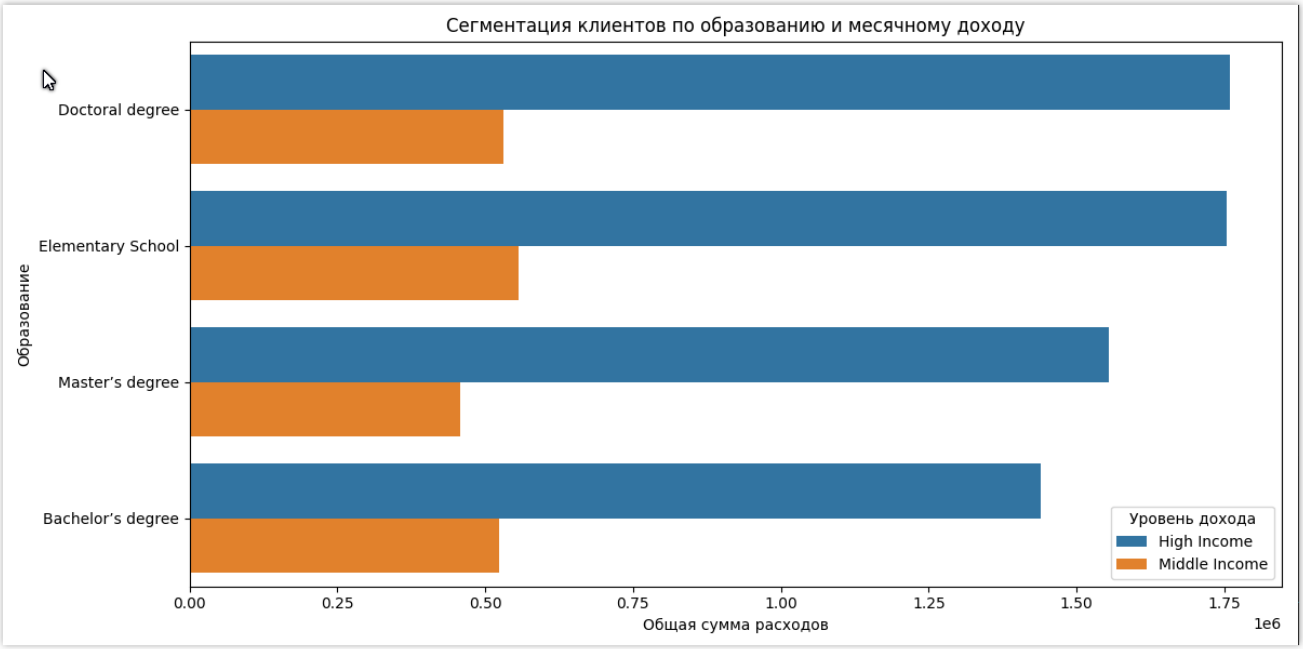
Объяснение запроса:

- Использует **CASE** для категоризации клиентов по диапазонам месячного дохода.
- Группирует результаты по уровню образования (**customer_degree**) и категории дохода (**Income_Level**).
- Считает общее количество заказов (**Total_Orders**) и общую сумму трат (**Total_Spent**) для каждой группы.
- Сортирует результаты по убыванию общей суммы трат, чтобы выявить, какие сегменты клиентов тратят больше всего.

Визуализация полученных результатов:

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

# Подключение к базе данных
conn = sqlite3.connect('D:/Google Drive/05 - WebBuilding/Learning/MathsHub/02.
SQL/Итоговый проект/BikeDBeaverDatabase/BikeDatabase_2.db')

# Загрузка данных
query = """
SELECT
    c.degree AS Education,
    CASE
        WHEN c.customer_monthly_income <= 1000 THEN 'Low Income'
        WHEN c.customer_monthly_income > 1000 AND c.customer_monthly_income <=
3000 THEN 'Middle Income'
        WHEN c.customer_monthly_income > 3000 THEN 'High Income'
    END AS Income_Level,
    COUNT(DISTINCT o.order_id) AS Total_Orders,
    ROUND(SUM(oi.quantity * (oi.product_price - oi.discount)), 2) AS Total_Spent
FROM
    customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN order_items oi ON o.order_id = oi.order_id
GROUP BY
    Education,
    Income_Level
ORDER BY
    Total_Spent DESC;
"""
```

```
df = pd.read_sql_query(query, conn)

# Закрытие подключения к базе данных
conn.close()

# Визуализация данных
plt.figure(figsize=(12, 6))
sns.barplot(data=df, x='Total_Spent', y='Education', hue='Income_Level')
plt.title('Сегментация клиентов по образованию и месячному доходу')
plt.xlabel('Общая сумма расходов')
plt.ylabel('Образование')
plt.legend(title='Уровень дохода')
plt.tight_layout()
plt.show()
```



Выводы:

1. В данном примере невозможно сделать достоверные выводы, так как распределение значений в таблице производилось рандомно. Уровень дохода клиента только со средним школьным образованием не может быть выше бакалавра и магистра, и не сопоставим с доходом доктора наук.
2. **Различия в расходах по образовательным уровням и доходам**: Этот анализ может показать, как образовательный уровень и доход влияют на покупательскую способность и предпочтения клиентов. Например, клиенты с высоким доходом могут тратить больше независимо от их образовательного уровня.
3. **Целевая аудитория для маркетинговых кампаний**: Понимание, какие сегменты тратят больше, может помочь компании более эффективно нацеливать свои маркетинговые усилия и предложения.
4. **Адаптация предложений**: На основе анализа компания может адаптировать свои продукты, услуги и предложения, чтобы лучше удовлетворить потребности различных сегментов клиентов, основываясь на их образовательном уровне и доходе.

5. **Стратегическое планирование**: Данные могут использоваться для стратегического планирования ассортимента продуктов, ценообразования и акций, направленных на максимизацию продаж и удовлетворения потребностей клиентов.

5.6. *Общий объем продаж по магазинам:*

```
CREATE VIEW sales_by_store AS
SELECT
    s.store_id,
    s.store_name,
    SUM(oi.quantity * oi.product_price) AS total_sales
FROM
    stores s
JOIN orders o ON s.store_id = o.store_id
JOIN order_items oi ON o.order_id = oi.order_id
GROUP BY
    s.store_id,
    s.store_name;
```

	123 store_id	ABC store_name	123 total_sales
1	1	Santa Cruz Bikes	1,790,145.91
2	2	Baldwin Bikes	5,826,242.21
3	3	Rowlett Bikes	962,600.76

Объяснение запроса:

1. Выбираем идентификатор магазина (**store_id**), название магазина (**store_name**) из таблицы **stores**.
2. С помощью операторов **JOIN**, запрос объединяем таблицу **stores** с таблицами **orders** и **order_items** по соответствующим идентификаторам магазинов и заказов. Это делается для того, чтобы получить доступ к данным о заказах и позициях в заказах, связанных с каждым магазином.
3. Для каждого магазина суммируем продукт количества проданных товаров (**quantity**) и их цен (**product_price**) из таблицы **order_items**. Это дает общую сумму продаж для каждого магазина.
4. Результаты группируем по **store_id** и **store_name**, что означает, что для каждого уникального магазина будет выведена одна запись с общей суммой продаж.
5. Вместо того, чтобы каждый раз выполнять весь запрос заново, создаём представление **sales_by_store**, которое можно использовать для быстрого доступа к обобщенным данным о продажах по магазинам.

Визуализация полученных результатов:

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

# Подключение к базе данных
conn = sqlite3.connect('D:/Google Drive/05 - WebBuilding/Learning/MathsHub/02. SQL/Итоговый проект/BikeDBeaverDatabase/BikeDatabase_2.db')

# Загрузка данных
query = """
CREATE VIEW sales_by_store AS
SELECT
    s.store_id,
    s.store_name,
    SUM(oi.quantity * oi.product_price) AS total_sales
FROM
    stores s
JOIN orders o ON s.store_id = o.store_id
JOIN order_items oi ON o.order_id = oi.order_id
GROUP BY
    s.store_id,
    s.store_name;
"""

sales_data = pd.read_sql_query(query, conn)

# Закрытие соединения с базой данных
conn.close()

# Визуализация данных
plt.figure(figsize=(10, 6))
plt.bar(sales_data['store_name'], sales_data['total_sales'])
plt.title('Общий объем продаж по магазинам')
plt.xlabel('Магазин')
plt.ylabel('Общий объем продаж')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

Выводы:

1. **Различия в продажах между магазинами**: Этот график показывает, какие магазины имеют наибольший объем продаж, что может указывать на их популярность, местоположение или ассортимент товаров.
2. **Определение лидеров и аутсайдеров**: Возможно определить, какие магазины являются лидерами по продажам, а какие отстают. Это может служить основой для дальнейшего анализа причин такого распределения.
3. **Планирование маркетинговых и операционных усилий**: Анализ объема продаж по магазинам помогает компании определить, где стоит увеличить

маркетинговые усилия, расширить ассортимент или улучшить обслуживание клиентов.

4. **Инвестиции и расширение**: Данные о продажах по магазинам могут быть использованы для принятия решений о дальнейших инвестициях, открытии новых магазинов или, наоборот, о закрытии неэффективных точек продаж.

5.7. *Эффективность персонала*

```
CREATE VIEW staff_efficiency AS
SELECT
    st.staff_id,
    st.staff_first_name,
    st.staff_last_name,
    COUNT(DISTINCT o.order_id) AS total_orders_handled,
    SUM(oi.quantity * oi.product_price) AS total_sales_generated
FROM
    staffs st
JOIN orders o ON st.staff_id = o.staff_id
JOIN order_items oi ON o.order_id = oi.order_id
GROUP BY
    st.staff_id,
    st.staff_first_name,
    st.staff_last_name;
```

	123 staff_id	ABC staff_first_name	ABC staff_last_name	123 total_orders_handled	123 total_sales_generated
1	2	Mireya	Copeland	164	837,423.65
2	3	Genna	Serrano	184	952,722.26
3	6	Marcelene	Boyer	553	2,938,888.73
4	7	Venita	Daniel	540	2,887,353.48
5	8	Kali	Vargas	88	516,695.17
6	9	Layla	Terrell	86	445,905.59

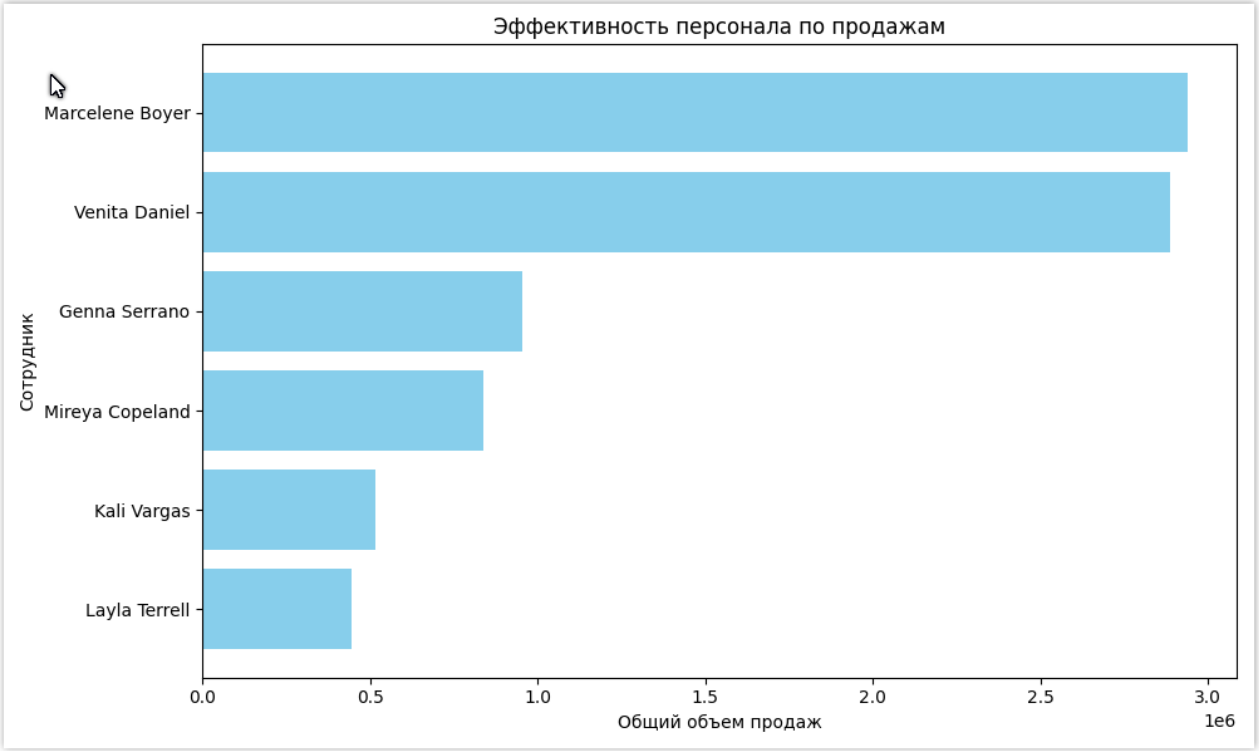
Объяснение запроса:

1. **st.staff_id, st.staff_first_name, st.staff_last_name**: Выбираем идентификатор, имя и фамилию каждого сотрудника из таблицы **staffs**.
2. **COUNT(DISTINCT o.order_id) AS total_orders_handled**: Считаем общее количество уникальных заказов, обработанных каждым сотрудником. Использование **DISTINCT** гарантирует, что заказы считаются один раз даже если в одном заказе было несколько позиций.
3. **SUM(oi.quantity * oi.product_price) AS total_sales_generated**: Вычисляем общую сумму продаж, сгенерированную каждым сотрудником, умножая количество каждого товара на его цену и суммируя результаты по всем заказам, обработанным сотрудником.
4. **FROM staffs st**: Указываем, что основной таблицей для запроса является таблица **staffs**, где хранится информация о сотрудниках.
5. **JOIN orders o ON st.staff_id = o.staff_id**: Соединяем таблицу **staffs** с таблицей **orders**, чтобы получить информацию о заказах, обработанных каждым сотрудником.
6. **JOIN order_items oi ON o.order_id = oi.order_id**: Соединяем таблицу **orders** с таблицей **order_items**, чтобы получить детали по каждому товару в заказах.

7. **GROUP BY st.staff_id, st.staff_first_name, st.staff_last_name**: Группируем результаты запроса по идентификатору, имени и фамилии сотрудника, чтобы агрегатные функции **COUNT** и **SUM** применялись к данным каждого сотрудника отдельно.

Визуализация полученных результатов:

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

# Подключение к базе данных
conn = sqlite3.connect('D:/Google Drive/05 - WebBuilding/Learning/MathsHub/02. SQL/Итоговый проект/BikeDBeaverDatabase/BikeDatabase_2.db')

# Загрузка данных
query = '''
SELECT
    st.staff_id,
    st.staff_first_name || ' ' || st.staff_last_name AS staff_name,
    COUNT(DISTINCT o.order_id) AS total_orders_handled,
    SUM(oi.quantity * oi.product_price) AS total_sales_generated
FROM
    staffs st
JOIN orders o ON st.staff_id = o.staff_id
JOIN order_items oi ON o.order_id = oi.order_id
GROUP BY
    st.staff_id,
    st.staff_first_name,
    st.staff_last_name;
'''
staff_efficiency_data = pd.read_sql_query(query, conn)

# Закрытие соединения с базой данных
conn.close()

# Визуализация данных
plt.figure(figsize=(10, 6))
staff_efficiency_data.sort_values('total_sales_generated', ascending=True, inplace=True)
plt.barh(staff_efficiency_data['staff_name'], staff_efficiency_data['total_sales_generated'], color='skyblue')
plt.xlabel('Общий объем продаж')
plt.ylabel('Сотрудник')
plt.title('Эффективность персонала по продажам')
plt.tight_layout()
plt.show()
```



Выводы:

- Столбчатая диаграмма дает наглядное представление о том, как сотрудники компании работают в плане продаж. Мы можем видеть, кто из сотрудников генерирует наибольший объем продаж, что может быть полезно для оценки их работы и внесения коррективов в процесс обучения и мотивации.
- Сотрудники с наибольшим объемом продаж могут быть награждены или использованы в качестве примера для других сотрудников.
- Анализ также может выявить потенциальные проблемы в работе отдельных сотрудников или групп, что позволит руководству предпринять соответствующие меры для улучшения общей эффективности команды.

5.8. *Расчитываем куммулятивный профит по месяцам*

```
WITH MonthlySales AS (
    SELECT
        SUBSTR(o.order_date, 4, 2) || '/' || SUBSTR(o.order_date, 7, 4) AS sale_month, --
Преобразование в MM/YYYY
        SUM(oi.quantity * oi.product_price) AS total_sales
    FROM
        orders o
    JOIN order_items oi ON o.order_id = oi.order_id
    GROUP BY
        SUBSTR(o.order_date, 4, 2) || '/' || SUBSTR(o.order_date, 7, 4)
),
CumulativeProfit AS (
    SELECT
        sale_month,
        total_sales,
        SUM(total_sales) OVER (ORDER BY sale_month) AS cumulative_profit
    FROM
```

```
    MonthlySales
)
SELECT
    sale_month,
    ROUND(cumulative_profit, 2) AS cumulative_profit
FROM
    CumulativeProfit
ORDER BY
    sale_month;
```

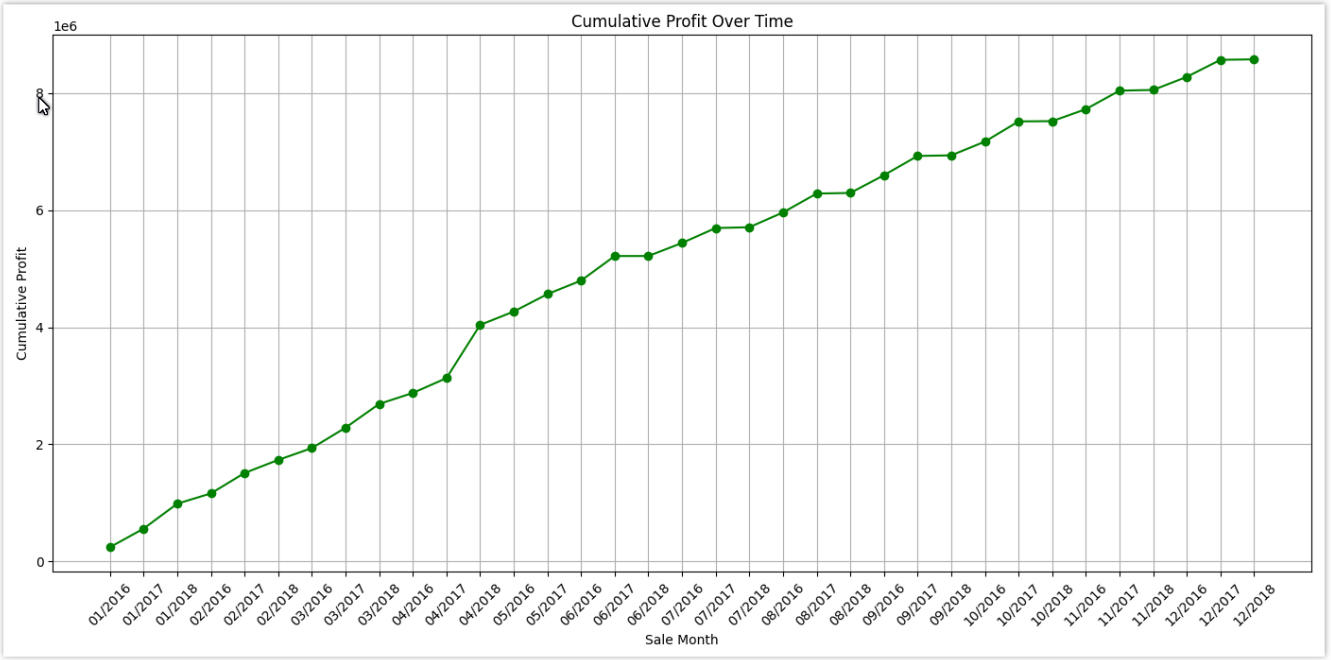
	sale_month	cumulative_profit
1	01/2016	241,184.15
2	01/2017	558,138.92
3	01/2018	984,440.64
4	02/2016	1,160,208.74
5	02/2017	1,508,949.21
6	02/2018	1,732,890.65
7	03/2016	1,935,047.79
8	03/2017	2,283,224.92
9	03/2018	2,689,926.12
10	04/2016	2,877,149.67
11	04/2017	3,131,255.24

Объяснение запроса:

1. **Определение месячных продаж (MonthlySales):**
   - Во внутреннем запросе создается временная таблица MonthlySales, где каждая запись представляет собой отдельный месяц (в формате MM/YYYY, полученном из order_date).
   - Для каждого месяца рассчитывается total_sales, суммируя продукт количества товаров (quantity) и их цены (product_price), учитывая каждую покупку в заказах.
2. **Расчет кумулятивного профита (CumulativeProfit):**
   - В следующем подзапросе CumulativeProfit рассчитывается кумулятивная сумма (cumulative_profit) для каждого месяца. Для этого используется оконная функция SUM(...) OVER (ORDER BY sale_month), которая складывает total_sales начиная с первого месяца и накапливая значения в порядке следования месяцев.
3. **Выборка и округление результатов:**
   - Во внешнем запросе выбираются месяц (sale_month) и кумулятивный профит (cumulative_profit), где кумулятивный профит округляется до двух десятичных знаков для более четкого представления.
4. **Сортировка данных:**
   - Результаты упорядочиваются по месяцам в возрастающем порядке, что обеспечивает последовательное представление кумулятивного профита от более ранних к более поздним периодам.

Визуализация полученных результатов:

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

# Подключение к базе данных
conn = sqlite3.connect('D:/Google Drive/05 - WebBuilding/Learning/MathsHub/02. SQL/Итоговый проект/BikeDBeaverDatabase/BikeDatabase_2.db')

# Загрузка данных
query = """
WITH MonthlySales AS (
    SELECT
        SUBSTR(o.order_date, 4, 2) || '/' || SUBSTR(o.order_date, 7, 4) AS sale_month,
        SUM(oi.quantity * oi.product_price) AS total_sales
    FROM
        orders o
    JOIN order_items oi ON o.order_id = oi.order_id
    GROUP BY
        SUBSTR(o.order_date, 4, 2) || '/' || SUBSTR(o.order_date, 7, 4)
),
CumulativeProfit AS (
    SELECT
        sale_month,
        total_sales,
        SUM(total_sales) OVER (ORDER BY sale_month) AS cumulative_profit
    FROM
        MonthlySales
)
SELECT
    sale_month,
    ROUND(cumulative_profit, 2) AS cumulative_profit
FROM
    CumulativeProfit
ORDER BY
    sale_month;
"""
df = pd.read_sql_query(query, conn)

# Закрытие соединения с базой данных
conn.close()

# Визуализация данных
plt.figure(figsize=(14, 7))
plt.plot(df['sale_month'], df['cumulative_profit'], marker='o', linestyle='-', color='green')
plt.title('Cumulative Profit Over Time')
plt.xlabel('Sale Month')
plt.ylabel('Cumulative Profit')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```



Выводы:

- **Нарастающий итог профита**: График показывает, как профит накапливается с течением времени. Это дает представление о росте или устойчивости доходности компании в разные месяцы.
- **Выявление трендов**: Можно увидеть определенные тренды или паттерны, например, увеличение прибыли в определенные сезоны или месяцы, что может указывать на сезонность бизнеса.
- **Планирование бюджета**: Анализ кумулятивного профита может помочь в планировании бюджета и финансовом прогнозировании на будущие периоды.
- **Оценка эффективности маркетинговых и продажных стратегий**: Понимание, как изменяется кумулятивный профит, может указывать на эффективность проведенных маркетинговых кампаний или изменений в стратегии продаж.

5.9. *Рассчитываем скользящее среднее профита*

```
WITH MonthlyProfit AS (
    SELECT
        SUBSTR(order_date, 7, 4) || '-' || SUBSTR(order_date, 4, 2) AS sale_month, --
Преобразование dd/mm/yyyy к yyyy-mm
        SUM(oi.quantity * oi.product_price - oi.discount) AS total_profit
    FROM
        orders o
        JOIN order_items oi ON o.order_id = oi.order_id
    GROUP BY
        sale_month
),
RollingAverageProfit AS (
    SELECT
        sale_month,
        total_profit,
        AVG(total_profit) OVER (
            ORDER BY sale_month
```

```
            ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
        ) AS rolling_avg_profit -- Скользящее среднее за 3 месяца
    FROM
        MonthlyProfit
)
SELECT
    sale_month,
    ROUND(rolling_avg_profit, 2) AS rolling_avg_profit -- Округление до сотых
FROM
    RollingAverageProfit
ORDER BY
    sale_month;
```

	sale_month	rolling_avg_profit
1	2016-01	241,168.89
2	2016-02	208,460.21
3	2016-03	206,354.32
4	2016-04	188,368.39
5	2016-05	206,013.35
6	2016-06	215,668.29
7	2016-07	227,544.25
8	2016-08	235,686.72
9	2016-09	259,738.52
10	2016-10	263,803.7
11	2016-11	247,866.41
12	2016-12	221,338.86
13	2017-01	248,640.31

Объяснение запроса:

1. Подзапрос **MonthlyProfit**

- **SUBSTR(order_date, 7, 4) || '-' || SUBSTR(order_date, 4, 2) AS sale_month**: Преобразует дату из формата **dd/mm/yyyy** в **yyyy-mm**, чтобы группировать данные по месяцам. **SUBSTR** извлекаем подстроки из **order_date** (год и месяц) и объединяем их с помощью оператора конкатенации **||**.
- **SUM(oi.quantity * oi.product_price - oi.discount) AS total_profit**: Здесь рассчитываем общий профит за каждый месяц как сумма продаж (**quantity * product_price**) минус скидки (**discount**) по всем заказам за месяц.
- **GROUP BY sale_month**: Данные группируем по месяцам, чтобы получить суммарный профит за каждый месяц.

2. Подзапрос **RollingAverageProfit**

- **AVG(total_profit) OVER (ORDER BY sale_month ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS rolling_avg_profit**: Оконная функция **AVG** используется для расчета скользящего среднего значения профита за 3 месяца (текущий месяц и два предыдущих). Это достигается за счет указания диапазона строк с помощью **ROWS BETWEEN 2 PRECEDING AND CURRENT ROW**.

3. Финальный **SELECT**

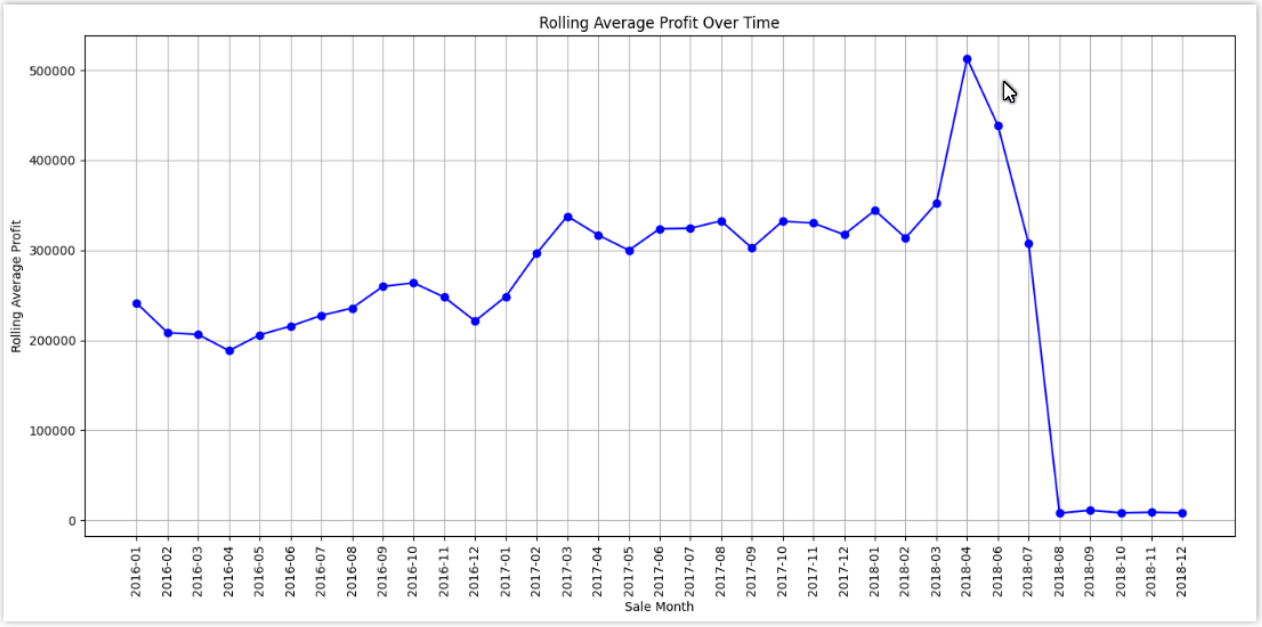
- **ROUND(rolling_avg_profit, 2) AS rolling_avg_profit**: Результат скользящего среднего округляем до двух десятичных знаков для каждого месяца.

- **ORDER BY sale_month**: Результаты упорядочиваем по месяцам, что позволяет наглядно увидеть изменение профита во времени.

Визуализация полученных результатов:

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

# Подключение к базе данных
conn = sqlite3.connect('D:/Google Drive/05 - WebBuilding/Learning/MathsHub/02. SQL/Итоговый проект/BikeDBeaverDatabase/BikeDatabase_2.db')

# Загрузка данных
query = """
WITH MonthlyProfit AS (
    SELECT
        SUBSTR(order_date, 7, 4) || '-' || SUBSTR(order_date, 4, 2) AS sale_month,
        SUM(oi.quantity * oi.product_price - oi.discount) AS total_profit
    FROM
        orders o
        JOIN order_items oi ON o.order_id = oi.order_id
    GROUP BY
        sale_month
),
RollingAverageProfit AS (
    SELECT
        sale_month,
        total_profit,
        AVG(total_profit) OVER (
            ORDER BY sale_month
            ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
        ) AS rolling_avg_profit
    FROM
        MonthlyProfit
)
SELECT
    sale_month,
    ROUND(rolling_avg_profit, 2) AS rolling_avg_profit
FROM
    RollingAverageProfit
ORDER BY
    sale_month;
"""
df = pd.read_sql_query(query, conn)

# Закрытие соединения с базой данных
conn.close()

# Визуализация данных
plt.figure(figsize=(14, 7))
plt.plot(df['sale_month'], df['rolling_avg_profit'], marker='o', linestyle='-', color='blue')
plt.title('Rolling Average Profit Over Time')
```

```
plt.xlabel('Sale Month')
plt.ylabel('Rolling Average Profit')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```



Выводы:

1. **Восходящий тренд:** С начала 2016 до середины 2018 года виден общий восходящий тренд в скользящем среднем профите. Это указывает на то, что компания в целом увеличивала свою прибыльность в течение этого периода.
2. **Резкий рост в апреле 2018**: В апреле 2018 года наблюдается значительный скачок профита. Это может быть связано с каким-то одноразовым событием, например, с крупной продажей, выходом нового продукта или сезонным увеличением спроса.
3. **Падение в июне-декабре 2018:** После апреля 2018 года происходит резкое падение скользящего среднего профита, достигая минимальных значений к концу года. Это могло быть вызвано различными факторами, включая снижение продаж, увеличение затрат или другие рыночные изменения.
4. **Необходимость анализа апреля 2018:** Важно детально изучить, что привело к резкому увеличению профита в апреле 2018, чтобы понять, можно ли воспроизвести эти условия в будущем.
5. **Исследование причин падения профита:** Также критически важно анализировать причины снижения профита в последние месяцы 2018 года, чтобы разработать стратегии по их устранению или минимизации влияния подобных ситуаций в будущем.
6. **Стратегическое планирование:** На основе анализа данных компании следует планировать стратегии нацеленные на поддержание роста профита и предотвращение будущих падений.

5.10. ***Прогнозирование продаж.***

```
WITH SalesAmount AS (
  SELECT
    o.order_date,
    SUM(oi.quantity * oi.product_price) AS total_sales
  FROM orders o
  JOIN order_items oi ON o.order_id = oi.order_id
  GROUP BY o.order_date
)
SELECT
  order_date,
  total_sales,
  ROUND(
    AVG(total_sales) OVER (
      ORDER BY order_date
      ROWS BETWEEN 6 PRECEDING AND CURRENT ROW
    ), 2
  ) AS rolling_avg_7days
FROM SalesAmount
ORDER BY order_date;
```

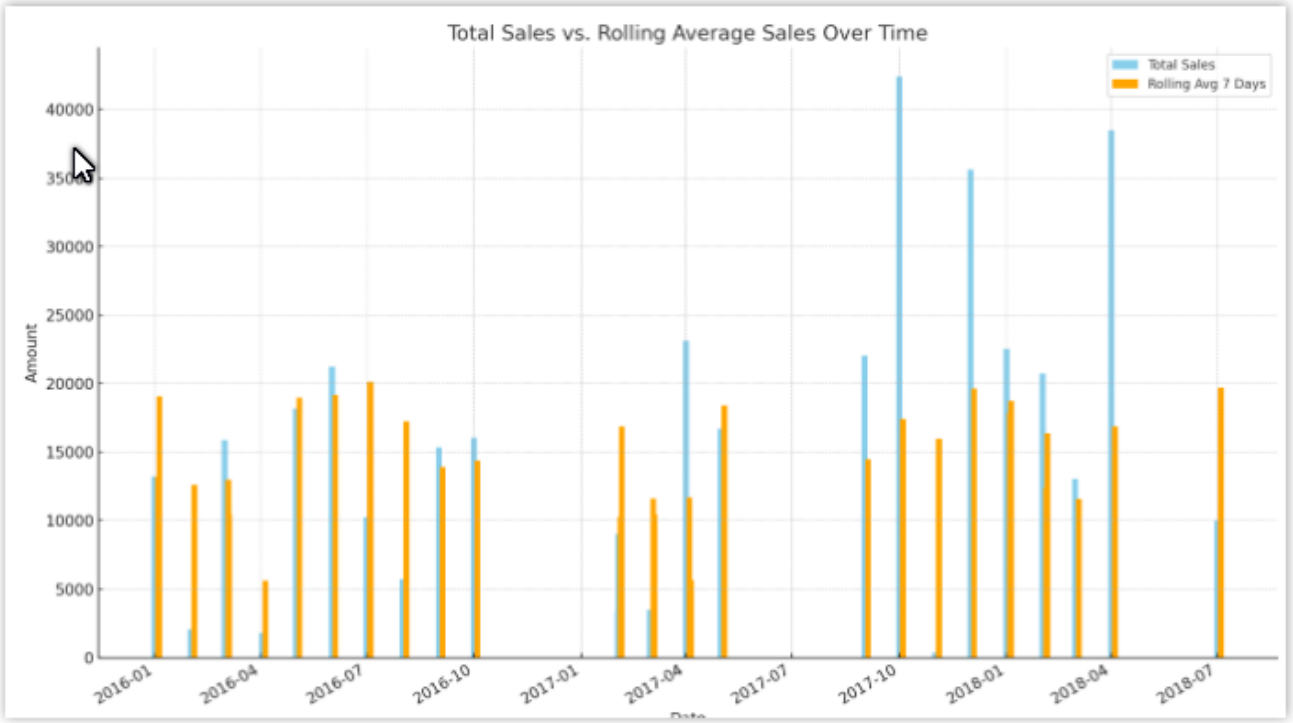
	order_date	total_sales	rolling_avg_7days
1	01/01/2016	13,197.91	13,197.91
2	01/01/2018	22,550.87	17,874.39
3	01/02/2016	2,069.96	12,606.25
4	01/02/2017	3,359.95	10,294.67
5	01/02/2018	20,740.93	12,383.92
6	01/03/2016	15,861.88	12,963.58
7	01/03/2017	3,489.92	11,610.2
8	01/03/2018	13,033.9	11,586.77
9	01/04/2017	23,148.77	11,672.19
10	01/04/2018	38,469.8	16,872.16
11	01/05/2016	18,165.81	18,987.29
12	01/05/2017	16,713.89	18,412
13	01/06/2016	21,238.88	19,180.14

Объяснение запроса:

- Сначала создается общая сумма продаж (**total_sales**) по дням с использованием подзапроса или CTE (**WITH**-конструкция), где данные группируются по дате заказа **order_date**.
- Затем, используя оконную функцию **AVG() OVER (...)**, рассчитывается скользящее среднее за 7 дней (**ROWS BETWEEN 6 PRECEDING AND CURRENT ROW**), что позволяет получить среднее значение общей суммы продаж за текущий и предыдущие 6 дней для каждой даты.

Визуализация полученных результатов:

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

# Подключение к базе данных
conn = sqlite3.connect('D:/Google Drive/05 - WebBuilding/Learning/MathsHub/02.
SQL/Итоговый проект/BikeDBeaverDatabase/BikeDatabase_2.db')

# Загрузка данных
query = """
WITH SalesAmount AS (
  SELECT
    o.order_date,
    SUM(oi.quantity * oi.product_price) AS total_sales
  FROM orders o
  JOIN order_items oi ON o.order_id = oi.order_id
  GROUP BY o.order_date
)
SELECT
  order_date,
  total_sales,
  ROUND(
    AVG(total_sales) OVER (
      ORDER BY order_date
      ROWS BETWEEN 6 PRECEDING AND CURRENT ROW
    ), 2
  ) AS rolling_avg_7days
FROM SalesAmount
ORDER BY order_date;
"""
df = pd.read_sql_query(query, conn)

# Закрытие соединения с базой данных
conn.close()

# Визуализация данных
plt.figure(figsize=(14, 7))
plt.plot(df['order_date'], df['total_sales'], label='Total Sales', color='blue')
plt.plot(df['order_date'], df['rolling_avg_7days'], label='Rolling Average 7 Days', color='red',
linestyle='--')
plt.title('Sales Forecasting with Rolling Average')
plt.xlabel('Order Date')
plt.ylabel('Total Sales')
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()
plt.show()
```



Выводы:

- **Общие продажи** показывают значительные колебания, что может быть связано с сезонностью, маркетинговыми акциями или изменением потребительского спроса.
- **Скользящее среднее** помогает идентифицировать общие тренды, минимизируя влияние краткосрочных колебаний. Это полезно для планирования и принятия стратегических решений.
- Наблюдаются периоды, когда скользящее среднее значительно отличается от общих продаж, что может указывать на необычные события или изменения в рыночной ситуации.
- **Оптимизация запасов** и планирование продаж должны учитывать эти анализы, чтобы лучше соответствовать потребительскому спросу и минимизировать издержки.

5.11. *Анализ зависимости между скидками и объемом продаж.*

```
SELECT
    c.category_name,
    COUNT(DISTINCT o.order_id) AS total_orders,
    SUM(oi.quantity) AS total_quantity_sold,
    AVG(oi.discount) AS average_discount,
    SUM(oi.quantity * (oi.product_price - oi.discount)) AS total_sales_value
FROM
    order_items oi
    JOIN orders o ON oi.order_id = o.order_id
    JOIN products p ON oi.product_id = p.product_id
    JOIN categories c ON p.category_id = c.category_id
GROUP BY
    c.category_name
ORDER BY
    total_sales_value DESC;
```

	category_name	total_orders	total_quantity_sold	average_discount	total_sales_value
1	Mountain Bikes	866	1,755	0.11	3,030,590.9
2	Road Bikes	315	559	0.11	1,852,497.33
3	Cruisers Bicycles	959	2,063	0.1	1,108,936.83
4	Electric Bikes	202	315	0.11	1,020,203.85
5	Cyclocross Bicycles	245	394	0.11	799,831.44
6	Comfort Bicycles	472	813	0.1	438,424.06
7	Children Bicycles	635	1,179	0.11	327,759.39

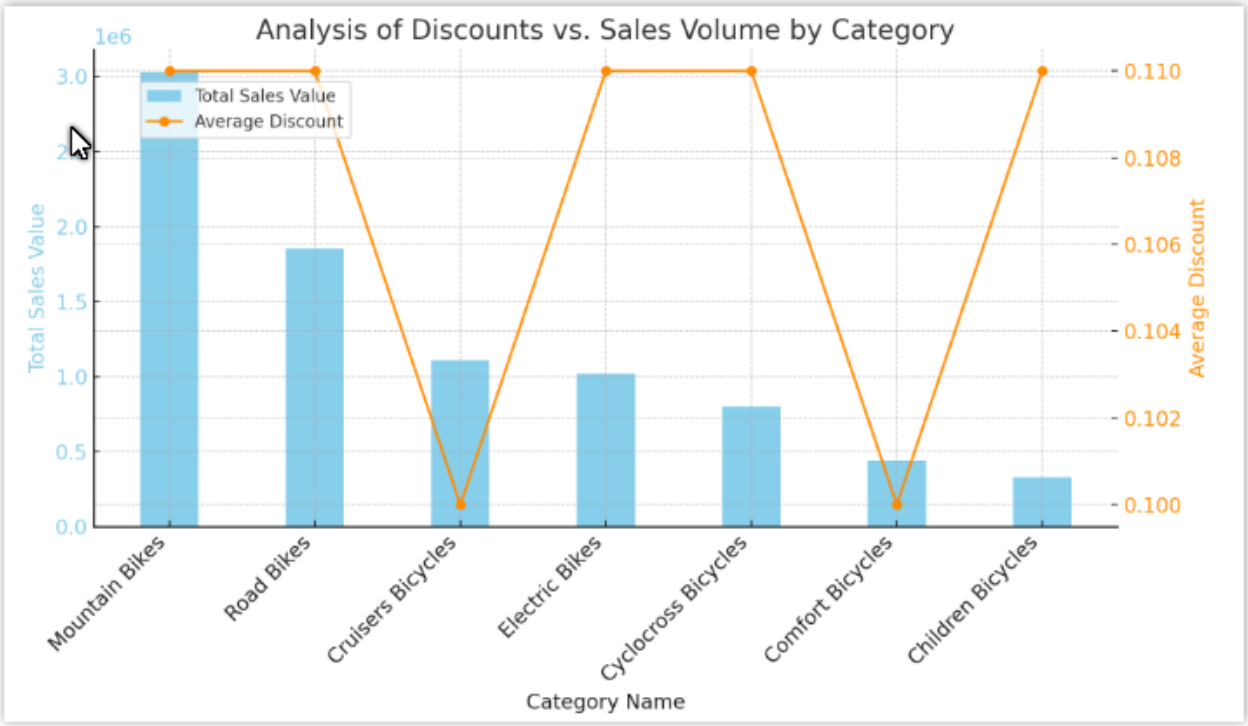
Объяснение запроса:

- **SELECT:**
  - **c.category_name**: Выбираем наименование категории из таблицы **categories**.
  - **COUNT(DISTINCT o.order_id) AS total_orders**: Считаем уникальные идентификаторы заказов из таблицы **orders**, чтобы узнать общее количество заказов по каждой категории.
  - **SUM(oi.quantity) AS total_quantity_sold**: Суммируем количество проданных товаров по каждой категории из таблицы **order_items**.
  - **AVG(oi.discount) AS average_discount**: Вычисляем среднюю скидку на товары в каждой категории.
  - **SUM(oi.quantity * (oi.product_price - oi.discount)) AS total_sales_value**: Считаем общую сумму продаж после применения скидок, умножая количество на цену товара с учетом скидки для каждой категории.
  - **FROM и JOIN**: Используются для соединения четырех таблиц: **order_items** (oi), **orders** (o), **products** (p), и **categories** (c). Соединение таблиц необходимо для того, чтобы собрать данные о заказах, продуктах и категориях товаров в одном запросе.
  - **GROUP BY c.category_name**: Группируем результаты по названию категории, чтобы агрегированные функции (**COUNT**, **SUM**, **AVG**) применялись к каждой категории отдельно. Это позволяет получить агрегированные данные (количество заказов, проданных товаров, среднюю скидку, общую сумму продаж) для каждой категории.
  - **ORDER BY total_sales_value DESC**: Сортируем результаты по убыванию общей суммы продаж (**total_sales_value**), чтобы категории с наибольшим объемом продаж были вверху списка.

Визуализация полученных результатов:

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

# Подключение к базе данных
conn = sqlite3.connect('D:/Google Drive/05 - WebBuilding/Learning/MathsHub/02. SQL/Итоговый проект/BikeDBeaverDatabase/BikeDatabase_2.db')

# Загрузка данных
query = """
SELECT
    c.category_name,
    COUNT(DISTINCT o.order_id) AS total_orders,
    SUM(oi.quantity) AS total_quantity_sold,
    AVG(oi.discount) AS average_discount,
    ROUND(SUM(oi.quantity * (oi.product_price - oi.discount)), 2) AS total_sales_value
```

```
FROM
    order_items oi
    JOIN orders o ON oi.order_id = o.order_id
    JOIN products p ON oi.product_id = p.product_id
    JOIN categories c ON p.category_id = c.category_id
GROUP BY
    c.category_name
ORDER BY
    total_sales_value DESC;
"""
data = pd.read_sql_query(query, conn)

# Закрытие соединения с базой данных
conn.close()

# Визуализация данных
plt.figure(figsize=(12, 8))
sns.barplot(x='total_sales_value', y='category_name', data=data, palette='viridis')
plt.title('Анализ зависимости между скидками и объемом продаж по категориям')
plt.xlabel('Общая стоимость продаж')
plt.ylabel('Категория товара')
plt.show()
```



Выводы:

- **Горные велосипеды (Mountain Bikes)** и **Дорожные велосипеды (Road Bikes)** имеют наибольший объем продаж среди всех категорий, что подчеркивает их популярность. Средняя скидка в этих категориях составляет 0.11, что свидетельствует о стандартной практике скидок для этих товаров.
- **Круизеры (Cruisers Bicycles)**, несмотря на сравнительно низкую среднюю скидку (0.1), также демонстрируют высокий объем продаж. Это может указывать на внутреннее качество или популярность данных товаров среди потребителей.
- **Электровелосипеды (Electric Bikes)** и **Велосипеды для циклокросса (Cyclocross Bicycles)** с средней скидкой 0.11 также показывают значительный

объем продаж, что может свидетельствовать о растущем интересе к этим видам велосипедов.
- Категории **Комфортные велосипеды (Comfort Bicycles)** и **Детские велосипеды (Children Bicycles)** имеют меньший объем продаж по сравнению с другими категориями, что может быть связано с более узкой целевой аудиторией или конкретными потребностями покупателей.
- Скидки имеют небольшое влияние на объем продаж по различным категориям, средняя скидка в большинстве категорий составляет около 0.1-0.11. Это указывает на то, что скидки, возможно, не являются основным драйвером продаж для данных категорий.
- Потребительская популярность и специфика товара играют ключевую роль в объеме продаж по категориям. Важно учитывать эти факторы при планировании маркетинговых кампаний и скидочных акций.

Рекомендации:

- Для категорий с высоким объемом продаж и стандартной скидкой рассмотреть возможность экспериментов с увеличением скидок для стимулирования дополнительных продаж.
- В категориях с меньшим объемом продаж исследовать другие маркетинговые стратегии, помимо скидок, для привлечения внимания к этим товарам.