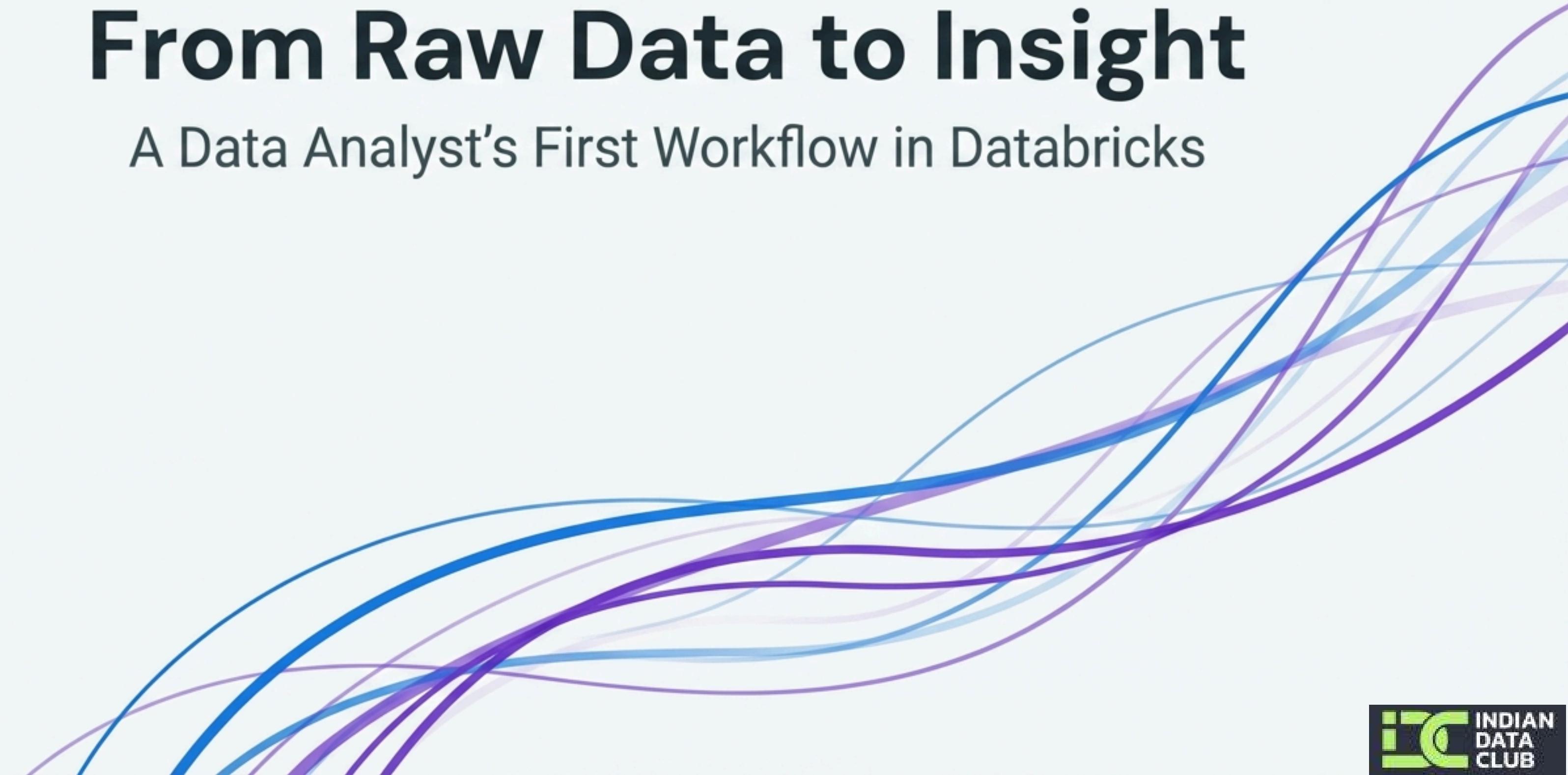


# From Raw Data to Insight

A Data Analyst's First Workflow in Databricks

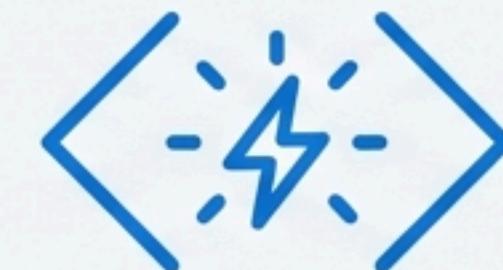


# The Objective: Analyze a New E-commerce Dataset

Our goal is to perform an initial exploration of a large e-commerce dataset. We need to locate the source files, load the data into a workable format, and ask a basic analytical question to verify our setup. The entire process will take place within the Databricks Data Intelligence Platform.



Discover Data



Load & Process



Query & Analyze

# Step 1: Discovering Our Data in the Unity Catalog

The screenshot shows the Databricks interface with the Unity Catalog selected in the sidebar. The main view displays the details of the 'ecommerce\_data' asset under the 'ecommerce' workspace. The asset overview shows a volume path of '/Volume/workspace/ecommerce/ecommerce\_data'. Below this, a file listing shows two large CSV files: '2019-Nov.csv' (8.28 GB) and '2019-Oct.csv' (5.28 GB), both modified 1 day ago. The sidebar also lists other workspaces like 'default' and 'samples'.

Before we can analyze anything, we need to find our data. The Unity Catalog provides a single, organized view of all our data assets.

We navigate a familiar folder structure to find our project files.

Our source files are substantial, totaling over 13 GB. We need a tool that can handle this scale.

# Our Workspace: The Interactive Databricks Notebook

The Databricks Notebook is our canvas. It's a multi-language, interactive environment where we can write and execute code, see results, and document our findings all in one place. Here, we'll use Python with Spark to process our data.



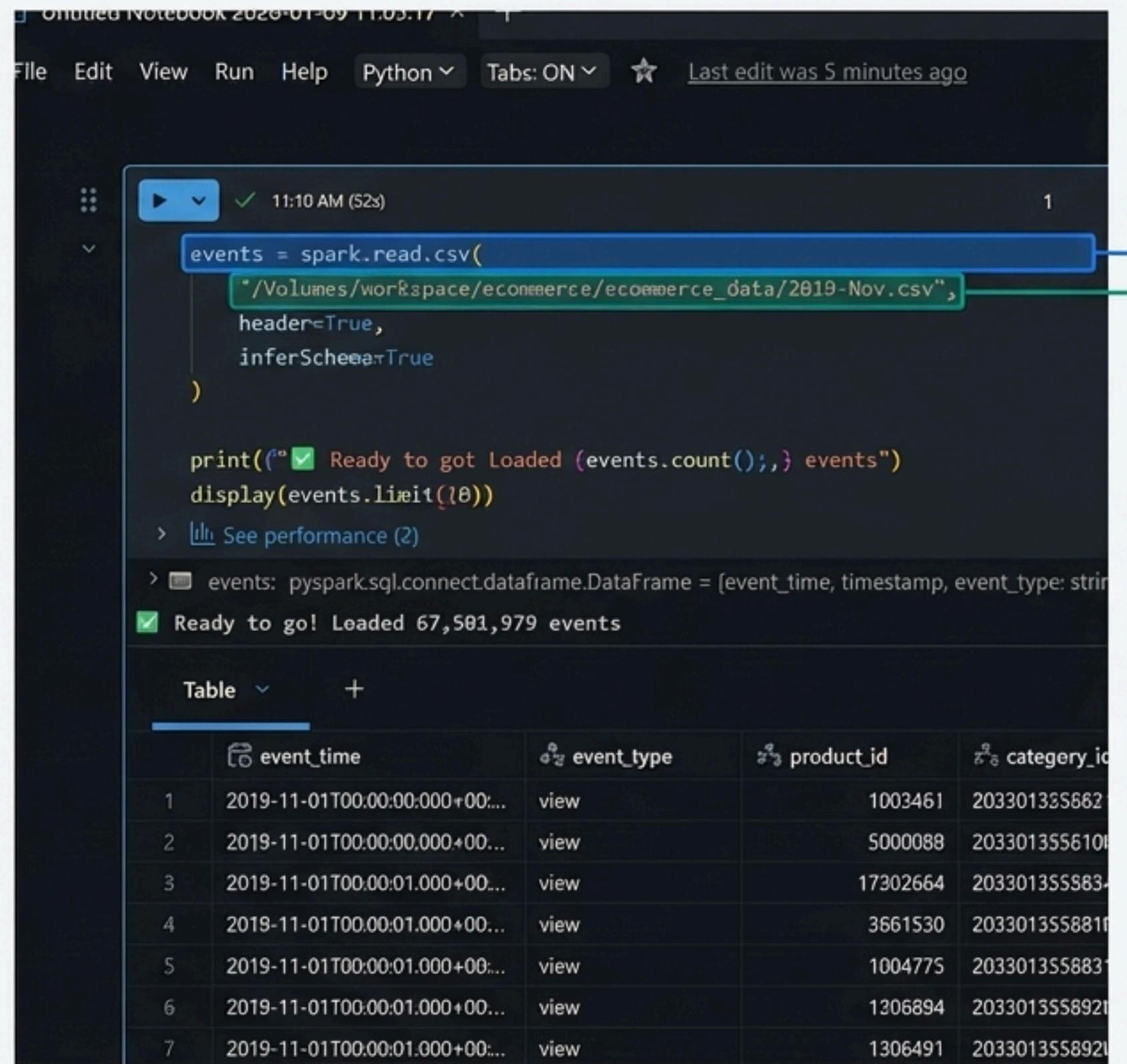
A screenshot of a Databricks Notebook interface. The top part shows a code cell with the following Python code:

```
```python
# Load e-commerce data
df = spark.read.format('csv').load('path/to/data')

# Display data summary
df.groupBy('category').count().display()
```
```

The bottom part displays a bar chart with three categories: Electronics, Clothing, and Home. The bars are colored blue, purple, and green respectively, corresponding to their count values.

| Category    | Count               |
|-------------|---------------------|
| Electronics | High (approx. 100)  |
| Clothing    | Medium (approx. 30) |
| Home        | Low (approx. 10)    |



The screenshot shows a Jupyter Notebook interface with a single code cell. The code uses the `spark.read.csv` method to load a CSV file from a local volume. The command includes parameters `header=True` and `inferSchema=True`. After running the cell, the notebook displays a summary message indicating 67,501,979 events were loaded. Below the code cell, a preview of the data is shown in a table format with columns: event\_time, event\_type, product\_id, and category\_id. The first seven rows of data are listed.

|   | event_time                    | event_type | product_id | category_id   |
|---|-------------------------------|------------|------------|---------------|
| 1 | 2019-11-01T00:00:00:000+00:00 | view       | 1003461    | 20330132S662  |
| 2 | 2019-11-01T00:00:00.000+00:00 | view       | 5000088    | 2033013556101 |
| 3 | 2019-11-01T00:01:000+00:00    | view       | 17302664   | 20330135583-  |
| 4 | 2019-11-01T00:01:000+00:00    | view       | 3661530    | 2033013558810 |
| 5 | 2019-11-01T00:01:000+00:00    | view       | 1004775    | 2033013558831 |
| 6 | 2019-11-01T00:01:000+00:00    | view       | 1306894    | 2033013558921 |
| 7 | 2019-11-01T00:01:000+00:00    | view       | 1306491    | 2033013558921 |

## Step 2: Bringing 67 Million Rows to Life

The core command for loading data.

With a single command, `spark.read.csv`, we can ingest the entire 8.39 GB November dataset. We instruct Spark to use the first row as the header and to automatically infer the data types for each column (`header=True, inferSchema=True`).

We use the direct path we found in the Unity Catalog.

# From One Command to a Fully Loaded DataFrame

The screenshot shows a Databricks notebook interface. On the left is the sidebar with various workspace options like Name, Workspace, Recents, Catalog, etc. The main area shows a notebook titled "Untitled Notebook 2026-01-09 11:03:17". A single code cell is running:

```
events = spark.read.csv(  
    "/Volumes/.../ecommerce_data/2019-Nov.csv",  
    header=True,  
    inferSchema=True  
)  
  
print(f"Ready to go! Loaded {events.count()} events")  
display(events.limit(6))
```

A blue callout box points to the output of the second line: "Ready to go! Loaded 67,561,379 events". Another blue callout box points to the resulting DataFrame preview: "events: pyspark.sql.connect.DataFrame". The preview shows 7 rows of data with columns: event\_time, event\_type, product\_id, category\_id, category\_code, brand, price, and over\_9.

Ready to go! Loaded 6779 events

events: pyspark.sql.connect.DataFrame

Ready to go! Loaded 67,561,379 events

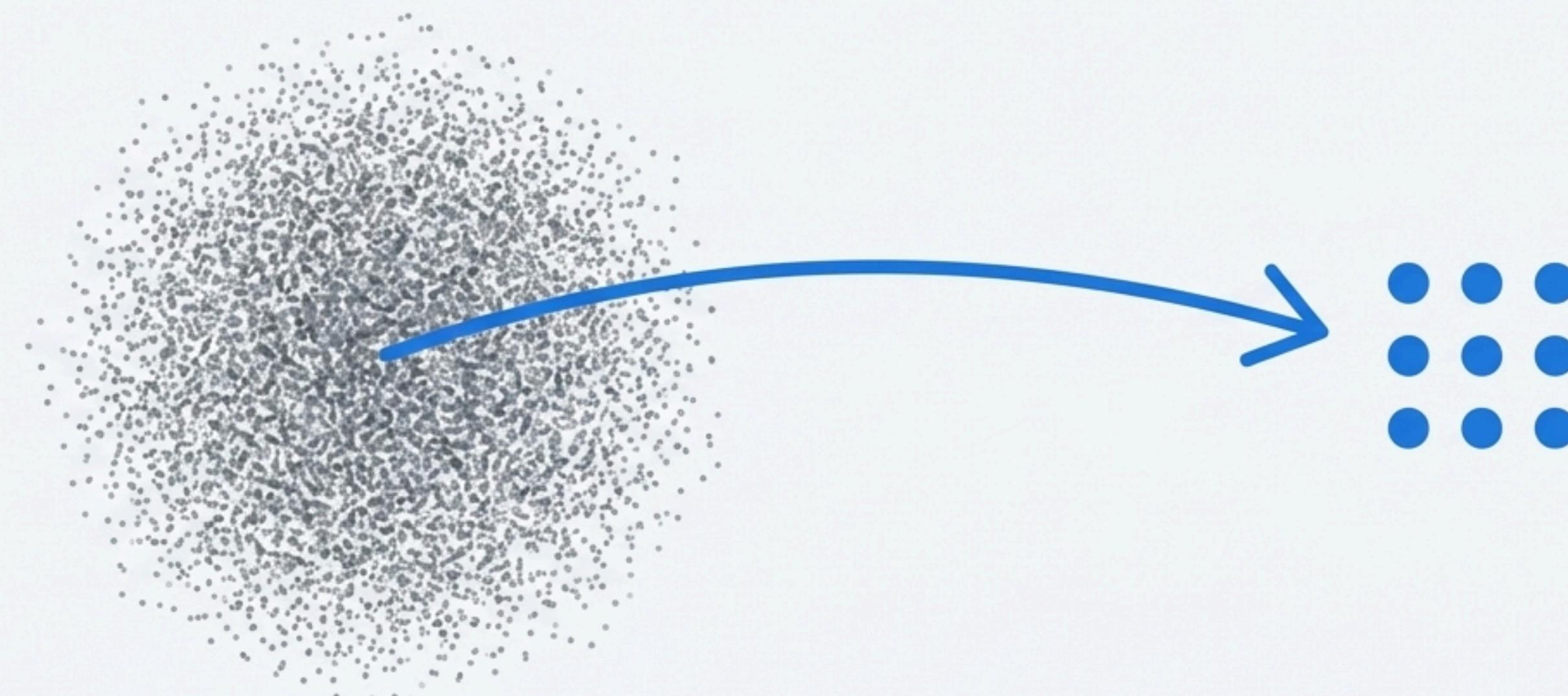
**67.5 Million Rows**

The code executes and immediately provides a confirmation count and a preview of the data.

The data is now a Spark DataFrame, a powerful, distributed data structure ready for analysis.

# Shifting from Macro to Micro to Master a Key Skill

We've confirmed we can handle massive datasets. To clearly understand the fundamentals of data manipulation, let's switch gears. We will now create a small, simple DataFrame from scratch. This allows us to focus on the **logic** of our analysis without the complexity of 67 million rows.



# Step 3: Building a Sample DataFrame in Seconds

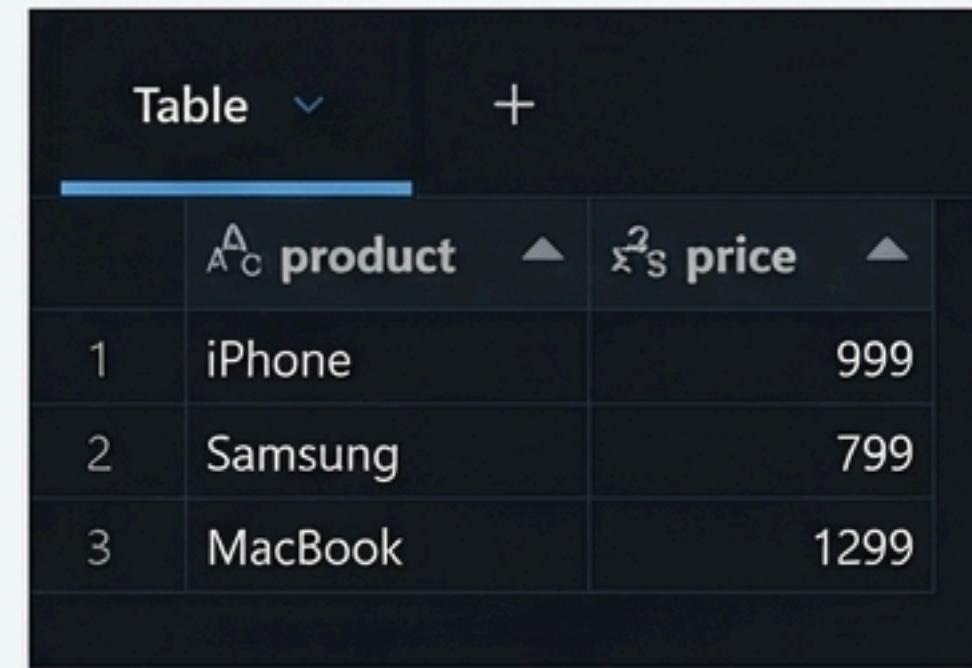
We can define data directly in our notebook using a simple Python list. Then, the `spark.createDataFrame` command instantly converts it into a fully functional Spark DataFrame, complete with named columns: "product" and "price".

```
code cell
# Create simple DataFrame
data = [('iPhone', 999), ('Samsung', 799), ('MacBook', 1299)]
df = spark.createDataFrame(data, ['product', 'price'])
df.show()
```

Standard Python list.

The command to convert our list into a distributed DataFrame.

# Our Sample Data, Ready for Analysis



|   | product | price |
|---|---------|-------|
| 1 | iPhone  | 999   |
| 2 | Samsung | 799   |
| 3 | MacBook | 1299  |

The result is a clean, structured table. Now that we have a simple dataset, we can easily ask questions of it.

# Step 4: Asking Our First Question—Which Products Cost More Than \$1000?

```
Just now (<1s)

# Filter expensive products
df.filter(df.price > 1000).show()

> See performance (1)

+-----+
|product|price|
+-----+
|MacBook| 1299|
+-----+
```

DataFrames have powerful built-in methods for analysis. To find expensive products, we use the `.filter()` method. The syntax is clear and expressive, directly translating our question into code. The `.show()` command displays the result.

The logical condition at the heart of our question.

# The Answer, Instantly Delivered

A screenshot of a Databricks Notebook interface. On the left is a sidebar with navigation links like Home, Workspace, Recents, Catalog, etc. The main area shows an 'Untitled Notebook' from '2025-01-09 11:03:17'. The notebook contains a table with columns 'product' and 'price', and three rows: iPhone (999), Samsung (759), and MacBook (1299). Below the table is a Python code cell:

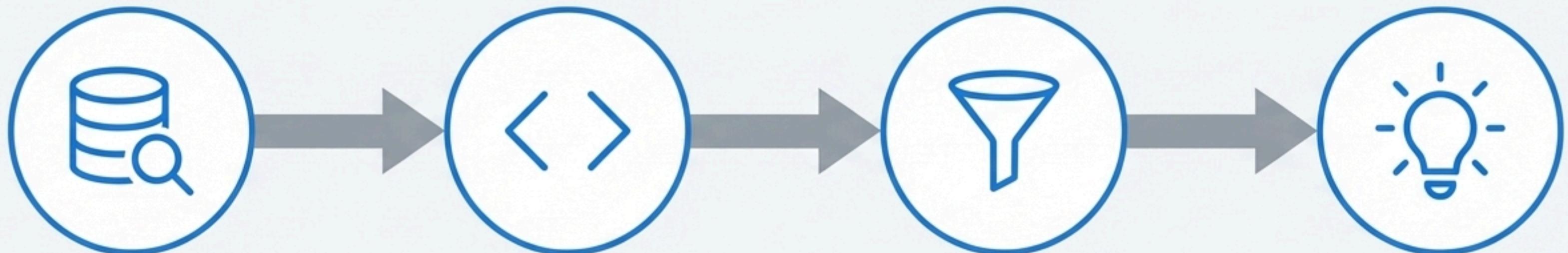
```
# Filter expensive products
df.filter(df.price > 1000).show()
```

An annotation with a blue arrow points from the table to the code cell. Another annotation with a purple box highlights the output of the code cell, which shows a filtered DataFrame with only the MacBook row:

|         |       |
|---------|-------|
| product | price |
| MacBook | 1299  |

**The DataFrame is filtered, returning only the rows that match our condition.**

# The Complete Workflow: From Discovery to Insight



**Discover:** Located our raw data files in the Unity Catalog.

**Load:** Ingested 67.5M rows into a Spark DataFrame with a single command.

**Analyze:** Asked a specific question using the `.filter()` method on a sample DataFrame.

**Insight:** Received an immediate, precise answer to our question.

Databricks Notebook

# Simple Tools for Complex Data

The Databricks platform provides a unified and interactive workflow. By seamlessly combining data discovery in the Unity Catalog with the large-scale processing of Spark and the analytical flexibility of DataFrames, data professionals can move from question to insight with unprecedented speed and simplicity.

