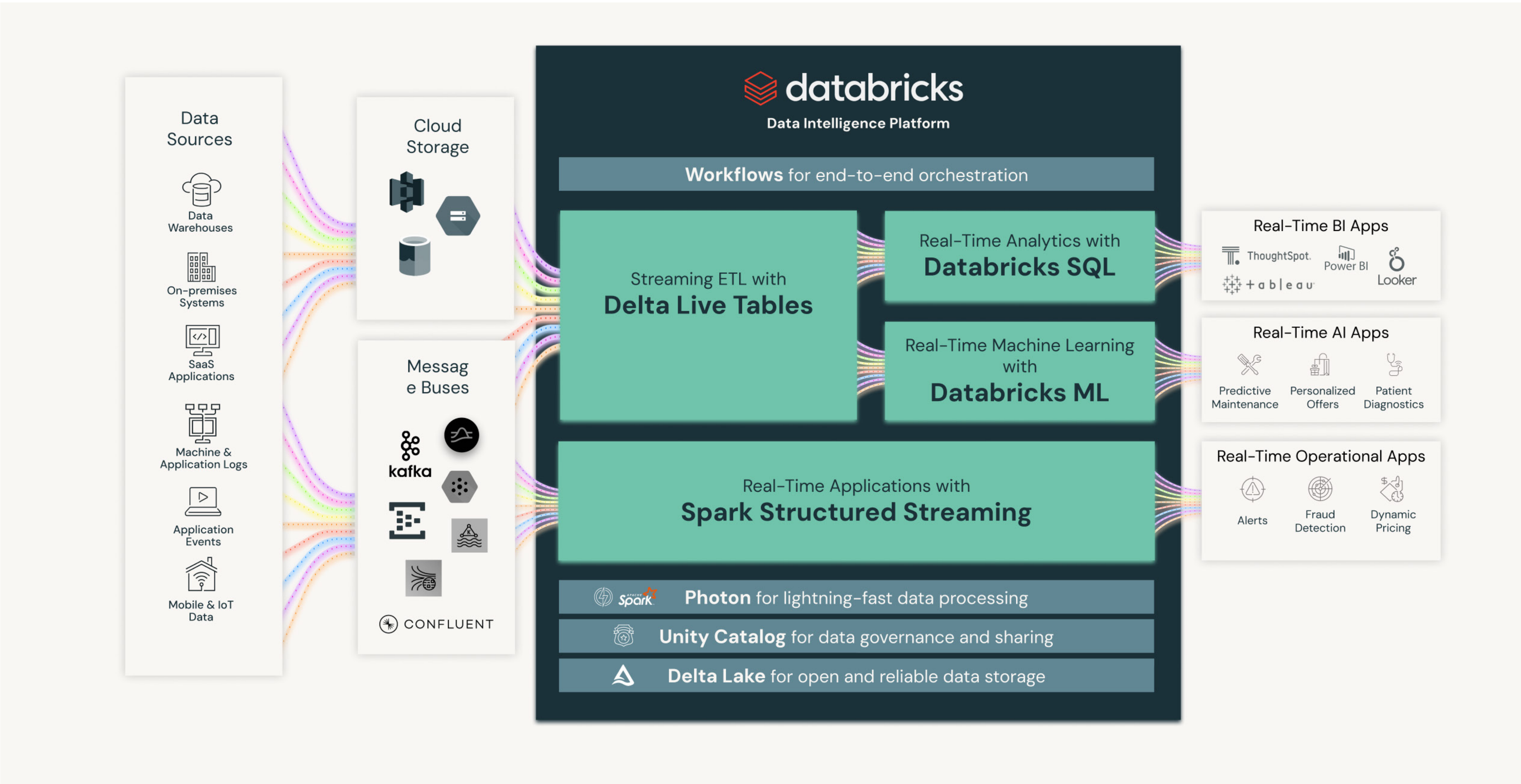
Why data engineers choose the Data Intelligence Platform

So how does the Data Intelligence Platform help with each of the data engineering challenges discussed earlier?

- **Efficient ingestion, wide range of data connectors:** Databricks allows you to efficiently ingest data, only bringing in new data or table updates. With a growing set of native connectors for popular data sources, as well as a broad network of data ingestion partners, you can easily move data from siloed systems into your data platform. Ingesting and storing your data in Delta Lake while leveraging the reliability and scalability of the Data Intelligence Platform is the first step in extracting value from your data and accelerating innovation.
- **Real-time data stream processing:** The Data Intelligence Platform simplifies development and operations by automating the production aspects associated with building and maintaining real-time data workloads. Delta Live Tables provides a declarative way to define streaming ETL pipelines, and Spark Structured Streaming helps build real-time applications for real-time decision-making (see diagram on next page).

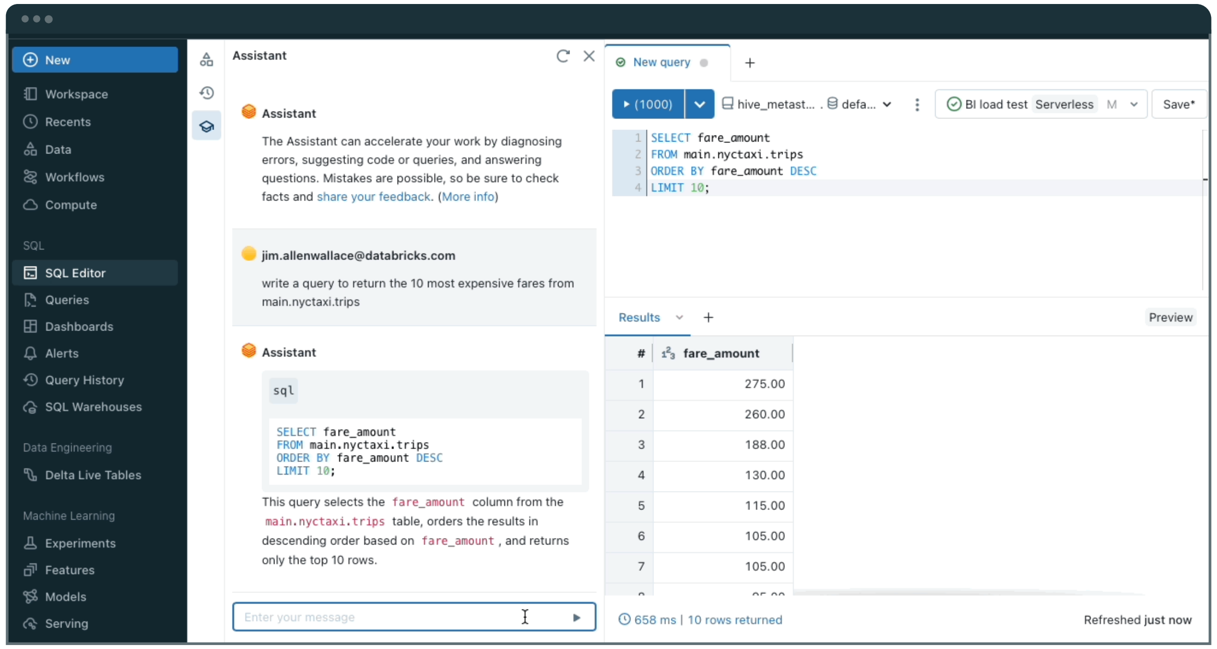


A unified set of tools for real-time data processing

- **Reliable data pipelines at scale:** Both Delta Live Tables and Databricks Workflows use smart autoscaling and auto-optimized resource management to handle high-scaled workloads. With lakehouse architecture, the high scalability of data lakes is combined with the high reliability of data warehouses, thanks to Delta Lake — the storage format that sits at the foundation of the lakehouse.
- **Data quality:** High reliability — starting at the storage level with Delta Lake and coupled with data quality–specific features offered by Delta Live Tables — ensures high data quality. These features include setting data “expectations” to handle corrupt or missing data as well as automatic retries. In addition, both Databricks Workflows and Delta Live Tables provide full observability to data engineers, making issue resolution faster and easier.
- **Unified governance with secure data sharing:** Unity Catalog provides a single governance model for the entire platform so every dataset and pipeline are governed in a consistent way. Datasets are discoverable and can be securely shared with internal or external teams using Delta Sharing. In addition, because Unity Catalog is a cross-platform governance solution, it provides valuable lineage information, so it’s easy to have a full understanding of how each dataset and table is used downstream and where it originates upstream.

In addition, data engineers using the Data Intelligence Platform benefit from cutting-edge innovations in the form of GenAI–infused intelligence:

- **AI-powered productivity:** Specifically useful for data engineers, the Databricks Assistant is a context-aware AI assistant that offers a conversational API to query data, generate code, explain code queries and even fix issues.



Conclusion

As organizations strive to innovate with AI, data engineering is a focal point for success by delivering reliable, real-time data pipelines that make AI possible. With the Databricks Platform, built on a lakehouse architecture and powered by Data Intelligence, data engineers are set up for success in dealing with the critical challenges posed in the modern data landscape. By leaning on the advanced capabilities of the Data Intelligence Platform, data engineers don't need to spend as much time managing complex pipelines or dealing with reliability, scalability and data quality issues. Instead, they can focus on innovation and bringing more value to the organization.

FOLLOW PROVEN BEST PRACTICES

In the next section, we describe best practices for data engineering and end-to-end use cases drawn from real-world examples. From data ingestion and real-time processing to orchestration and data federation, you'll learn how to apply proven patterns and make the best use of the different capabilities of the Data Intelligence Platform.

As you explore the rest of this guide, you can find datasets and code samples in the various Databricks Solution Accelerators, so you can get your hands dirty and start building on the Data Intelligence Platform.

02 Guidance and Best Practices

databricks

Databricks Assistant Tips and Tricks for Data Engineers

by Jackie Zhang, Rafi Kurlansik and Richard Tomlinson

The generative AI revolution is transforming the way that teams work, and Databricks Assistant leverages the best of these advancements. It allows you to query data through a conversational interface, making you more productive inside your Databricks Workspace. The Assistant is powered by DatabricksIQ, the Data Intelligence Engine for Databricks, helping to ensure your data is secured and responses are accurate and tailored to the specifics of your enterprise. Databricks Assistant lets you describe your task in natural language to generate, optimize, or debug complex code without interrupting your developer experience.

In this chapter we'll discuss how to get the most out of your Databricks Assistant and focus on how the Assistant can improve the life of Data Engineers by eliminating tedium, increasing productivity and immersion, and accelerating time to value. We will follow up with a series of posts focused on different data practitioner personas, so stay tuned for upcoming entries focused on data scientists, SQL analysts, and more.

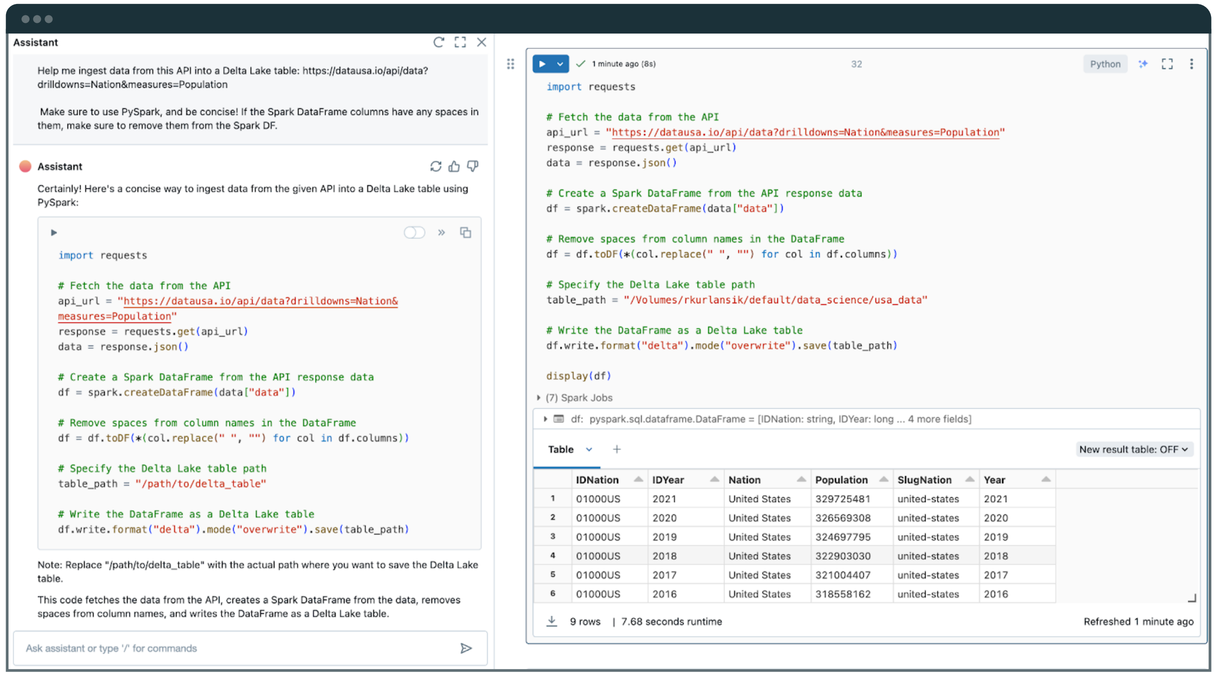
INGESTION

When working with Databricks as a data engineer, ingesting data into Delta Lake tables is often the first step. Let's take a look at two examples of how the Assistant helps load data, one from APIs, and one from files in cloud storage. For each, we will share the prompt and results. As mentioned in the 5 tips blog, being specific in a prompt gives the best results, a technique consistently used in this article.

To get data from the datausa.io API and load it into a Delta Lake table with Python, we used the following prompt:

Help me ingest data from this API into a Delta Lake table: https://datausa.io/api/data?drilldowns=Nation&measures=Population

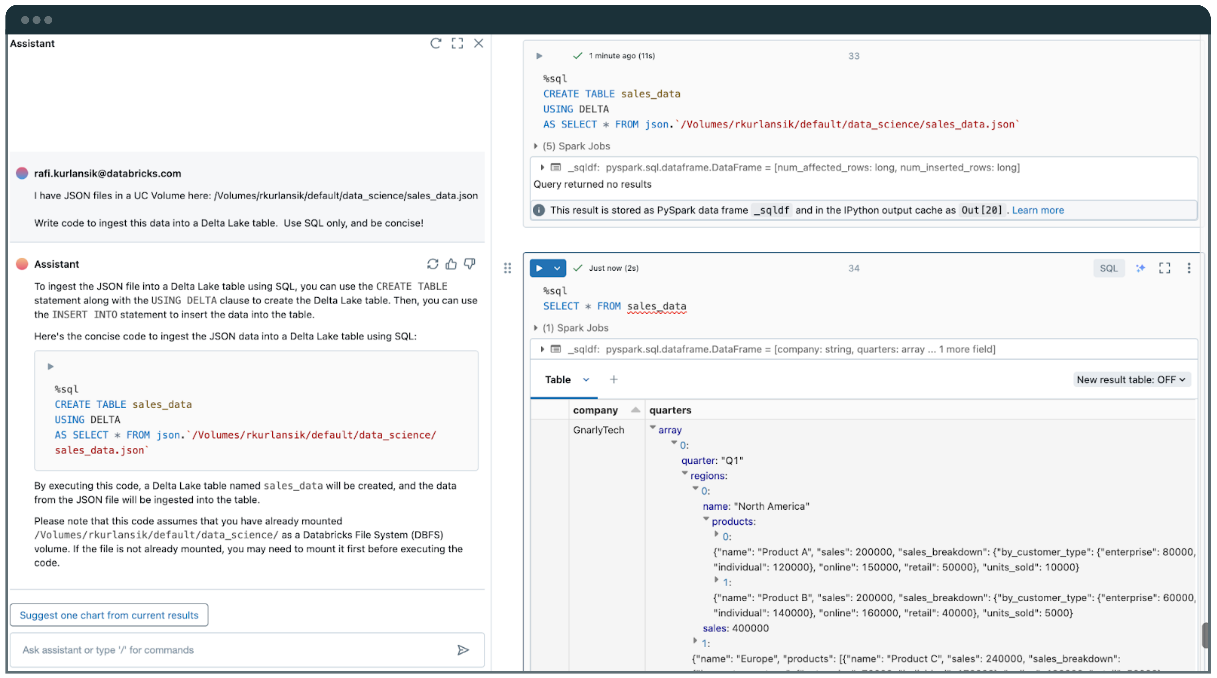
Make sure to use PySpark, and be concise! If the Spark DataFrame columns have any spaces in them, make sure to remove them from the Spark DF.



A similar prompt can be used to ingest JSON files from cloud storage into Delta Lake tables, this time using SQL:

I have JSON files in a UC Volume here: /Volumes/rkurlansik/default/data_science/sales_data.json

Write code to ingest this data into a Delta Lake table. Use SQL only, and be concise!



TRANSFORMING DATA FROM UNSTRUCTURED TO STRUCTURED

Following tidy data principles, any given cell of a table should contain a single observation with a proper data type. Complex strings or nested data structures are often at odds with this principle, and as a result, data engineering work consists of extracting structured data from unstructured data. Let's explore two examples where the Assistant excels at this task — using regular expressions and exploding nested data structures.

Regular expressions

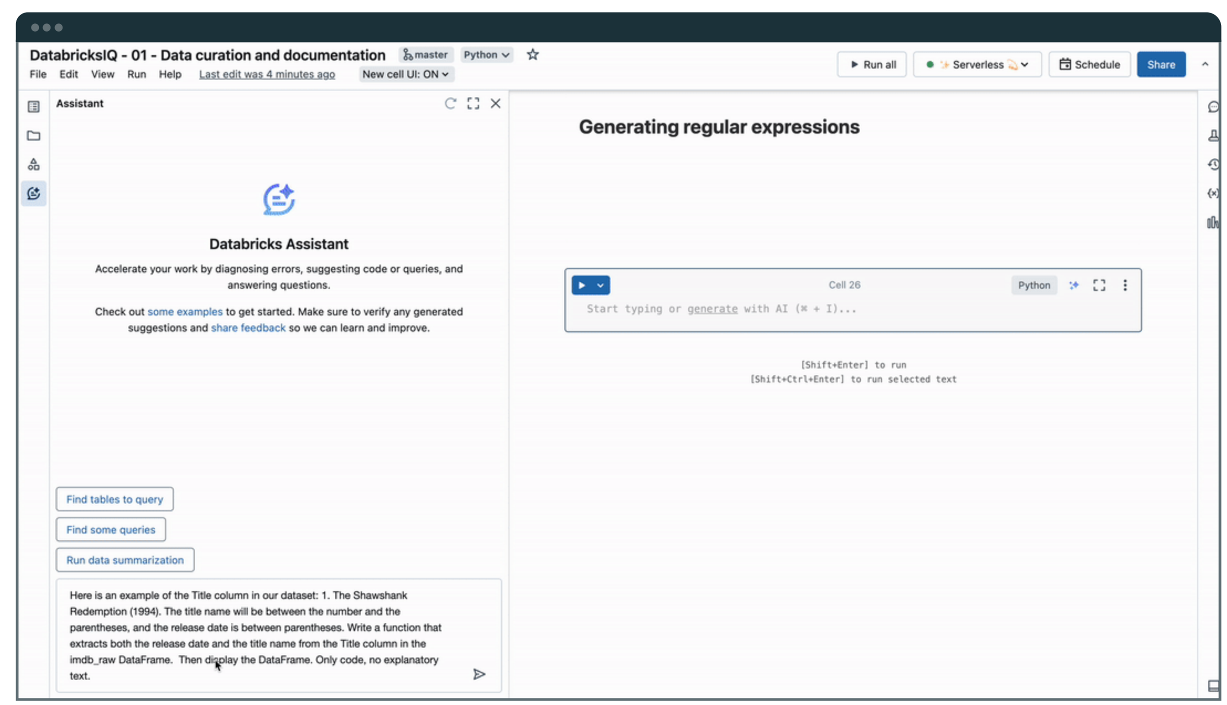
Regular expressions are a means to extract structured data from messy strings, but figuring out the correct regex takes time and is tedious. In this respect, the Assistant is a boon for all data engineers who struggle with regex.

Consider this example using the *Title* column from the IMDb dataset:

Title	Certificate	Duration	Genre
1. The Shawshank Redemption (1994)	R	142 min	Drama

This column contains two distinct observations — film title and release year. With the following prompt, the Assistant identifies an appropriate regular expression to parse the string into multiple columns.

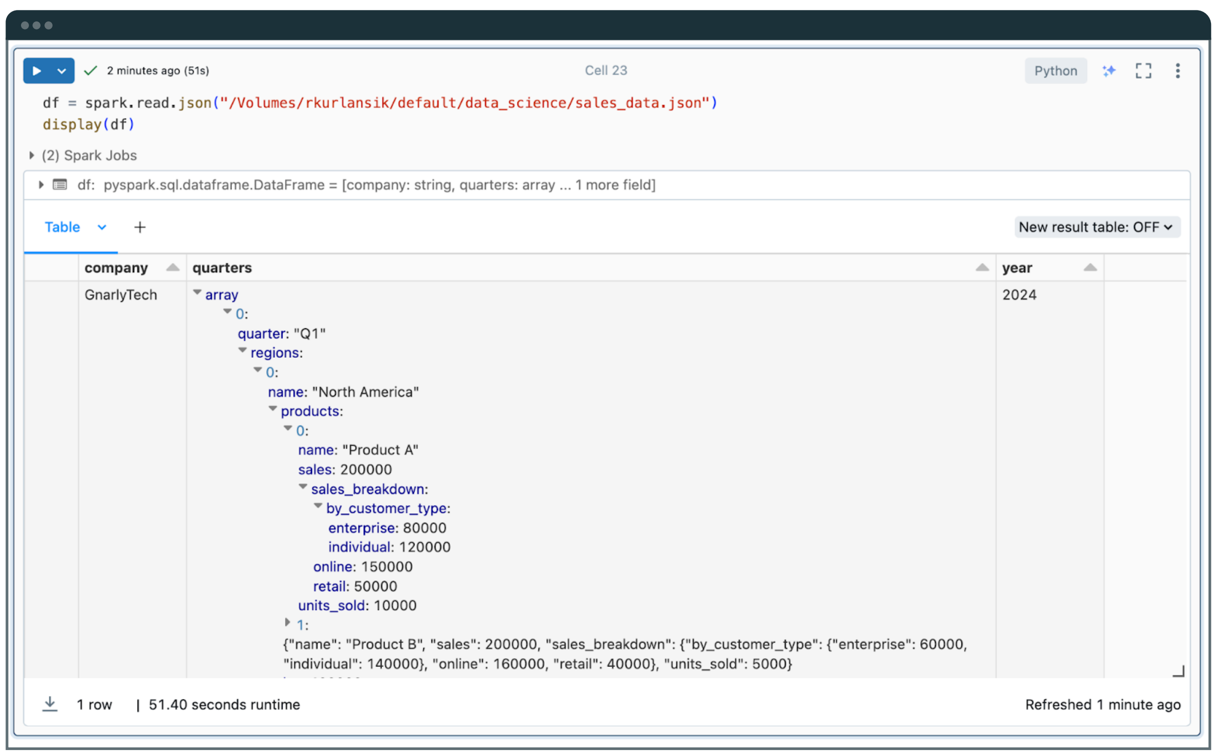
Here is an example of the Title column in our dataset: 1. The Shawshank Redemption (1994). The title name will be between the number and the parentheses, and the release date is between parentheses. Write a function that extracts both the release date and the title name from the Title column in the imdb_raw DataFrame.



Providing an example of the string in our prompt helps the Assistant find the correct result. If you are working with sensitive data, we recommend creating a fake example that follows the same pattern. In any case, now you have one less problem to worry about in your data engineering work.

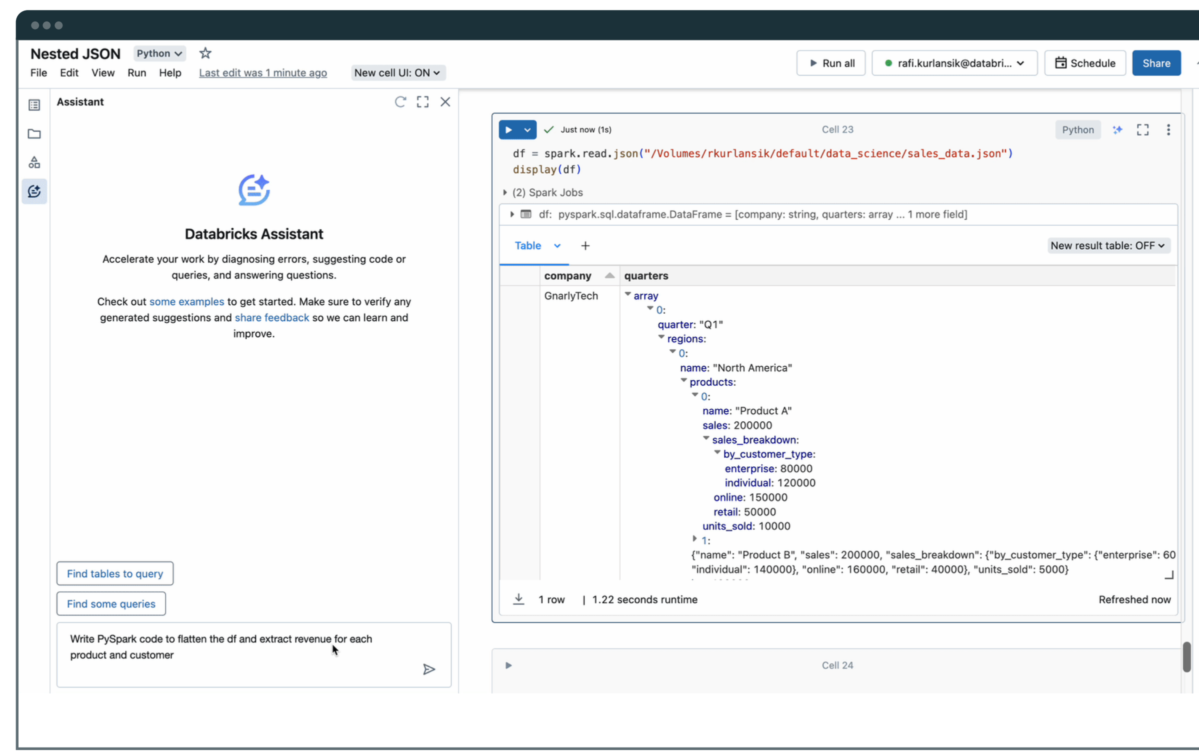
Nested Structs, Arrays (JSON, XML, etc)

When ingesting data via API, JSON files in storage, or noSQL databases, the resulting Spark DataFrames can be deeply nested and tricky to flatten correctly. Take a look at this mock sales data in JSON format:



Data engineers may be asked to flatten the nested array and extract revenue metrics for each product. Normally this task would take significant trial and error — even in a case where the data is relatively straightforward. The Assistant, however, being context-aware of the schemas of DataFrames you have in memory, generates code to get the job done. Using a simple prompt, we get the results we are looking for in seconds.

Write PySpark code to flatten the df and extract revenue for each product and customer



REFACTORING, DEBUGGING AND OPTIMIZATION

Another scenario data engineers face is rewriting code authored by other team members, either ones that may be more junior or have left the company. In these cases, the Assistant can analyze and explain poorly written code by understanding its context and intent. It can suggest more efficient algorithms, refactor code for better readability, and add comments.

Improving documentation and maintainability

This Python code calculates the total cost of items in an online shopping cart.

```
def calculate_total(cart_items):
    total = 0
    for i in range(len(cart_items)):
        if cart_items[i]['type'] == 'book':
            discount = cart_items[i]['price'] * 0.05
            total += cart_items[i]['price'] - discount
        else:
            total += cart_items[i]['price']
    return total

cart_items = [{'name': 'Python Programming', 'type': 'book', 'price': 50},
              {'name': 'Laptop', 'type': 'electronics', 'price': 800}]
total_price = calculate_total(cart_items)
```

The use of conditional blocks in this code makes it hard to read and inefficient at scale. Furthermore, there are no comments to explain what is happening. A good place to begin is to ask the Assistant to explain the code step by step. Once the data engineer understands the code, the Assistant can transform it, making it more performant and readable with the following prompt:

Rewrite this code in a way that is more performant, commented properly, and documented according to Python function documentation standards

The generated example below properly documents the code, and uses generator expressions instead of conditional blocks to improve memory utilization on larger datasets.

```
def calculate_total(cart_items):
    """
    Calculate the total price of items in the shopping cart.

    Apply a 5% discount to book items. The total is calculated by summing
    the prices, taking into account the discount for books.

    Parameters:
    cart_items (list of dict): A list of dictionaries, where each dictionary
                               represents an item in the cart with keys 'name',
                               'type', and 'price'.

    Returns:
    float: The total price of all items in the cart after discounts.
    """
    # Use a generator expression to iterate over cart items, applying a 5% discount
to books.
    return sum(item['price'] * 0.95 if item['type'] == 'book' else item['price'] for
item in cart_items)

# Example usage
cart_items = [{'name': 'Python Programming', 'type': 'book', 'price': 50},
              {'name': 'Laptop', 'type': 'electronics', 'price': 800}]
total_price = calculate_total(cart_items)
```