Google Cloud

Enriching Databricks Pipelines with Google Cloud's pre-trained ML APIs

Yang Li and Vinny Vijeyakumaar

1. Abstract

Data enrichment is a core step in data pipelines for adding value to raw data for business stakeholders. Enrichment includes cleansing, enforcing quality controls, incorporating external data sources such as weather information, and much more.

A valuable enrichment step is bringing meaning to unstructured data (e.g., free-form text, documents, images, and videos). This involves applying ML classification models to the data to generate metadata to be stored alongside them. However, this is rarely easy to achieve. Organizations encounter challenges when dealing with distinct platforms for structured and unstructured data. They also find the process of developing or accessing high-quality models to be a time-consuming task. Operating and maintaining these models while keeping costs at a minimum adds to the complexity of these struggles.

With Databricks running on Google Cloud's ML solutions, these problems have become a thing of the past.

2. Databricks on Google Cloud

Databricks on Google Cloud allows you to store all of your data on a simple, open lakehouse platform that combines the best of data warehouses and data lakes to unify all of your analytics and AI workloads. Tight integration with Google Cloud Storage, BigQuery, and the Google Cloud AI Platform enables Databricks to work seamlessly across data and AI services on Google Cloud.

A pipeline's steps can include interacting with Google Cloud's ML APIs to generate meaning about the data. Google Cloud's AI tools are armed with the best of Google's research and technology to help developers focus exclusively on solving problems that matter. Ready-to-use ML services, such as the Natural Language API, are pre-trained by Google based on its corpus of data. Therefore, as a practitioner, you don't need to worry about creating a model from scratch. Instead of bearing the high cost of training these models independently, data engineers and data scientists can immediately start leveraging these models via API calls.

3. Workflow: Classifying Customer Reviews

Let's see this in action for a common scenario in retail marketplaces. Marketplaces rely on quality suppliers and customers to succeed - meaning nurturing high-quality suppliers and remedying low-quality suppliers. A critical signal for supplier quality is customer reviews.

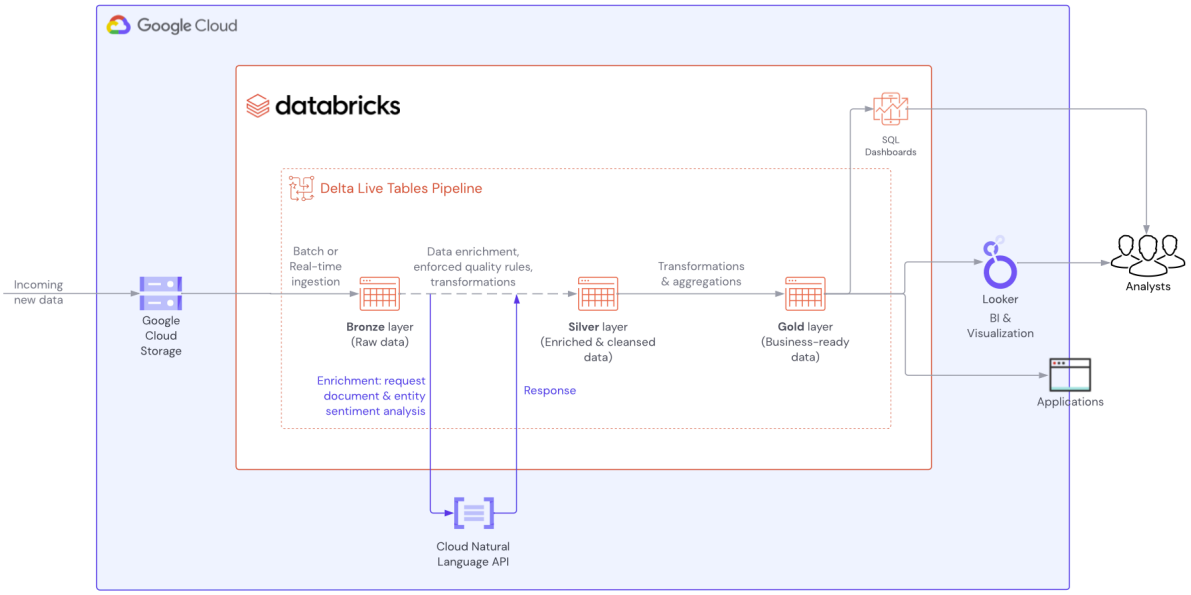
Collecting customer reviews from websites and review partners is straightforward, but making sense of the data at scale is not.

For example, how do we make sense of these customer reviews at scale? How can we break it down into specific components that they enjoyed or disliked?

Let's Imagine we're a marketplace that connects consumers and businesses for reservations and home deliveries. We have a rich set of unstructured customer reviews that our business stakeholders are asking to derive value from:

- C-suite: want an overall pulse of supplier quality
- Marketing: want to identify, nurture and promote high-quality suppliers
- Product: want to identify and remedy factors that contribute towards low-quality suppliers

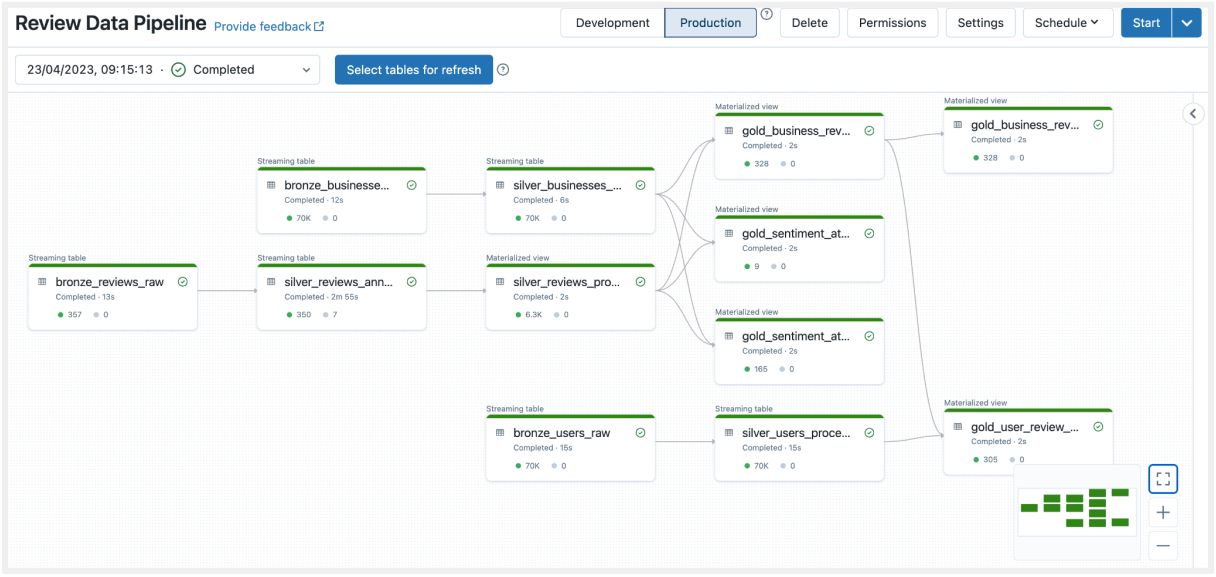
In this example, we'll utilize Yelp's open dataset of business reviews and leverage Databricks' Delta Live Tables and Google Cloud's ML APIs to build a declarative data processing pipeline that cleanses and enriches the data.



The Delta Live Tables pipeline will step us through processing the data through the medallion architecture:

- **Bronze**: ingest raw customer review data from Google Cloud Storage (GCS)
- **Silver**: cleanse, enrich, and validate data
    - We will enrich the data with entity and sentiment analysis provided by Google's Natural Language APIs
- **Gold**: aggregate the enriched data into business-ready tables for our C-suite, Marketing, and Product teams

Once implemented, our pipeline flow will look like this:



4. Prerequisites

To follow along, there are prerequisites to consider with both your Databricks and Google Cloud setup.

Databricks

- A Databricks Workspace (see the getting started guide)

- A Google Service Account created in your Workspace's Google Cloud project

Google Cloud

- Enable the Natural Language API
- Google Cloud Storage bucket to receive your incoming customer review data.
    - Ensure the Service Account created above has the Storage Object Viewer role in reading from this bucket

5. Building your Delta Live Tables Pipeline

We'll utilize Delta Live Tables for its simplicity in declaring and managing complex ETL pipelines as we transform data from its raw state to business-ready gold layer tables.

Delta Live Tables (DLT) is the first ETL framework that uses a simple declarative approach to building reliable data pipelines. DLT automatically manages your infrastructure at scale so data analysts and engineers can spend less time on tooling and focus on getting value from data.

- **Accelerate ETL development:** Declare SQL/Python and DLT automatically orchestrates the DAG, handles retries, schema evolution, etc.
- **Automatically manage your infrastructure:** Automates complex activities like recovery, auto-scaling, and performance optimization.
- **Ensure high data quality:** Deliver reliable data with built-in quality controls, testing, monitoring, and enforcement.
- **Unify batch and streaming:** Get the simplicity of SQL with the freshness of streaming with one unified API.

You can declare DLT pipelines in SQL or Python. In this example, we'll primarily focus on SQL and use Python to handle our interactions with the Natural Language API.

5.1 Ingesting Data into the Bronze Layer

The first step is to populate a bronze-layer table with the latest data delivered into Google Cloud Storage.

Let's assume that new reviews, business, and user data is processed upstream and frequently delivered to the following GCS paths:

- Review data: gs://vinny-demo-yelp-landing-zone/landing_review/
- Business data: gs://vinny-demo-yelp-landing-zone/landing_business/
- User data: gs://vinny-demo-yelp-landing-zone/landing_users/

Using Databricks' Auto Loader capabilities, you are guaranteed that only new files in the bucket are processed whenever your DLT pipeline is run. It can scale to support near real-time ingestion of millions of files per hour.

Let's set up our ingestion steps:

Unset

```
CREATE OR REFRESH STREAMING LIVE TABLE reviews_bronze_raw
COMMENT "Raw review data"
AS SELECT *
FROM CLOUD_FILES("gs://vinny-demo-yelp-landing-zone/landing_review/",
 "json", MAP("cloudFiles.inferColumnTypes", "true")
)
```

Unset

```
CREATE OR REFRESH STREAMING LIVE TABLE businesses_bronze_raw
COMMENT "Raw business data"
AS SELECT *
FROM CLOUD_FILES("gs://vinny-demo-yelp-landing-zone/landing_business/",
 "json", MAP("cloudFiles.inferColumnTypes", "true")
)
```

Unset

```
CREATE OR REFRESH STREAMING LIVE TABLE users_bronze_raw
```

```
COMMENT "Raw user data"
AS SELECT *
FROM CLOUD_FILES("gs://vinny-demo-yelp-landing-zone/landing_users/",
 "json", MAP("cloudFiles.inferColumnTypes", "true")
)
```

For each of these we are:

- Declaring a DLT STREAMING LIVE table. Don't let the name "streaming" deceive you; these can be used for both streaming and batch data ingestion. This type of table is used for incremental handling of new data.
- CLOUD_FILES is a pseudo-table used to configure Auto Loader:
    - It points to the GCS path that should be processed.
    - json instructs it to only look at JSON files in the folder. Other file types in the bucket are ignored.
    - inferColumnTypes instructs Auto Loader to automatically infer column types based on the data held within. This helps to minimize our data type transformation steps later down the line.

5.2 Bronze to Silver: Enriching review data with Google's Natural Language APIs

Now that we have the latest data in our bronze tables, we'll clean, transform and enrich that data and materialize it in our silver layer tables. We'll use the Natural Language API to provide enrichment for our unstructured data.

Natural Language API

Google's Natural Language API offers a variety of powerful features for processing and analyzing text, such as Entity analysis, Sentiment analysis, Syntax analysis, Text classification, etc. In the following example, we will illustrate the process of conducting sentiment analysis and entity analysis on customer reviews.

Setup the Natural Language API

To use Google's Natural Language APIs, you will need to follow these general steps:

- Enable the Natural Language API in your Google Cloud project
- Generate an API key for authentication purposes. However, for the purposes of this exercise, we'll be authenticating via a service account
- Install the client library for your preferred programming language
- Write code to access the Natural Language API
- Pass text or content to the Natural Language API for analysis and processing, and receive a response containing information about the text's sentiment, entities, syntax, and more.

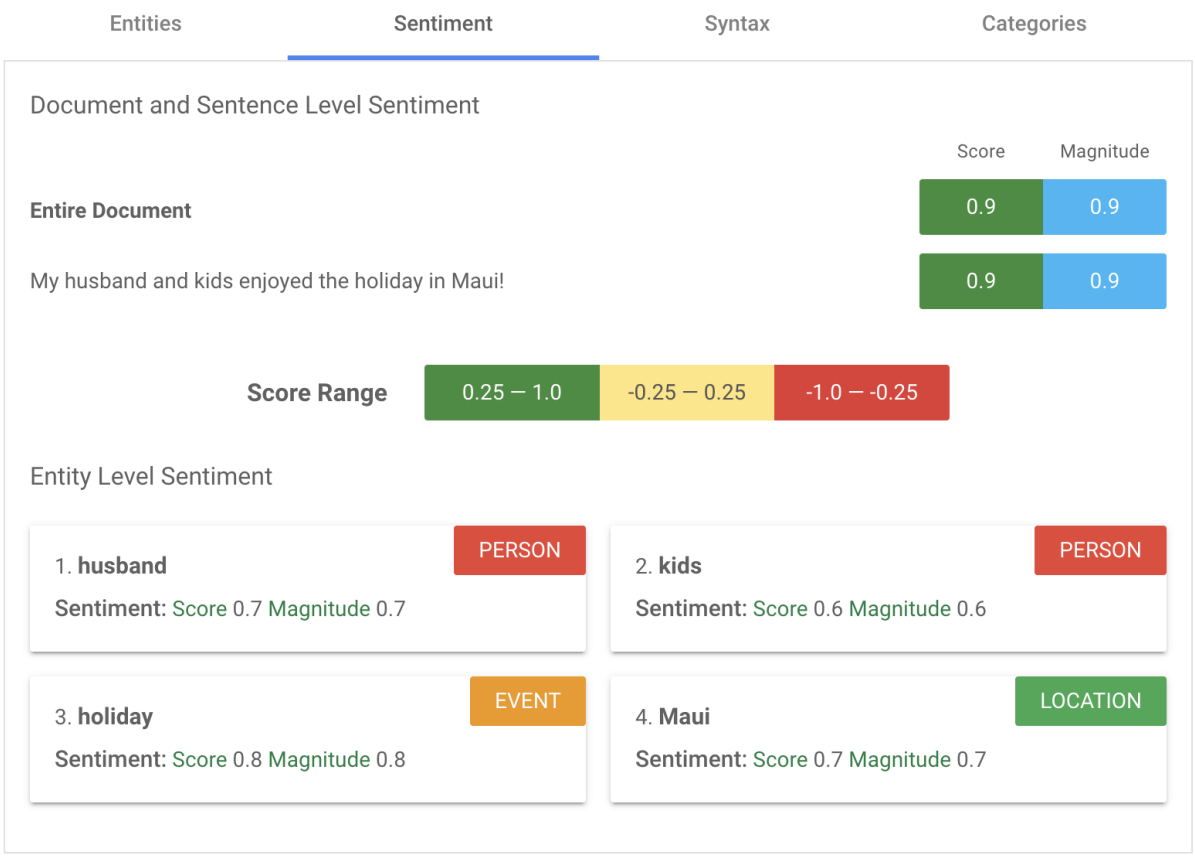
Sentiment Analysis

Google Cloud Natural Language API analyzes text and identifies its overall sentiment as positive, negative, or neutral. With the help of Google Cloud Natural Language API, you can quickly and easily analyze the sentiment of large volumes of text data, and gain valuable insights into customer opinions and preferences.

To analyze sentiment in a document, make a POST request to the documents:analyzeSentiment REST method and provide the appropriate request body. Here is an example:

```
Unset
curl -X POST \
     -H "Authorization: Bearer "$(gcloud auth application-default
print-access-token) \
     -H "Content-Type: application/json; charset=utf-8" \
     --data "{
                'encodingType': 'UTF8',
                'document': {
                  'type': 'PLAIN_TEXT',
                  'content': 'My husband and kids enjoyed the holiday in Maui!'
                }
              }" \
       "https://language.googleapis.com/v1/documents:analyzeSentiment"
```

If the request is successful, the server returns a 200 OK HTTP status code and the response in JSON format. It can then be parsed to enrich the original data or generate a visualization such as below.



For more information on sentiment analysis, please refer to the GCP documentation.

Entity Analysis

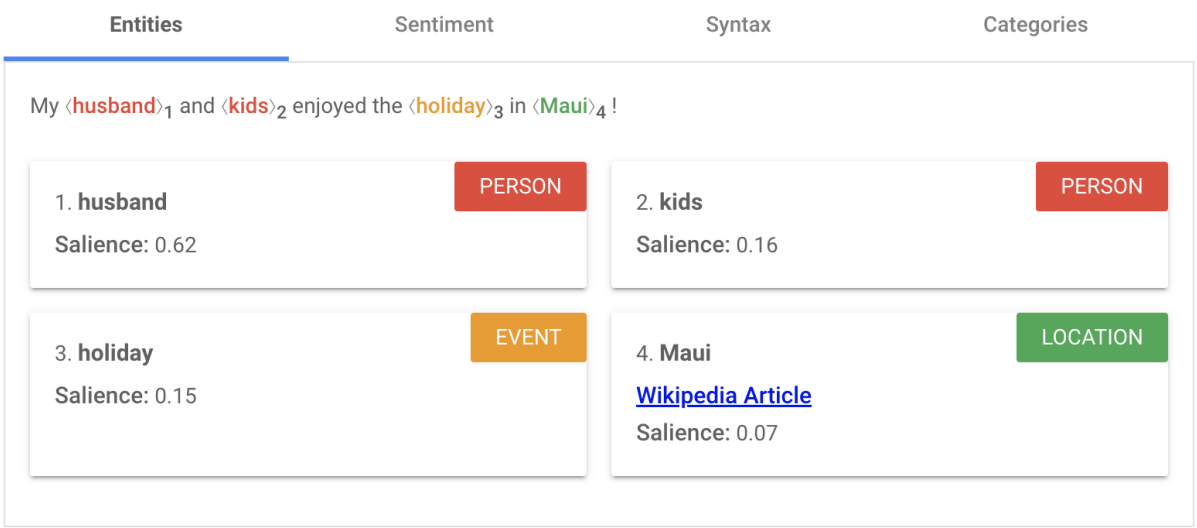
The Entity Analysis feature identifies and classifies entities within text. It provides information about the type of entity, and its relevance to the text. The API can also disambiguate entities with the same name and provide additional metadata.

To analyze entities in a document, make a POST request to the documents:analyzeEntities REST method and provide the appropriate request body. Here is an example.

```
Unset
curl -X POST \
```

```
     -H "Authorization: Bearer "$(gcloud auth application-default
print-access-token) \
     -H "Content-Type: application/json; charset=utf-8" \
     --data "{
                'encodingType': 'UTF8',
                'document': {
                  'type': 'PLAIN_TEXT',
                  'content': 'My husband and kids enjoyed the holiday in Maui!'
                }
              }" \
       "https://language.googleapis.com/v1/documents:analyzeEntities"
```

If the request is successful, the server returns a 200 OK HTTP status code and the response in JSON format. It can then be parsed to enrich the original data or generate a visualization such as below.



Salience refers to a measure of the importance or relevance of an entity within a piece of text. When the API analyzes a text, it can identify and extract entities, such as people, organizations, and locations, that are mentioned in the text. The salience score indicates how important or central each entity is to the overall meaning of the text.

Bronze to Silver Review Data

Let's take a look at the logic to process and enhance our raw review data.

```
Unset
CREATE OR REFRESH STREAMING LIVE TABLE silver_reviews_annotated (
 CONSTRAINT valid_business_id EXPECT (business_id IS NOT NULL) ON VIOLATION FAIL UPDATE,
 CONSTRAINT valid_user_id EXPECT (user_id IS NOT NULL) ON VIOLATION FAIL UPDATE,
 CONSTRAINT valid_stars_num EXPECT (stars BETWEEN 1 AND 5) ON VIOLATION DROP ROW,
 CONSTRAINT contains_review EXPECT (LENGTH(TRIM(text)) > 0)
)
COMMENT "Reviews: cleansed, data formats corrected, with Google Natural Language API
responses in a single column"
AS
SELECT CAST(date AS TIMESTAMP) AS date,
 business_id, review_id, user_id, stars,
 -- Apply Google Natural Language API to each record using the
 -- 'ANNOTATE_COMMENT()' user-defined function defined in dlt_init.py
 ANNOTATE_COMMENT(text) AS comment_analysis,
 cool, funny, useful, text
FROM STREAM(LIVE.bronze_reviews_raw)
```

Now let's break down these statements to understand what is happening under the hood.

Ensuring Quality Data

As we start to build out our silver data, we first add data quality constraints to the pipeline. With data quality checks, you have the flexibility to drop or continue processing bad rows, or fail the job completely. Reporting on data quality issues can be used to troubleshoot and rectify issues with upstream data sources.

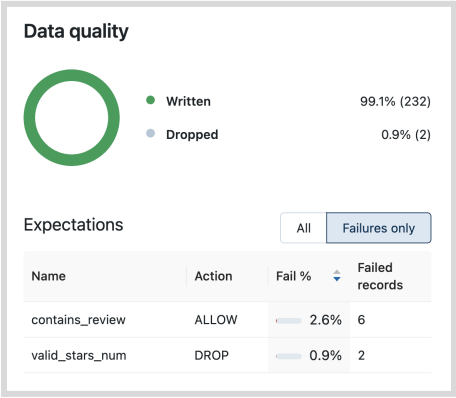
In our example we introduce the following data quality constraints:

- Continue processing, but flag this data, if reviews are empty: these are tolerable issues, and we can address the data source at a later date
- Drop rows if stars is not between 1 and 5 : these are rows we don't want in our final datasets, and we'll continue to process the rest of the new data
- Fail the pipeline if business or user IDs are missing: this reflects a major issue with the data, and we fail the job completely so that downstream tables are not polluted. Any

data processed up to this point is not persisted in the downstream tables. We can then follow-up with our upstream providers to fix the issues at their root.

```
Unset
CREATE OR REFRESH LIVE TABLE reviews_silver_raw (
 CONSTRAINT valid_business_id EXPECT (business_id IS NOT NULL) ON VIOLATION FAIL UPDATE,
 CONSTRAINT valid_user_id EXPECT (user_id IS NOT NULL) ON VIOLATION FAIL UPDATE,
 CONSTRAINT valid_stars_num EXPECT (stars BETWEEN 1 AND 5) ON VIOLATION DROP ROW,
 CONSTRAINT contains_review EXPECT (LENGTH(TRIM(text)) > 0)
)
```

When we run our Delta Live Tables pipelines, we get insight into these data quality checks, and can take proactive steps to remedy failures for subsequent runs. You can also programmatically query data quality from the pipeline's event log.



Enriching with the Natural Language API Results

Now we start utilizing the Natural Language API to generate and store metadata about our customers' comments through the ANNOTATE_COMMENT() function. This is a user defined function (UDF) to handle the interactions with the API. UDF execution on Databricks is parallelised for efficiency.

We use a Python UDF to handle our API calls and error handling. First, install the Natural Language API Python client library.

Unset

```
%pip install --upgrade google-cloud-language
```

Next, we declare our UDF. annotate_comment() does the following:

- Passes the provided comment string to the Natural Language API
- Requests an overall sentiment score for the comment (extract_document_sentiment)
- Requests extraction of all entities and a sentiment score for each entity (extract_entity_sentiment)

Unset

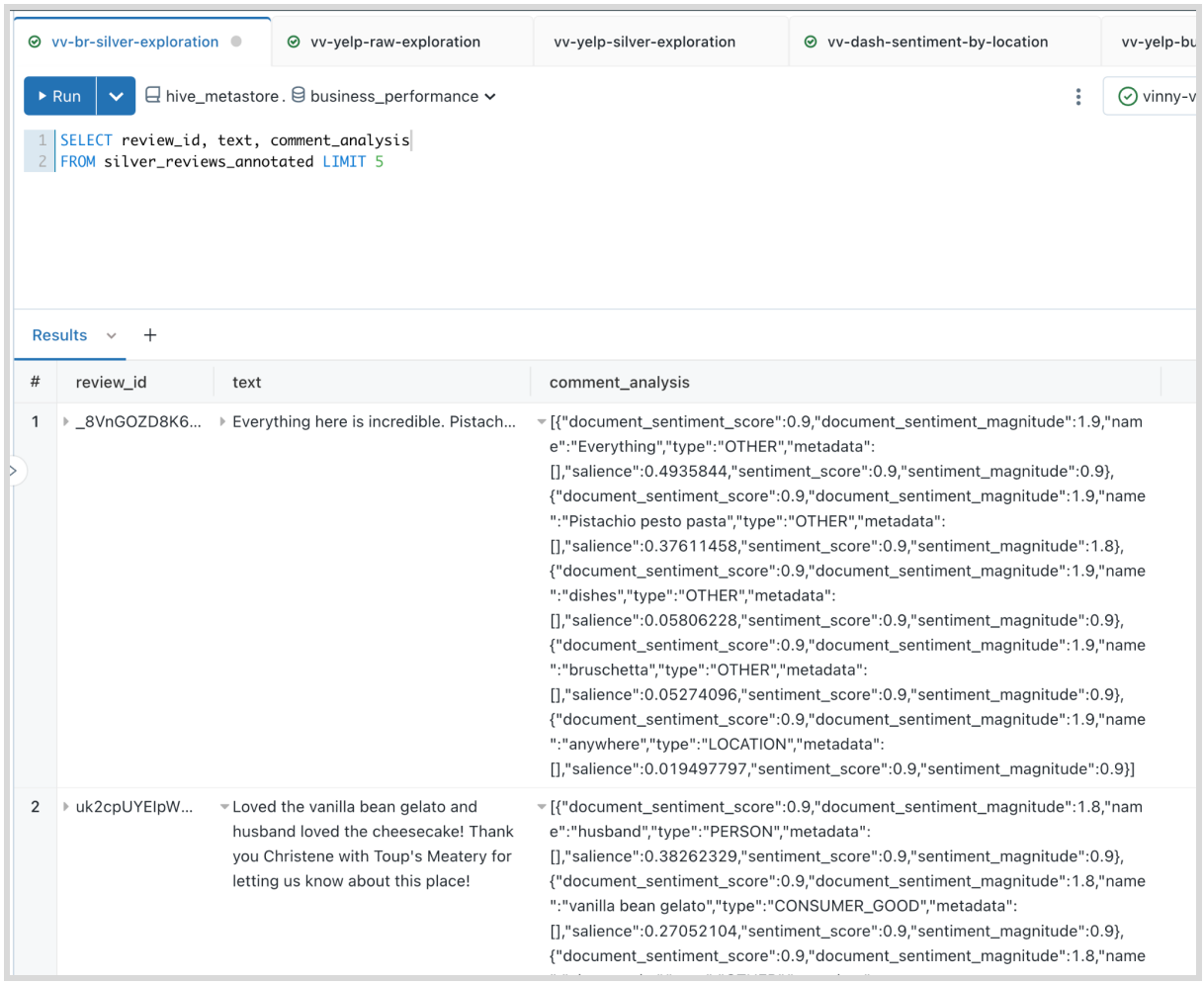
```
import dlt
from google.cloud import language_v1
from pyspark.sql.types import ArrayType, FloatType, MapType, StringType, StructField,
StructType



def annotate_comment(comment: str) -> list[dict]:
   """
   UDF to annotate a comment using Google Cloud Natural Language API.

   Args:
       comment: A string representing the comment to be annotated.

   Returns:
       A list of dictionaries containing annotation information for each entity found in
the comment.
       Each dictionary has the following keys:
           - score (float): The sentiment score of the entity.
           - magnitude (float): The sentiment magnitude of the entity.
           - name (str): The name of the entity.
           - type_ (str): The type of the entity.
           - metadata (list): A list of tuples representing the metadata associated with
the entity.
           - salience (float): The salience score of the entity.
```

```
            - sentiment_score (float): The sentiment score of the entity's sentiment.
            - sentiment_magnitude (float): The sentiment magnitude of the entity's
sentiment.
    """

    document = language_v1.Document(
        content=comment, type_=language_v1.Document.Type.PLAIN_TEXT
    )

    #
https://cloud.google.com/python/docs/reference/language/latest/google.cloud.language_v1.ty
pes.AnnotateTextRequest.Features
    features = {
        "extract_syntax": False,
        "extract_entities": False,
        "extract_document_sentiment": True,
        "extract_entity_sentiment": True,
        "classify_text": False,
    }

    try:
        client = language_v1.LanguageServiceClient()

        # Performs multiple analysis types (sentiment, entities) in one call
        response = client.annotate_text(
            request={"document": document, "features": features}
        )

        # Entity definition:
https://cloud.google.com/python/docs/reference/language/latest/google.cloud.language_v1.ty
pes.Entity
        scores = [
            (
                response.document_sentiment.score,
                response.document_sentiment.magnitude,
                entity.name,
                entity.type_.name,
                [(k, v) for k, v in entity.metadata.items()],
```

```
                entity.salience,
                entity.sentiment.score,
                entity.sentiment.magnitude,
            )
            for entity in response.entities
        ]

        return scores

    except Exception as e:
        return []


# Explicitly define schema to prevent data quality issues downstream
annotation_schema = StructType(
    [
        StructField("document_sentiment_score", FloatType(), True),
        StructField("document_sentiment_magnitude", FloatType(), True),
        StructField("name", StringType(), True),
        StructField("type", StringType(), True),
        StructField("metadata", ArrayType(MapType(StringType(), StringType(), True))),
        StructField("salience", FloatType(), True),
        StructField("sentiment_score", FloatType(), True),
        StructField("sentiment_magnitude", FloatType(), True),
    ]
)

# Register our UDF for usage in Delta Live Tables SQL
spark.udf.register("annotate_comment", annotate_comment, ArrayType(annotation_schema))
```

Further refining the silver table

We now have structured metadata to help bring meaning to our corpus of customer reviews!



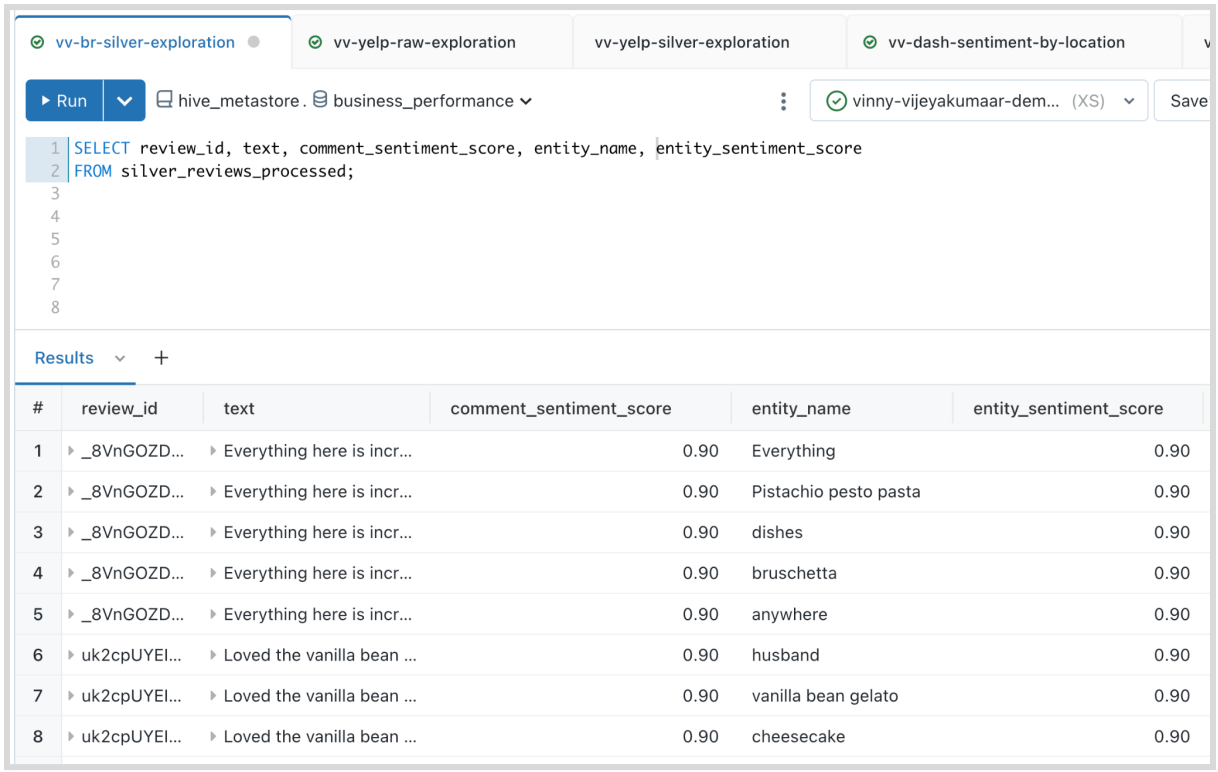
However, this form of data is not always easily queryable by BI tools. So let's create another table where we break out the data into a new row for each entity in a review, along with columns for each piece of metadata returned by the Natural Language API.

Unset

```
CREATE OR REFRESH LIVE TABLE silver_reviews_processed
COMMENT "Reviews exploded into rows and comment analysis attributes broken out into separate columns"
AS
WITH exploded_analyses AS (
 SELECT * EXCEPT (comment_analysis)
   -- Create a row for each entity analysis result per review
   , EXPLODE(comment_analysis) AS comment_analysis
 FROM live.silver_reviews_annotated
)
```

```
SELECT * EXCEPT(comment_analysis),
 comment_analysis.document_sentiment_score AS comment_sentiment_score,
 comment_analysis.document_sentiment_magnitude AS comment_sentiment_magnitude,
 comment_analysis.name AS entity_name,
 comment_analysis.metadata AS entity_metadata,
 comment_analysis.salience AS entity_salience,
 comment_analysis.sentiment_score AS entity_sentiment_score,
 comment_analysis.sentiment_magnitude AS entity_sentiment_magnitude
FROM exploded_analyses
```

That's much better!



5.3 Building out the Gold Layer

At this point we have high quality enriched data in our silver table. The next step is to create gold layer tables that are consumable by business users. They can query this data using

Databricks SQL, through BI Tools (e.g. Looker, ThoughtSpot), or programmatically with Databricks' APIs and SDKs.

We'll create three gold tables:

- gold_business_review_sentiment_summary: a summary of sentiment by time and business
- gold_sentiment_attributes_positive: provides insights to the marketing team on what attributes makes a successful business, and be able to promote them
- gold_sentiment_attributes_negative: provides insights to the product management team on which suppliers need improvement and what red flags to watch out for when onboarding new suppliers

```
Unset
CREATE OR REFRESH LIVE TABLE gold_business_review_sentiment_summary (
 CONSTRAINT valid_business_name EXPECT (name IS NOT NULL)
)
COMMENT "Summary of reviews & sentiments by business"
SELECT r.date, r.business_id, b.name, r.review_id, r.user_id,
 MAX(r.comment_sentiment_score) AS review_sentiment_score,
 MAX(r.comment_sentiment_magnitude) AS review_sentiment_magnitude,
 r.stars, b.latitude, b.longitude, b.state
FROM LIVE.silver_reviews_processed r
LEFT JOIN LIVE.silver_businesses_processed b
 ON r.business_id = b.business_id
GROUP BY r.date, r.business_id, b.name, r.review_id, r.user_id, r.stars, b.latitude,
b.longitude, b.state
```

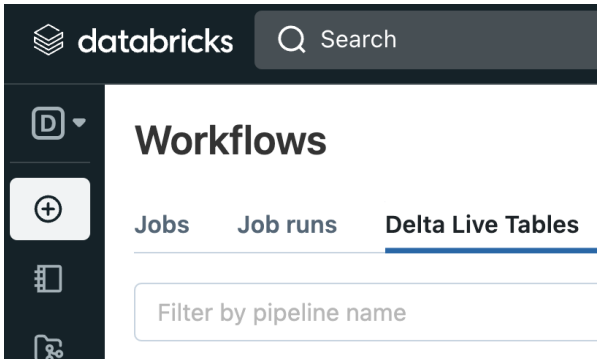
```
Unset
CREATE OR REFRESH LIVE TABLE gold_sentiment_attributes_positive
COMMENT "Positive entity attributes where entity sentiment >= 0.6 and salience >= 0.5"
SELECT
 DATE(r.date) AS date,
 r.business_id, b.name, b.review_count, b.stars AS business_stars, b.categories,
 r.stars AS review_stars, r.entity_name, r.entity_salience, r.entity_sentiment_score,
 r.entity_sentiment_magnitude
```

```
FROM LIVE.silver_reviews_processed r
INNER JOIN LIVE.silver_businesses_processed b ON r.business_id = b.business_id
WHERE entity_sentiment_score >= 0.6
 AND entity_salience >= 0.5
```

6. Executing the Delta Live Tables Pipeline

Databricks Workflows is utilized to manage and execute our Delta Live Tables pipeline. Now that our pipeline is ready, let's configure it and schedule it to run every 30 minutes.

- Go to Workflows > Delta Live Tables



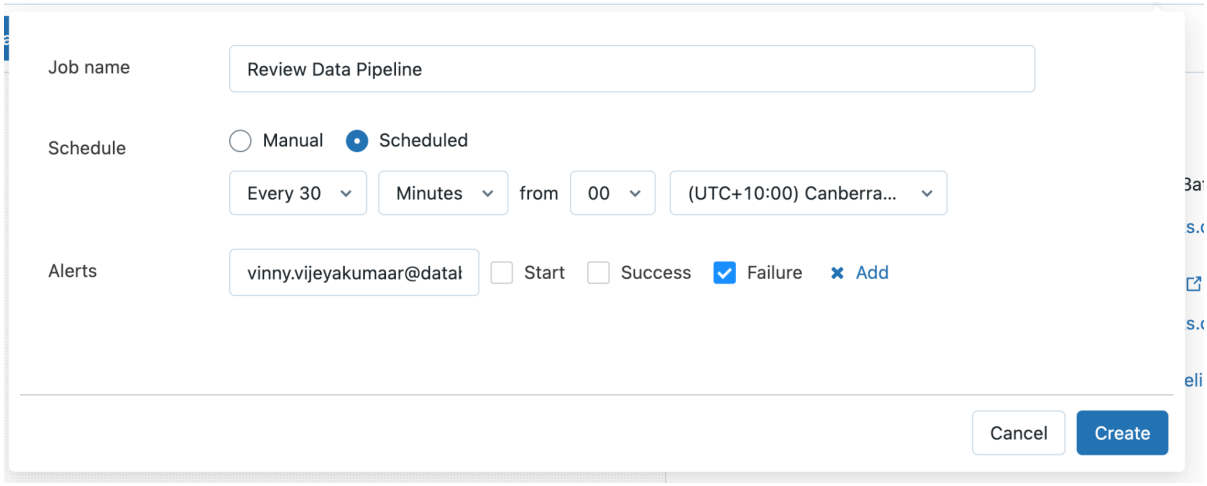
- Click Create Pipeline
- Give your pipeline a name
- Set Pipeline mode to Triggered so that we only run this on a per-needed basis. If our data source is a real-time streaming data source we would set this to Continuous
- In Source code add the paths to your code that make up the Delta Live Tables pipeline. In this example we have two notebooks:
    - dlt_init: contains the definition of our UDF
    - dlt_pipeline: contains our SQL transformations
- In Target schema provide the target database for your materialized tables. For our example we use business_performance

- Switch on Photon Acceleration to ensure that your jobs run in the shortest time possible
- You can also add Cluster tags to your Delta Live Tables pipeline execution. Tags are useful for attributing consumption usage data to your jobs. In our example we tag the business owner, department, and cost center

business-owner	jennyhoward
department	business_performance
cost-center	995

- Next we need to add the Google Cloud service account we created for accessing the storage buckets and Natural Language API. Click on JSON in the upper right corner

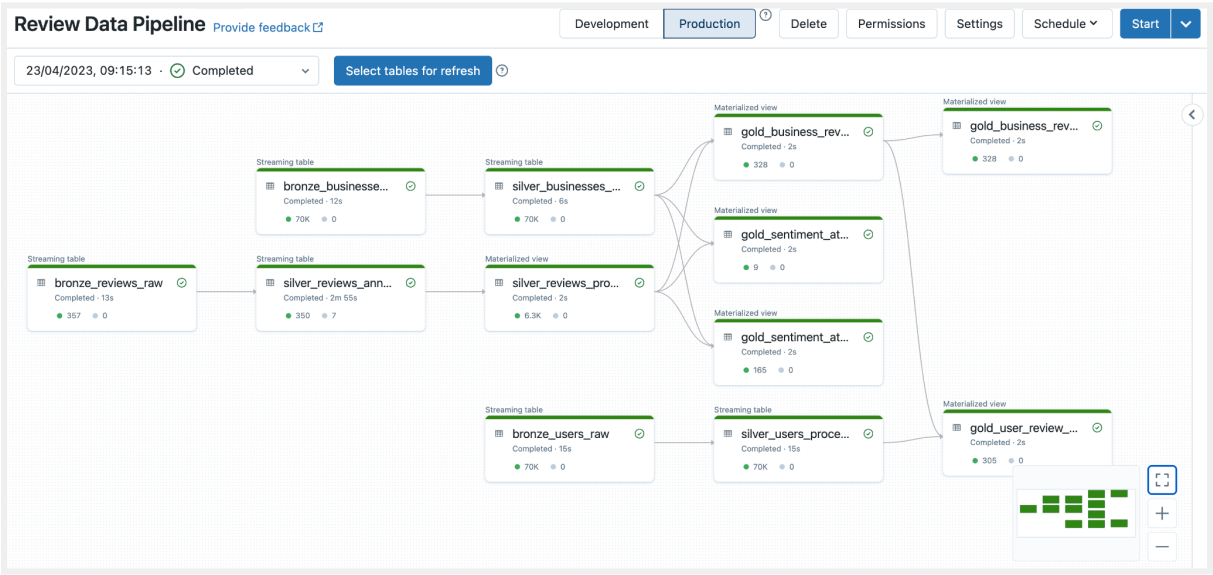
- Add the following definition to the default and maintenance clusters, making sure to reference the service account that you created earlier
  "gcp_attributes": {"google_service_account": "service-account-name@your-project.iam.gserviceaccount.com"}
- Click on Save
- Next, we set a schedule for our Pipeline. Click on Schedule > Add a schedule
- We'll set our Delta Live Tables pipeline to run every 30 minutes



- Click on Create

We now have a Delta Live Tables that's ready to run! To test it out, click on Start.

A completed pipeline run will look like this:



6.1 What is happening under the hood

Delta Live Tables takes care of all the things you would usually worry about when defining your own workflows:

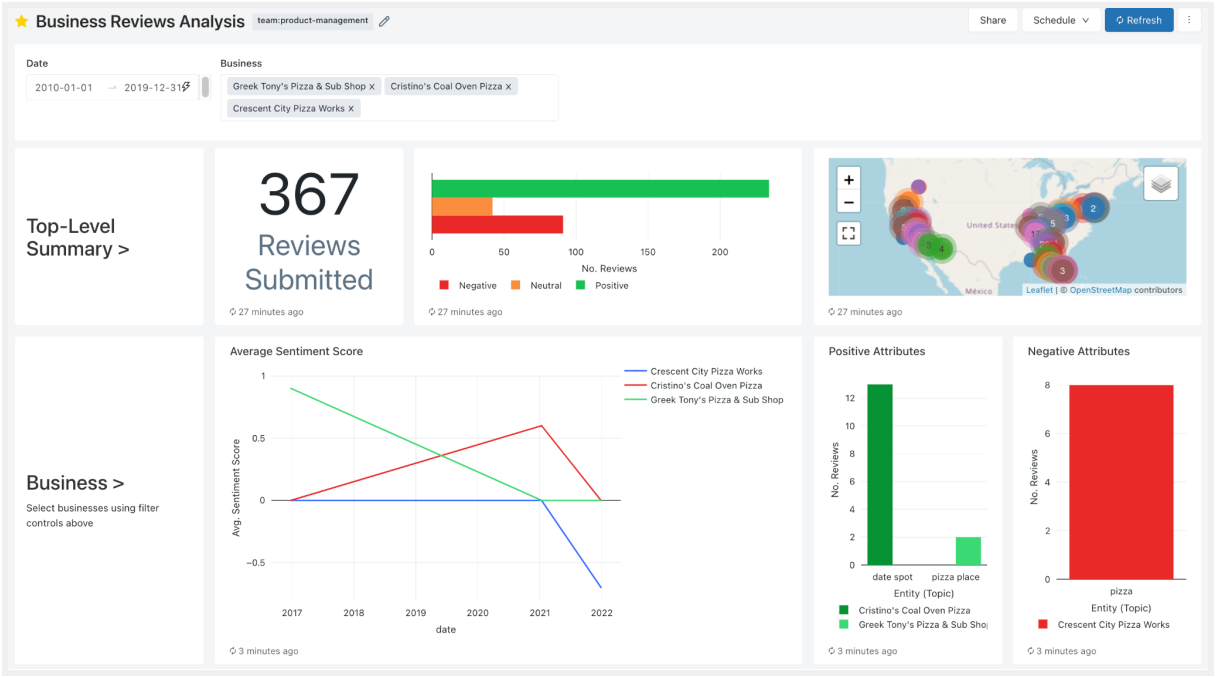
- Automatically takes care of authentication with Google's services using the provided Service Account
- Management of compute
    - Spins up the necessary clusters with the best determined configuration
    - Autoscales clusters up and down based on the complexity of each task. Autoscaling is also useful in real-time streaming ingestion scenarios when there are peak loads
- Data management
    - Processes your data and stores them as Delta tables, thus giving you access to features such as time travel, change data feeds, and more
    - Enforces data quality controls on all data as it moves through each stage of your pipeline

    - Handles the logic of writing incremental new data to downstream tables including SCD Type 1 and 2 support
    - Handles schema evolution
- Performs daily maintenance tasks on your tables to ensure optimal performance for analytics

7. Analyzing Gold Layer Data

Now that we have our gold table, analysts can query that data using Databricks SQL, by connecting BI Tools (e.g. Looker, Tableau, ThoughtSpot), or programmatically with Databricks' APIs and SDKs.

For analysts that operate primarily inside of Databricks, we can create Dashboards for them utilizing a rich variety of visualizations and filters.



No matter the form of your data, you can easily create value and generate insights from that data thanks to the combination of Databricks' Delta Live Tables and Google Cloud's ML APIs. Delta Live Tables removes the burden of writing and managing complex pipelines while ensuring data quality for your business users. Google Cloud's ML APIs allows you to instantly apply ML to your data without needing to train and manage your own models.

8. Further Materials

If you would like to find out more about leveraging the best of Databricks on Google Cloud, check out

- Getting started documentation
- Google AI Services
- Databricks on Google Marketplace
- dbdemos.ai: comprehensive demos covering Databricks' full range of capabilities. Directly install them in your Databricks Workspace or view the Notebooks online
- Solution Accelerators: end-to-end comprehensive code solving large-scale enterprise problems across industries

Authors

- Yang Li - Staff Cloud Solutions Architect, Google Cloud
- Vinny Vijeyakumaar - Senior Solutions Architect, Retail, Databricks