Shopify Abandoned Checkout Recovery Automation

Section 1: Initialization & Abandoned Checkout Fetch

Instead of using a scheduled trigger, I followed the requirement that the flow should start when a checkout is abandoned. Shopify doesn't seem to provide a direct webhook for abandoned checkouts (or at least I couldn't find one). If that does exist, it's easy to swap in later.

For now, I've created a webhook that listens to checkout updates. Whenever a checkout is updated (which typically happens when a user interacts with it), the workflow is triggered.

Once triggered, the flow immediately calls Shopify's GraphQL Admin API to fetch the latest list of abandoned checkouts. The API gives us:

- Customer info (name, email, etc.)
- Cart contents (product names, quantities)
- Total price and currency
- Any discount codes
- The abandoned checkout recovery URL

This is the raw data we process and personalize later in the flow.

Section 2: Deduplication & Customer Enrichment

For every abandoned checkout retrieved, the workflow loops through and handles it individually.

1. Deduplication Check

The first thing I do is check Airtable to see if this checkout has already been messaged.

- I search for a record matching the checkout_id.
- If there's no match (isEmpty = true), we continue.
- If the record exists, we skip it — no duplicates.

2. New Checkout Handling

If it's a new one, I log it in Airtable right away. This record includes the customer email, product details, total price, recovery link, etc. It acts as a snapshot and makes the whole process traceable.

3. Customer Enrichment

Next, I try to enrich the data using the customer's email. I query Shopify to fetch their full profile and order history.

In my test data, this didn't return anything meaningful, so I couldn't demonstrate advanced personalization. But I've built the structure to support this if/when more data becomes available, like past purchases, tags, or lifetime value.

The key focus here was on building the flow logic so it can scale or be improved later as better customer data comes in.

Section 3: AI Message Generation, Feedback Loop, Evaluation & Klaviyo Trigger

This is the heart of the automation. It's where the system crafts a message, checks how persuasive it is, and either sends it or improves it based on feedback.

1. AI-Powered Message Writing

Once a checkout is identified as new and unmessaged, I pass the data to GPT-4o. The system prompt gives it the role of a friendly and knowledgeable beauty advisor.

The message uses whatever details we have: customer name if available, the cart items, the total price, any discount code, and the recovery link. It's generated as a single flowing paragraph without line breaks, written to feel warm, helpful, and personal.

2. Feedback Loop via Persuasion Evaluation

That message is then passed to another AI, which scores it on persuasiveness from 1 to 10 and provides a suggestion for improvement.

If the score is 7 or more, we're done.

If the score is below 7, the suggestion is passed back to the message writer, which tries again with the feedback in mind.

This loop continues until we either get a persuasive enough message or we hit a maximum of 3 tries. The suggestion text is stored in Airtable so it can persist between nodes.

4. Klaviyo Event Trigger

Once a message passes the evaluation or we hit the attempt limit, the final message gets sent to Klaviyo as a custom event.

The event includes the message content, the persuasion score, the customer's email, and the checkout ID. This lets Klaviyo trigger any follow-up automations and also gives us analytics around message effectiveness.

Additional thoughts / scaling to production / scope for improvement

1. Shopify provides a native API to retrieve abandoned checkouts, which follows its internal logic for determining abandonment. While this was sufficient for the test task, in production it might not align with the brand's definition of abandonment. For instance, a business may want to wait 1–2 hours before acting or exclude sessions where email wasn't captured. A custom trigger with a delay and verification step could provide more control and align better with business logic.

2. In production, it would be critical to filter out low-priority checkouts early — for example, discarding checkouts without email addresses, or those older than 3 days. This would reduce the number of API calls, Airtable writes, and OpenAI requests, leading to better performance and cost control.

3. While a per-checkout trigger provides immediacy, it may not scale well under heavy traffic due to execution limits and potential API rate constraints. A more scalable approach would involve a scheduled workflow that pulls all recent abandoned checkouts in batches, filters them, and processes each one. This makes retries, logging, and error handling easier to manage.

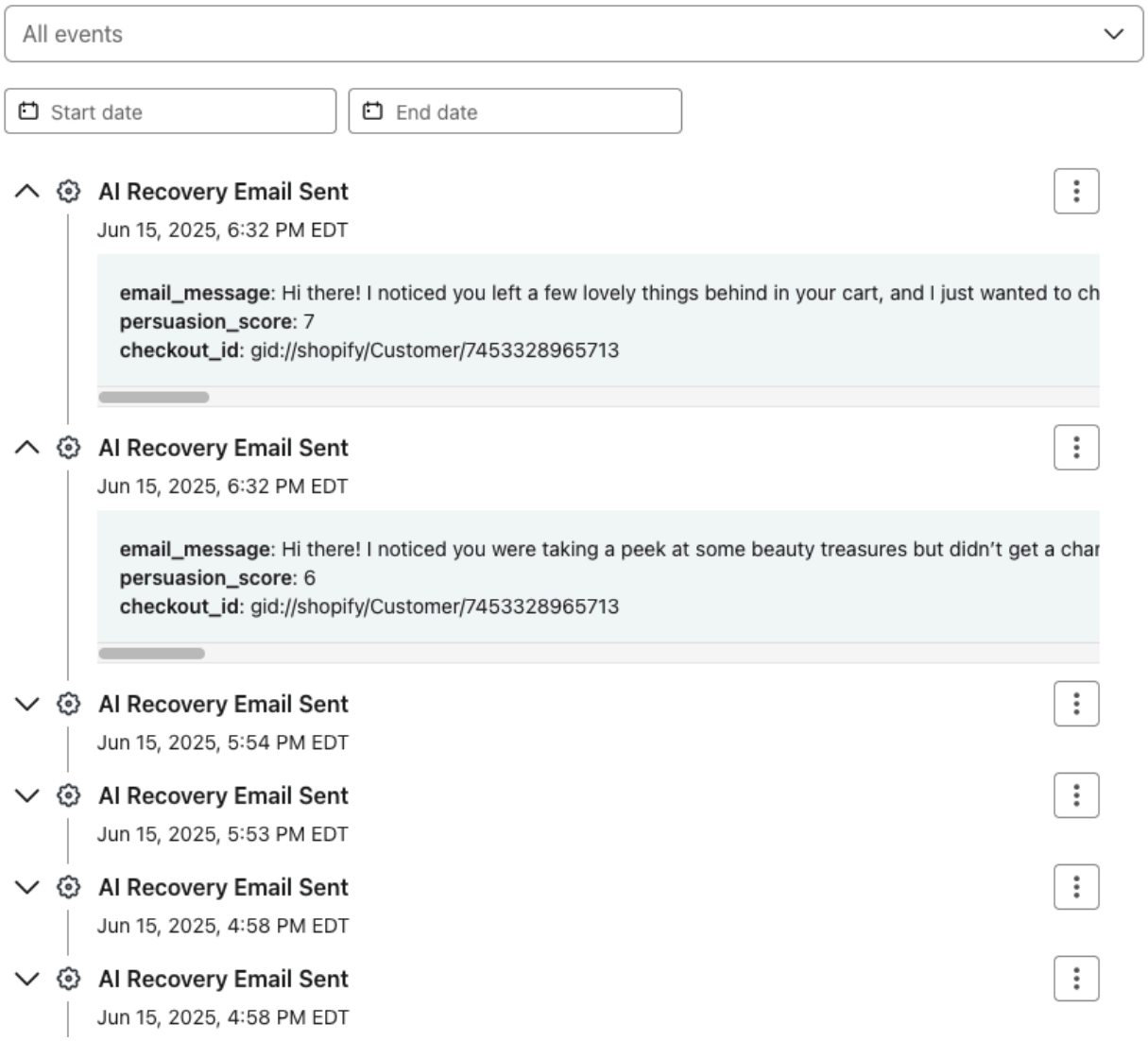
4. To maintain a consistent brand voice, we could encode eBeauty's tone and values into the assistant using a persistent system prompt, effectively acting as a constitution or brand style guide. This would ensure that all generated messages stay aligned with tone preferences (e.g., warm, expert, non-pushy), regardless of the operator.

5. Historical data on successful abandoned cart recoveries could be used to improve the assistant's output. This could be done via few-shot examples in the system prompt or by storing them in a vector store for retrieval-augmented generation (RAG). Over time, this would create a feedback loop where high-performing messages help shape future ones.

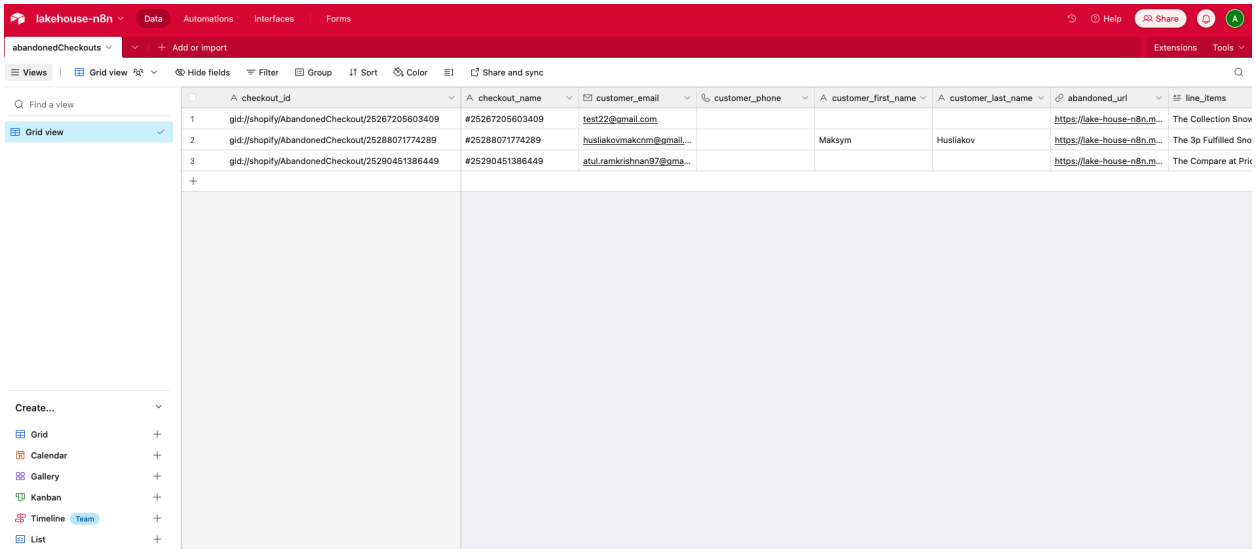
6. To handle the feedback loop between the evaluator and the message writer, I used Airtable to store both the latest suggestion and the number of attempts. I initially tried to keep this in-memory within the workflow, but persisting state between iterations turned out to be trickier than I expected. This Airtable-based approach isn't the cleanest, but it works reliably and also gives transparency into what's happening at each step. There's probably a cleaner way to handle this loop internally without relying on an external database. I'll dig into the docs over the next few days to see if I can figure out a better pattern.

Possibly Relevant Screenshots

Klaviyo



Airtable


lakehouse-n8n
Data
Automations
Interfaces
Forms
Help
Share
abandonedCheckouts
Add or import
Extensions
Tools
Views
Grid view
Hide fields
Filter
Group
Sort
Color
Share and sync
Find a view
Grid view
checkout_id
checkout_name
customer_email
customer_phone
customer_first_name
customer_last_name
abandoned_url
line_items
1
gid://shopify/AbandonedCheckout/25267205603409
#25267205603409
test22@gmail.com
https://lake-house-n8n.m...
The Collection Snow
2
gid://shopify/AbandonedCheckout/25288071774289
#25288071774289
husliakovmakcnm@gmail....
Maksym
Husliakov
https://lake-house-n8n.m...
The 3p Fulfilled Sno
3
gid://shopify/AbandonedCheckout/25290451386449
#25290451386449
atul.ramkrishnan97@gma...
https://lake-house-n8n.m...
The Compare at Pric
Create...
Grid
Calendar
Gallery
Kanban
Timeline
Team
List