First:

Before delving into explanations of the solutions and approaches to solving two problems, I'd like to express my keen interest in this position, especially within the company. I want to refer to a dialogue from the film "Before Sunset" (2004), where Jesse and Celine are seated in a café having a conversation. Here's the dialogue:

Celine: "Well, for example, I was working for this organization that helped villages in Mexico. And their concern was how to get the pencils sent to the kids in these little country schools. It was not about big revolutionary ideas, it was about pencils."

And honestly, I believe that anyone, whether it's about a pencil or preventing food waste, can contribute to making the world a better place. This is what truly makes me happy. It's something that ResQ is currently doing, and I would love to see it become active in many countries someday.

It's rare for someone to find a job they love, a company they admire, and the company's goal (which involves contributing to a better world) all aligning in one moment and within one position.

In fact, this is what makes me very interested in this position and your company. A friend once told me to work where my soul pours itself out (due to passion), and I believe ResQ is that place for me. Thank you for giving me this opportunity.

Matt,

Introduction:

I considered two approaches to solve the problems. First, I implemented what was requested in the questions using Python code, which has been uploaded to the repository. Furthermore, I applied the same solutions but in the form of SQL code on a database.

First Problem:

To create a table that a data analyst can use to respond to the given questions, there are several approaches.
The first approach involves creating a view table using the following query. But Why a view?

- **Data Consistency**: A view always displays the latest data present in the original tables. Any changes in the original tables will be automatically reflected in the view, ensuring data consistency.
- **Simplicity**: Views simplify query complexity. You can create a view with complex joins, aggregations, and calculations, then use it like a regular table, making SQL code more readable and maintainable.
- **Performance**: Although views don't inherently improve performance, they can enhance query performance when used with techniques like indexing.

A view doesn't store data; it's a saved query operating on underlying tables. Hence, it doesn't consume additional storage space like a new table. However, since a view's data isn't stored, it recalculates each time it's accessed, potentially impacting performance based on the view's complexity.

For these reasons, using a view is preferred over creating a separate table in this question. The query to create this view is:

```
CREATE view presentation AS
SELECT orders.*, strftime('%Y-%m', createdAt) AS month, strftime('%Y',
users.registeredDate) AS registeredYear, providers.defaultOfferType
FROM orders
INNER JOIN users ON orders.userId = users.id
INNER JOIN providers ON orders.providerId = providers.id;
```

This code is placed in the file **viewQuery.txt**.

The second solution:

involves using the connection available in SQLite through Python to prepare the data in the files **problemOne.ipynb** and **problemOne.py**, eventually storing it in a .db file like **orders_regDate_offer.db**.

The third solution:

Applying the same SQL code placed in **viewQuery.txt** using the SQLite connection in Python on the database and creating a view, similar to the files **problemOneSql.ipynb** and **problemOneSql.py**.
It's worth mentioning that to automate these tasks, one can define cron jobs to execute Python files at specific times.

Second Problem:

First Solution:

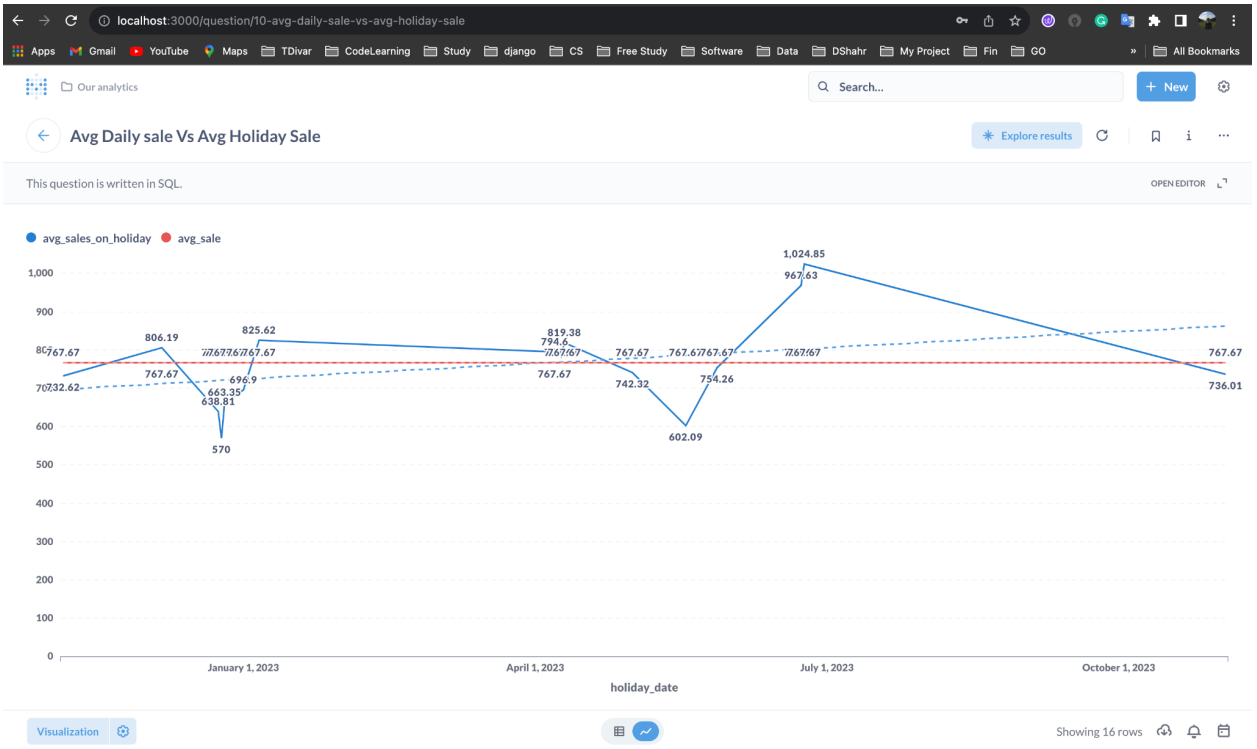
To create a dashboard for the sales team, I utilized Metabase. You can run Metabase either through Docker or using the jar file. I have used Jar file to run Metabase locally on my laptop. After running Metabase, I connected it to the database.

Once Metabase was linked to the database, for this question:
"Do the sales on public holidays differ from the average daily sales?"
The following query was executed in the "new -> sqlquery" section:

```
WITH daily_sales AS (
    SELECT
        DATE(createdAt) AS order_date,
        AVG(sales) AS avg_sales
    FROM
        orders
    WHERE
        DATE(createdAt) NOT IN (
            SELECT DATE(holiday_date) FROM holiday
        )
    GROUP BY
        DATE(createdAt)
),
avg_daily_sales AS (
    SELECT AVG(avg_sales) AS avg_sale
    FROM daily_sales
)
SELECT
    holiday.holiday_date,
    AVG(orders.sales) AS avg_sales_on_holiday,
    avg_sale
FROM
    holiday
```

```
JOIN
    orders ON DATE(orders.createdAt) = DATE(holiday.holiday_date)
CROSS JOIN
    avg_daily_sales
GROUP BY
    holiday.holiday_date, avg_sale;
```

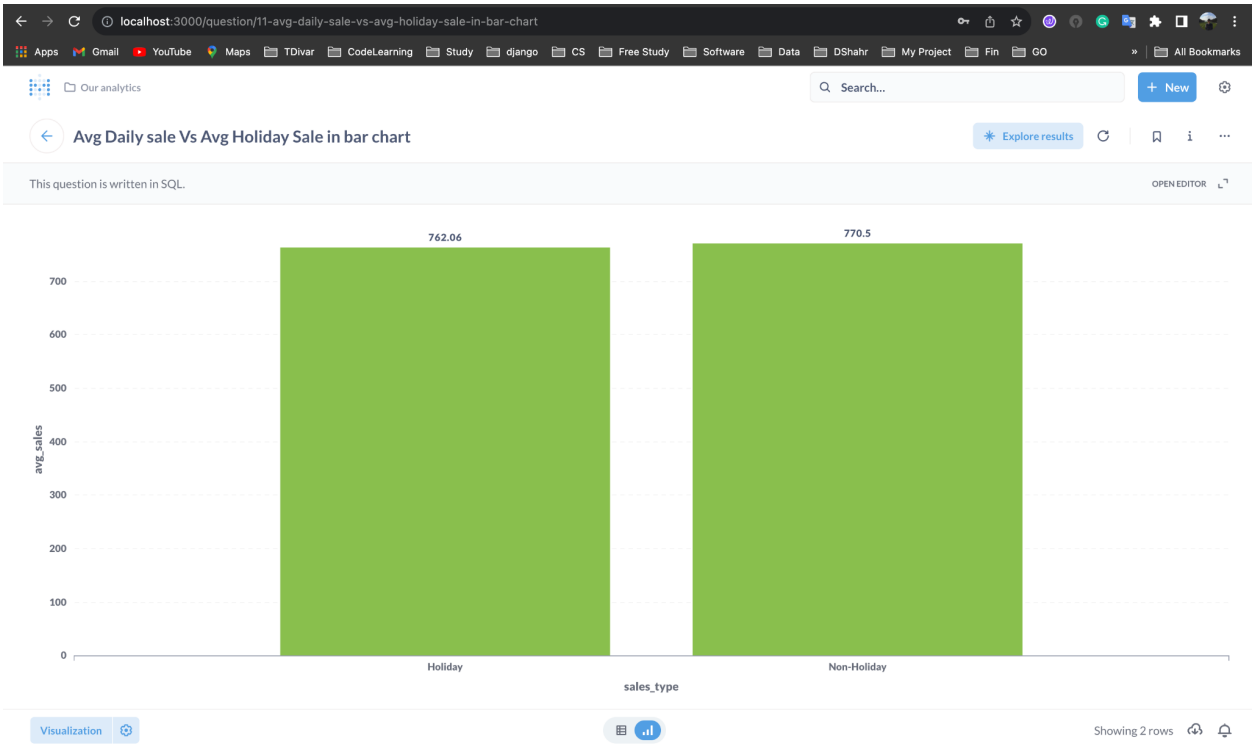
This query was executed, and appropriate visualizations, such as plots, were selected using Metabase. This query outputs the average sales on holidays compared to regular days.



Additionally, for further comparison, the following query can be used to compare the average sales on holidays versus regular days, which can be visualized using Metabase plots to achieve the following figure.

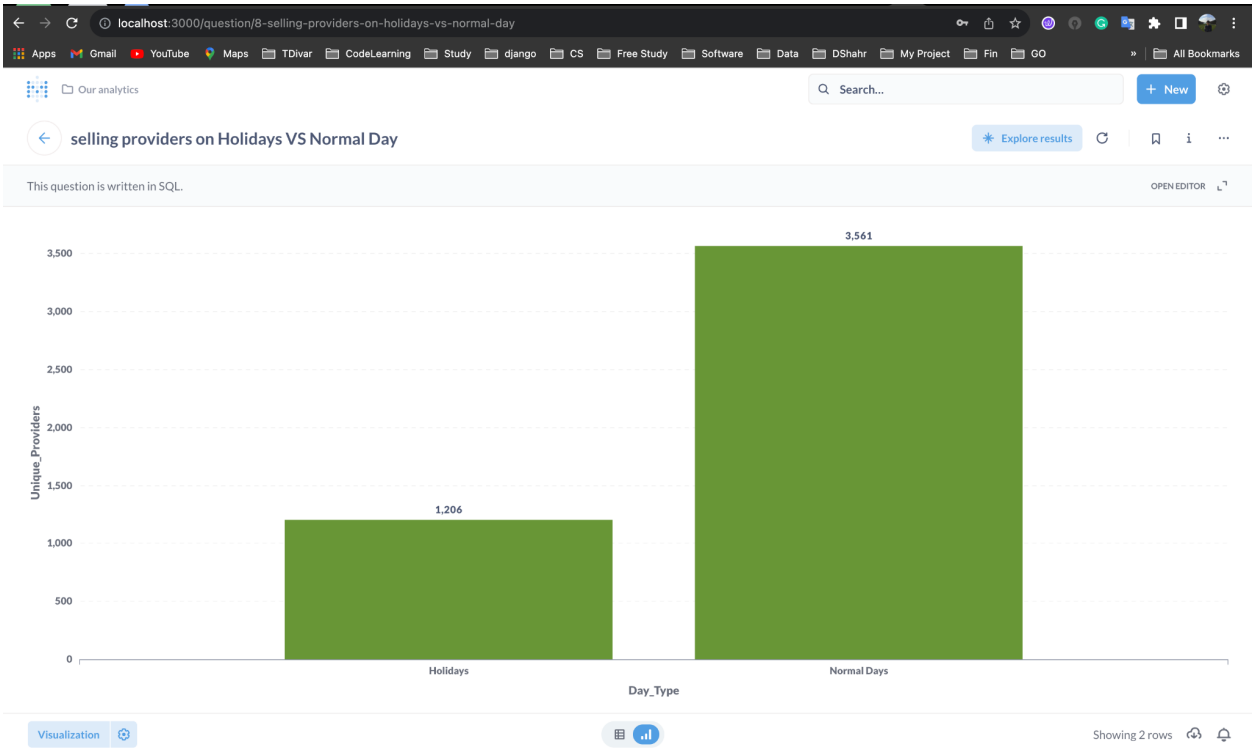
```
WITH holiday_sales AS (
    SELECT
        'Holiday' AS sales_type,
        AVG(sales) AS avg_sales
    FROM
        orders
    WHERE
        DATE(createdAt) IN (
            SELECT DATE(holiday_date) FROM holiday
        )
```

```
),
daily_sales AS (
    SELECT
        'Non-Holiday' AS sales_type,
        AVG(sales) AS avg_sales
    FROM
        orders
    WHERE
        DATE(createdAt) NOT IN (
            SELECT DATE(holiday_date) FROM holiday
        )
)
SELECT * FROM holiday_sales
UNION ALL
SELECT * FROM daily_sales;
```



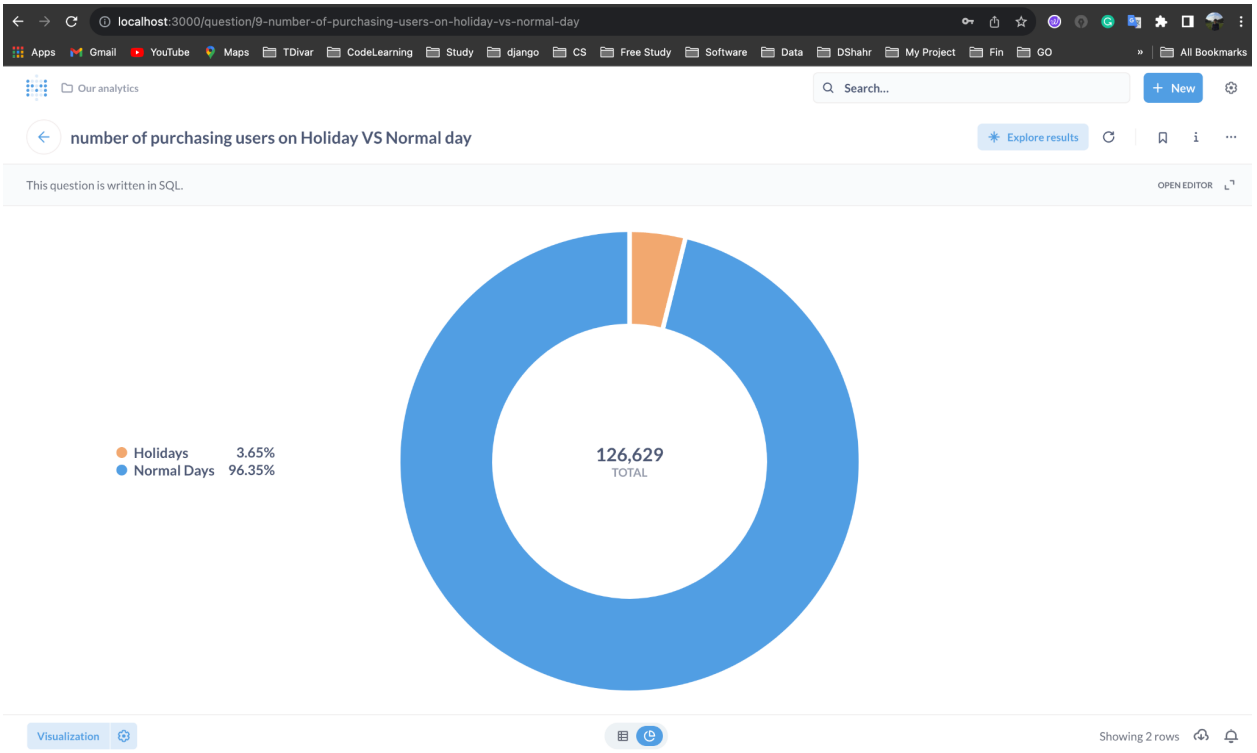
Similarly, for the question "Is the number of selling providers different on public holidays compared to normal days?", the following query can be applied in Metabase on the database, and the resulting query plot can be visualized as follows:

```
SELECT
    'Holidays' AS Day_Type,
    COUNT(DISTINCT providerId) AS Unique_Providers
FROM
    orders
WHERE
    DATE(createdAt) IN (
        SELECT DATE(holiday_date) FROM holiday
    )
UNION ALL
SELECT
    'Normal Days' AS Day_Type,
    COUNT(DISTINCT providerId) AS Unique_Providers
FROM
    orders
WHERE
    DATE(createdAt) NOT IN (
        SELECT DATE(holiday_date) FROM holiday
    );
```

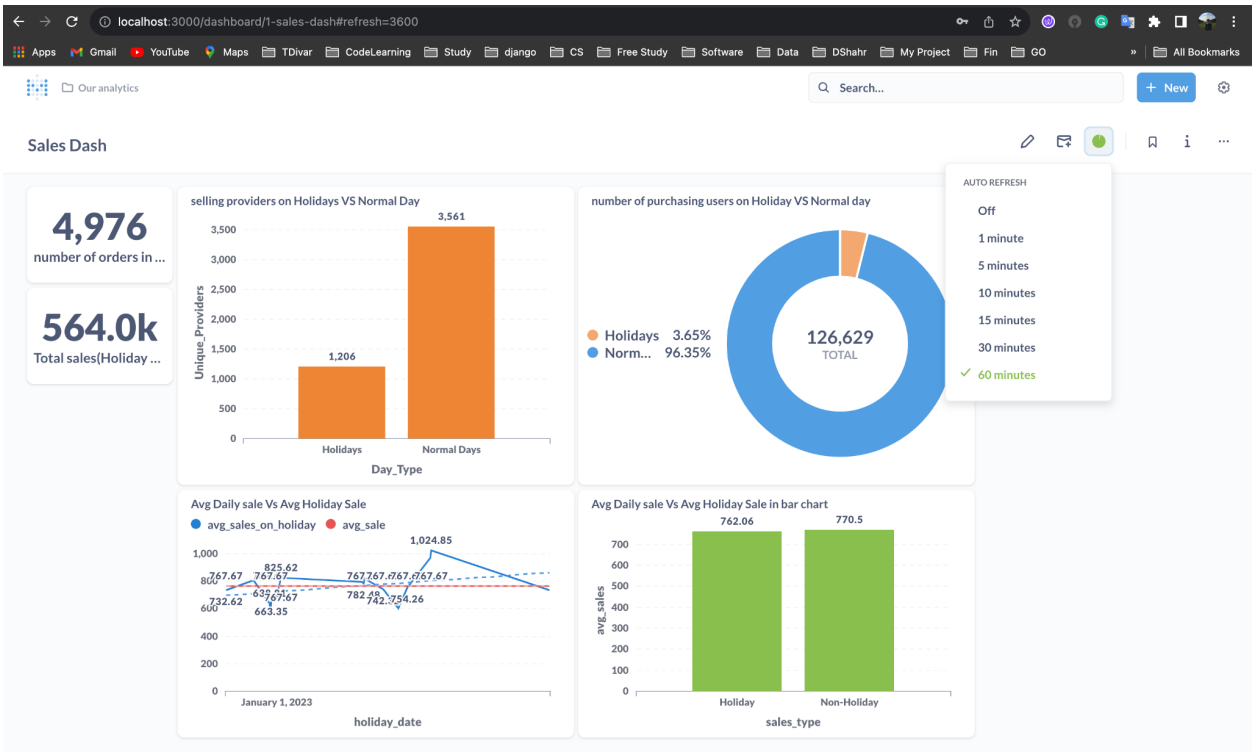


Similarly, for the question "IIs the number of purchasing users different on public holidays compared to normal days?", the following query can be applied in Metabase on the database, and the resulting query plot can be visualized as follows:

```sql
SELECT
    'Holidays' AS Day_Type,
    COUNT(DISTINCT userId) AS Unique_Users
FROM
    orders
WHERE
    DATE(createdAt) IN (
        SELECT DATE(holiday_date) FROM holiday
    )
UNION ALL
SELECT
    'Normal Days' AS Day_Type,
    COUNT(DISTINCT userId) AS Unique_Users
FROM
    orders
WHERE
    DATE(createdAt) NOT IN (
        SELECT DATE(holiday_date) FROM holiday
    );
```



Ultimately, we'll have a **dashboard** for the sales team that presents the desired concepts in the simplest forms. Utilizing Metabase's **auto-refresh** feature(as you can see in the picture), the dashboard can be automatically updated at different intervals, providing the team with the necessary insights.



The second solution:
available is to, similar to the first problem, connect to SQLite using a Python connection and perform the necessary analytics, as done in the files **problemTwo.ipynb** and **problemTwo.py**. Simply running the command **python problemTwo.py** executes this Python file, generating a PDF output of the graphs and providing brief data explanations in the console.
To automate this task, for instance, daily, a cron job can be written to execute this Python file every day and log the output.

A simple cron job in Linux:
Open your terminal.
Type crontab -e to edit the cron table.
Add the following line to schedule your script to run at midnight (00:00) every day:

```
0 0 * * * /usr/bin/python3 /path/to/your/problemTwo.py
```

In Production:

In production, we can integrate Airflow and Metabase to automate the creation and preparation of data, as well as the plotting of results.

Airflow, with its Directed Acyclic Graph (DAG) model, simplifies the orchestration and scheduling of transformations, ensuring accurate and timely execution. Each task in the pipeline is represented as a node in the DAG, and the dependencies between tasks are represented as edges. This allows for complex workflows to be visualized and managed effectively.

Meanwhile, Metabase offers an intuitive and user-friendly interface for visualizing the transformed data, enabling users to gain valuable insights effortlessly. By integrating these tools, you can enhance your data workflows, automate processes, and make informed decisions with ease.