Ideas

1. Automated Data Quality Auditor: Combine your anomaly detection with LLMs to create a system that flags potential data quality issues (e.g., missing or inconsistent records) and generates suggestions for corrections.

2. Disease Information Hub: Use an LLM to create a chatbot that provides insights into specific diseases based on historical trends in your dataset. (edited) .

3. Anomaly Explanation Assistant: Use an LLM to interpret anomalies detected by the Isolation Forest model. The LLM can provide context by generating hypotheses about why a spike or drop might occur (e.g., seasonal diseases, public health campaigns, or reporting errors).

4. Predictive Text for Outbreak Preparedness: Train an LLM to generate preparedness recommendations for potential outbreaks based on historical data patterns and detected anomalies.

Query Understanding:  
  
1.Anomaly Detection for Spikes in Disease Cases

WITH Weekly\_Avg AS (

SELECT

disease\_id,

location\_id,

AVG(current\_week) AS AvgCases,

STDDEV(current\_week) AS StdDevCases

FROM

`ba-882-group3.NNDSS\_Dataset.Weekly\_Data`

GROUP BY

disease\_id, location\_id

),

Anomalies AS (

SELECT

w.disease\_id,

w.location\_id,

w.current\_week\_flag,

w.current\_week,

wa.AvgCases,

wa.StdDevCases

FROM

`ba-882-group3.NNDSS\_Dataset.Weekly\_Data` w

JOIN

Weekly\_Avg wa

ON

w.disease\_id = wa.disease\_id AND w.location\_id = wa.location\_id

WHERE

w.current\_week > (wa.AvgCases + 2 \* wa.StdDevCases) -- Detects cases 2 standard deviations above average

)

SELECT

\*

FROM

Anomalies

ORDER BY

current\_week\_flag DESC;

output

| Row | **disease\_id** | **location\_id** | **current\_week\_flag** | **current\_week** | **AvgCases** | **StdDevCases** |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 3367990528 | 3076029967 | - | 8 | 2.1 | 2.5582111805799856 |  |
| 2 | 3367990528 | 2512340649 | - | 2 | 0.5 | 0.70710678118654757 |  |
| 3 | 2171136365 | 1688957154 | - | 2 | 0.19999999999999998 | 0.63245553203367588 |  |
| 4 | 2171136365 | 3705701479 | - | 1 | 0.1 | 0.31622776601683794 |  |
| 5 | 2171136365 | 7093385505 | - | 1 | 0.1 | 0.31622776601683 |  |

Meaning

Your query appears logically correct based on the objective of detecting anomalies where the current\_week value exceeds the mean (AvgCases) by more than 2 standard deviations (StdDevCases). Here’s a breakdown of its correctness:

### **Query Validation:**

1. **Weekly\_Avg CTE**:  
   * **Calculates Average and Standard Deviation**:
     + For each disease\_id and location\_id combination, AVG(current\_week) and STDDEV(current\_week) are computed. This is correct for deriving the statistical parameters to identify anomalies.
   * **Grouping by disease\_id and location\_id**:
     + The grouping ensures averages and standard deviations are calculated within the appropriate context.
2. **Anomalies CTE**:  
   * **Joins with Weekly\_Avg**:
     + Matches rows from Weekly\_Data to the computed averages and standard deviations by disease\_id and location\_id. This is the correct approach.
   * **Anomaly Condition**:
     + Identifies rows where current\_week is more than AvgCases + 2 \* StdDevCases. This standard deviation-based threshold is a widely accepted method for detecting anomalies.
3. **Final Selection**:  
   * Retrieves all rows flagged as anomalies, with an order specified by current\_week\_flag DESC.

### **Validation Against Output:**

* **Detected Anomalies**:  
  + Rows in the output show current\_week values significantly higher than the mean (AvgCases), with corresponding standard deviations (StdDevCases) supporting the calculation.
  + For example:
    - Row 1: current\_week = 8, AvgCases = 2.1, StdDevCases = 2.558. The anomaly threshold is 2.1+2×2.558=7.2162.1 + 2 \times 2.558 = 7.216, and 8 exceeds this, making it an anomaly.
    - This logic holds true for all rows in the output.
* **current\_week\_flag**:  
  + The current\_week\_flag is included for reference but does not affect anomaly detection in this query.

### **Suggestions for Improvement:**

* If current\_week\_flag is important for the analysis, consider its role. For instance:
  + Filter anomalies for specific weeks (e.g., current\_week\_flag = '1' for the current week).
* Ensure current\_week values of 0 or missing data are handled appropriately if they exist.

### **Conclusion:**

The query is logically correct for identifying anomalies. The output aligns with the intended logic and shows statistically significant deviations from the mean for the respective disease\_id and location\_id combinations. Well done!

2.

SELECT

disease\_id,

location\_id,

SUM(CASE WHEN current\_week\_flag = '1' THEN current\_week ELSE 0 END) AS current\_week\_total,

MAX(CASE WHEN previous\_52\_week\_max\_flag IS NOT NULL THEN previous\_52\_week\_max ELSE NULL END) AS previous\_52\_week\_max,

SUM(CASE WHEN cumulative\_ytd\_flag IS NOT NULL THEN cumulative\_ytd ELSE 0 END) AS cumulative\_ytd\_total,

SUM(CASE WHEN cumulative\_ytd\_previous\_flag IS NOT NULL THEN cumulative\_ytd\_previous ELSE 0 END) AS cumulative\_ytd\_previous\_total

FROM

`ba-882-group3.NNDSS\_Dataset.Weekly\_Data`

GROUP BY

disease\_id, location\_id

ORDER BY

disease\_id, location\_id;

Output:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Row | **disease\_id** | **location\_id** | **current\_week\_total** | **previous\_52\_week\_max** | **cumulative\_ytd\_total** | **cumulative\_ytd\_previous\_total** |  |
| 1 | 6253204 | 2331783 | 0 | 0 | 0 | 0 |  |
| 2 | 6253204 | 264087263 | 0 | 0 | 0 | 0 |  |
| 3 | 6253204 | 352323047 | 0 | 2 | 9 | 1 |  |
| 4 | 6253204 | 432799535 | 0 | 0 | 0 | 0 |  |
| 5 | 6253204 | 446991497 | 0 | 0 | 0 | 0 |  |

Meaning:  
Your query is logically sound and appropriately aggregates data as per your stated requirements. Here's a breakdown of what each field in the output represents, along with a validation of correctness:

### **Explanation of Each Column:**

1. **disease\_id and location\_id**:  
   * These fields group the data, so each row corresponds to a unique combination of disease\_id and location\_id.
2. **current\_week\_total**:  
   * This sums up the current\_week values where current\_week\_flag equals '1'.
   * If the flag was correctly implemented in the data to indicate the current week, then this column accurately reflects the total cases for the current week.
3. **previous\_52\_week\_max**:  
   * This column takes the maximum value of previous\_52\_week\_max where its flag is not NULL.
   * Ensure that the previous\_52\_week\_max\_flag is used to filter valid rows in the dataset. If the data contains nulls inappropriately, double-check its meaning.
4. **cumulative\_ytd\_total**:  
   * This sums the cumulative\_ytd values where the corresponding flag is not null.
   * If cumulative\_ytd\_flag is reliable, this correctly aggregates year-to-date cumulative totals.
5. **cumulative\_ytd\_previous\_total**:  
   * Similar to the above, this sums the cumulative\_ytd\_previous values where the flag is not null.
   * This column reflects the previous year's year-to-date cumulative totals.

### **Validation Against Sample Output:**

1. **Rows with All Zeros**:  
   * For rows where all the totals are zero, confirm that the data has current\_week\_flag = '1', non-null previous\_52\_week\_max, or valid cumulative\_ytd\_flag entries. If such data doesn't exist for these disease\_id and location\_id pairs, the zeros are expected.
2. **Non-Zero Values**:  
   * For rows with non-zero values (e.g., current\_week\_total, previous\_52\_week\_max, etc.), validate by cross-checking with a few individual records in the source data to confirm the aggregations are accurate.

### **Suggestions:**

* If current\_week\_flag, previous\_52\_week\_max\_flag, etc., contain other meaningful values (e.g., 'N', 'NC'), you might need additional conditions to handle those cases correctly.
* Use filters like WHERE disease\_id = '6253204' to inspect the raw data for specific disease\_id and location\_id combinations, ensuring your query logic aligns with the actual data patterns.

### **Conclusion:**

The query appears correct based on the provided schema and output. Ensure the flags and conditions used in the query are accurately aligned with the intended logic of the dataset.

3.

SELECT

disease\_id,

SUM(current\_week) AS TotalCases

FROM

`ba-882-group3.NNDSS\_Dataset.Weekly\_Data`

WHERE

disease\_id = 8704807513 -- Replace with specific DiseaseID

GROUP BY

disease\_id

ORDER BY

TotalCases DESC;

Output:

|  |  |  |  |
| --- | --- | --- | --- |
| Row | **disease\_id** | **TotalCases** |  |
| 1 | 8704807513 | 20 |  |

Explanation:  
**Query Analysis:**

1. **Purpose**:
   * This query calculates the total current\_week cases for disease\_id = 8704807513.
2. **Logic**:
   * SUM(current\_week) correctly aggregates the values of current\_week for all rows where disease\_id = 8704807513.
   * GROUP BY disease\_id ensures the sum is grouped by the specified disease\_id.
3. **Output Validation**:
   * The result shows TotalCases = 20 for disease\_id = 8704807513.
4. **Correctness**:
   * The query is logically correct.
   * The accuracy of the result depends on the underlying data in the current\_week column. Specifically:
     + If the current\_week values for disease\_id = 8704807513 sum up to 20, the result is correct.
     + If there are missing or null values in the current\_week column, the result may be impacted, as SUM() skips nulls by default.

4. Overall Trend Comparison Across Years

SELECT

disease\_id,

'All-Time' AS Period,

SUM(current\_week) AS TotalCases

FROM

`ba-882-group3.NNDSS\_Dataset.Weekly\_Data`

WHERE

disease\_id = 8704807513

GROUP BY

disease\_id, Period

ORDER BY

TotalCases DESC;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Row | **disease\_id** | **Period** | **TotalCases** |  |
| 1 | 8704807513 | All-Time | 20 |  |

Explanation:

### **Analysis of the Query:**

1. **Purpose**:  
   * The query calculates the total number of cases (SUM(current\_week)) for disease\_id = 8704807513 across the entire dataset.
2. **Logic**:  
   * The SUM(current\_week) aggregates all values in the current\_week column for rows where disease\_id = 8704807513.
   * The 'All-Time' AS Period provides a label indicating the aggregation is across the entire dataset.
3. **Output**:  
   * TotalCases = 20: This indicates that the sum of current\_week values for all rows matching disease\_id = 8704807513 is 20.
4. **Correctness**:  
   * If the raw data has 20 rows with current\_week = 1 (as confirmed in your earlier queries), this result is accurate.
   * The query groups by disease\_id and the literal string 'All-Time' (which doesn’t affect aggregation), ensuring a single row in the output.

### **Validation:**

* You already ran a query to inspect the raw data for disease\_id = 8704807513:
  + Rows included multiple current\_week values, with most being 0 and some 1.
  + The total count of 1s was consistent

Creating a chatbot powered by a large language model (LLM) to provide insights into specific diseases based on historical trends involves several steps. Here's how you can approach this task:

### **1. Define Chatbot Objectives and Features**

* **Objective:** The chatbot should provide insights into specific diseases based on historical trends in the NNDSS dataset.
* **Core Features:**
  + Query disease trends by name and time period.
  + Provide summaries of historical data and trends.
  + Highlight anomalies or unusual spikes for specified diseases.
  + Offer explanations for trends based on pre-processed insights.

### **2. Prepare the Dataset for Integration**

* **Data Preprocessing:**
  + Ensure the NNDSS dataset is cleaned and structured for querying.
  + Create summaries of disease data (e.g., yearly trends, seasonal patterns, anomalies).
  + Store processed data in a format accessible to the chatbot (e.g., database or API).
* **Example Fields for Each Disease:**
  + Disease Name
  + Year/Month
  + Number of Cases
  + Trend Indicators (e.g., increasing, decreasing, stable)
  + Anomaly Flags
  + Insights or Notes

### **3. Select an LLM and Integration Tools**

* **LLM Options:**
  + OpenAI's GPT models
  + Google's Bard API
  + Hugging Face's Transformer models
* **Frameworks for Integration:**
  + Use Python libraries like LangChain or LLMChain for chaining LLMs with other tools.
  + Employ tools like Streamlit or Flask for building the chatbot interface.

### **4. Build the Chatbot Workflow**

1. **Frontend Development:**
   * Create a user-friendly interface for users to input queries.
   * Tools: Streamlit (for simplicity), React, or Flask.
2. **Backend Development:**
   * **Query Handling:**
     + Parse user input to extract disease name, time period, and specific questions.
   * **Data Querying:**
     + Use SQL or Python to retrieve relevant data from the processed dataset.
   * **LLM Integration:**
     + Use the LLM to process user queries and contextualize responses with historical trends.
   * **Response Formatting:**
     + Structure the response in a clear and concise manner.
3. **LLM Prompt Design:**

Use a system message to guide the LLM, such as:  
 You are a chatbot designed to provide disease insights based on historical trends from the NNDSS dataset. Answer questions with data-driven insights and highlight anomalies or trends where applicable.

Example User Prompt:  
 What were the trends in influenza cases over the past 5 years? Were there any anomalies?

### **5. Train or Fine-Tune (Optional)**

* Fine-tune the LLM with domain-specific knowledge or FAQs if needed.
* Use tools like Hugging Face’s Trainer or OpenAI’s fine-tuning capabilities.

### **6. Implement the Chatbot**

* **Development:**
  + Create API endpoints for querying data and interacting with the LLM.
  + Tools: FastAPI, Flask, or Django.
* **Integration:**
  + Combine the frontend with the backend and deploy the chatbot.

### **7. Deployment**

* **Hosting Options:**
  + Cloud Platforms: AWS, Google Cloud, Azure.
  + Web Hosting: Streamlit Cloud for simple deployments.
* **Scaling:**
  + Use a scalable backend to handle multiple simultaneous users.

### **8. Test and Optimize**

* Test the chatbot for:
  + Accuracy of disease insights.
  + Clarity and relevance of responses.
  + Performance under load.
* Gather user feedback and improve the chatbot iteratively.

### **Example Code Snippets**

#### **Query Handling**

import sqlite3

import pandas as pd

def get\_disease\_trends(disease, start\_year, end\_year):

conn = sqlite3.connect('nndss\_data.db')

query = f"""

SELECT year, cases

FROM disease\_trends

WHERE disease\_name = '{disease}' AND year BETWEEN {start\_year} AND {end\_year}

"""

data = pd.read\_sql\_query(query, conn)

conn.close()

return data

#### **LLM Integration**

import openai

def generate\_insight(query, data\_summary):

prompt = f"""

You are a disease trend analyst. Based on the following data, provide insights:

{data\_summary}

User Query: {query}

"""

response = openai.ChatCompletion.create(

model="gpt-4",

messages=[{"role": "system", "content": "You are an expert in disease analytics."},

{"role": "user", "content": prompt}]

)

return response['choices'][0]['message']['content']

### **9. Enhance with Advanced Features**

* **Natural Language Queries:** Add NLP capabilities to interpret free-text user queries.
* **Interactive Visuals:** Integrate visualizations (e.g., trend charts) in the chatbot responses.
* **Personalization:** Allow users to set preferences (e.g., diseases of interest, alert thresholds).

Would you like help implementing any specific part of this workflow?