Analytics Engineer Take Home Challenge

Instruction

In this data challenge, we would like you to answer some questions about a hypothetical Account Takeover Product (ATO) using SQL (note that while Abnormal has an ATO product similar to that presented here, that data provided in this challenge is not actual product data). The ATO data is included as the attached csv files.

You are welcome to use any tool you are comfortable with to complete this challenge, but we ask that all questions are answered using SQL. As output, please return your working notebook / script / .sql files, plus optionally any answer write-up if not already included in a working notebook. If desired, see included instructions for how to use SQL for analysis within a Jupyter notebook.

As a next step after your completed submission, we will be scheduling a 1-hour follow up call with you, discussing your results, as well as exploring your thoughts on how best to measure the health of the ATO product and establish a product metrics program.

Challenge prompt

ATO product description

The Account Takeover product ingests signals related to end user activity, including:

- Sign in events
- Mailfilter events (example: the creation of a mail rule to automatically route messages from `acme_marketing.org` to junk)
- Posture change events (example: User permissions are upgraded from low to high)
- Data download events

Multiple types of detectors run on these events. When an event is flagged as suspicious with sufficient confidence, an Account Takeover Case is created. The case is surfaced to customers in the Customer Portal, and a notification is sent to the customer. Customers have 3 options for case remediation:

1. Notification only
2. Revoke active sessions
3. Revoke active sessions and Reset password

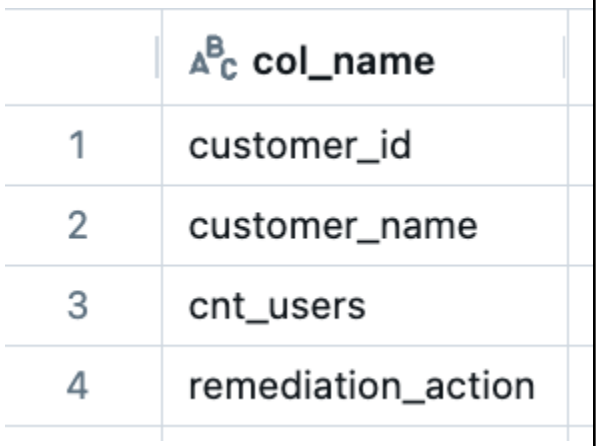
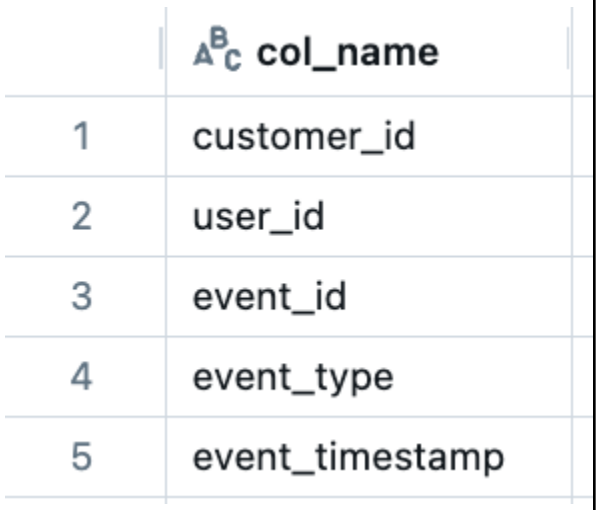
If the customer has remediation actions enabled beyond the notification, these will be automatically executed. Within the Customer Portal, the customer is able to view the case page, which has details about the case event timeline, and to respond to the case by selecting from a dropdown menu, with the following options:

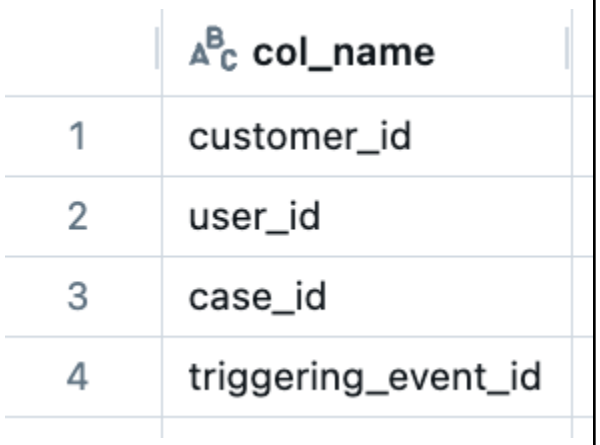
- `Acknowledge` (this leaves the case open)
- `Not a compromise` (this marks the case as a False Positive internally and closes the case)
- `Resolve` (this marks the case as a True Positive internally and closes the case)

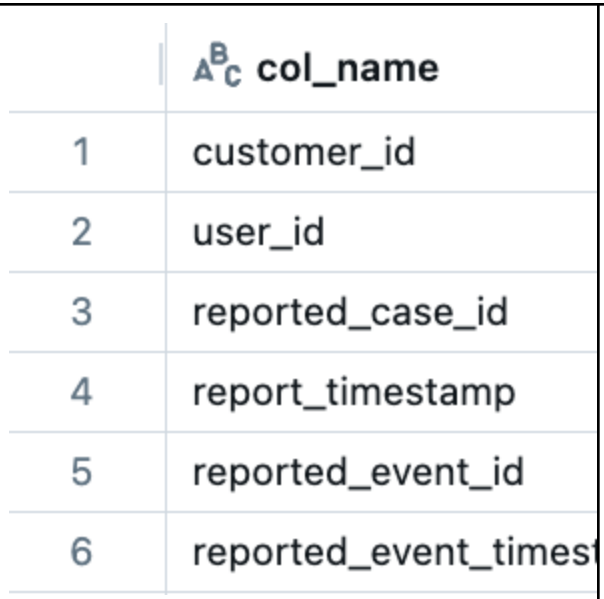
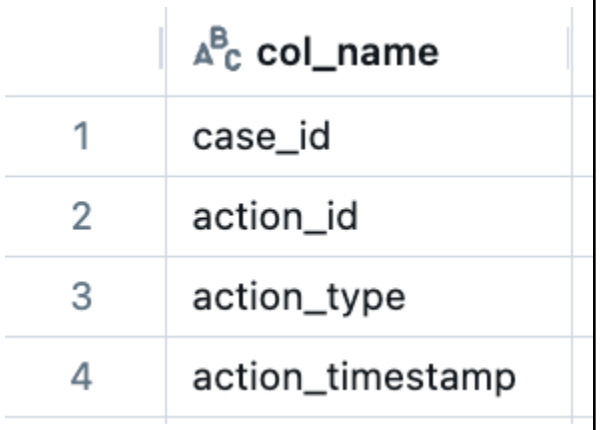
Selecting any option also allows the customer to enter a comment in a free form text box.

If a customer has an employee account breach that was not flagged by Abnormal (ie: a False Negative), they are able to report that within the Customer Portal by using the `Report Account Breach` module. This will create a customer ATO case and will trigger the case remediation actions enabled by the customer.

ATO product source data table

Table name	Description	Schema	Notes
`customers`	Customer metadata		
`user_events`	All events that we record for each user		

`detector_events`	All events that are flagged as suspicious and the detector that flagged them		
`case_events`	ATO cases that are triggered by Abnormal		The `triggering_event_id` in the `case_events` table is the same as the `event_id` in the `user_events` and `detector_events` tables

`customer_case_events`	All cases missed by Abnormal that are created by customer reports		The `reported_event_id` in the `customer_case_events` table is the same as the `event_id` in the `user_events` table
`case_action_events`	All actions that Abnormal took on ATO cases		

`customer_action_events`	All subsequent Portal actions that customers took on an ATO case		

	col_name
1	customer_id
2	case_id
3	customer_action_type
4	customer_action_ts
5	customer_comment

Questions

1. Describing the data
    a. How many customers do we have ATO product data for?
    b. Over what time period does the ATO data exist?
    c. How many ATO detectors are there?
    d. What is the range of customer sizes by number of users?

2. Calculating metrics
    a. What is the average monthly false negative rate (FNR) of the ATO product?
        i. $FNR = FN / (TP + FN)$
    b. What is the average monthly false discovery rate (FDR) of the ATO product?
        i. $FDR = FP / (TP + FP)$
    c. What fraction of user events are flagged by a detector at any confidence? At high confidence?
    d. Calculate the precision of each detector based on cases which are made visible to customers. In other words, if a case is not made visible to the customer, it does not count as either a TP or an FP. Which detector has the highest precision? Which detector has the lowest precision?
        i. $Precision = TP / (TP + FP)$
    e. Which detector flags the highest number of customer-confirmed true positive cases?

3. Building a data pipeline
    a. Calculating each metric above individually only requires a small subset of tables joined per metric. However, we want to enable all our downstream consumers, including Product and Engineering stakeholders, to have one table act as a source of truth for all analyses. We also want this same table to be upstream of all metrics calculations. Write a query which joins together all necessary information into a single table that can be used to answer all of the metrics questions above.

Performing analysis in SQL within a Jupyter notebook using duckdb

duckdb is a free, open source OnLine Analytical Processing (OLAP) DataBase Management System (DBMS). It is optimized for doing data analysis and performing complex queries. To use duckdb for analysis within a Jupyter notebook, first install it using pip (for other installation options, including for Windows machines, see documentation):

In a terminal	`$ pip install duckdb`

Open your notebook in the same folder as your analysis csv files. Import duckdb at the top of your notebook:

In a cell	`import duckdb`

SQL queries can be executed against csv files in duckdb by using the file name:

In a cell	`duckdb.sql("SELECT * FROM customers.csv;").show()`

Or by using the `read_csv` method (below we store the query in the relation `customers`, which can then be directly referenced in future queries):

In a cell	`customers = duckdb.read_csv("customers.csv")` `customer_case_events = duckdb.read_csv("customer_case_events.csv")`

The output of SQL queries can also be saved as a csv file:

In a cell	`customers = duckdb.read_csv("customers.csv")`
	`customer_case_events = duckdb.read_csv("customer_case_events.csv")`
	
	`query = """`
	`SELECT`
	`  c.*,`

```
  cce.*
FROM customers c
JOIN customer_case_events cce ON cce.customer_id = c.customer_id;
"""

enhanced_customer_case_events = duckdb.sql(query)
enhanced_customer_case_events.write_csv("enhanced_customer_case_events.csv")
```