## **Azure Databricks Data Visualization & Partner Connect Overview**

Azure Databricks offers powerful data visualization capabilities, both through its **native visualization features** and **Partner Connect tools** such as Tableau, Power BI, and others. Below is a detailed breakdown of these features, capabilities, and the differences between native visualizations and partner tools.

## **1. Azure Databricks Native Visualization Features**

### **Capabilities & Features**

Azure Databricks provides built-in visualization tools for quick and effective data exploration directly within notebooks.

* **Built-in Charting & Graphing:**
  + Supports bar charts, scatter plots, line graphs, pie charts, histograms, area charts, and box plots.
* **Table & Pivot Table Support:**
  + Allows tabular visualization and pivoting for easy data summarization.
* **SQL & Python-Based Visualizations:**
  + Supports built-in visualization options within SQL queries using display() and visualization libraries such as matplotlib, seaborn, plotly, and ggplot in Python.
* **Interactive & Configurable Charts:**
  + Ability to customize chart types, axes, labels, colors, and drill-down features.
* **Integration with ML & AI Workflows:**
  + Supports quick visualization of machine learning datasets and model outputs within notebooks.
* **Time-Series Analysis:**
  + Supports visualization of time-series data with appropriate aggregations.

### **When to Use Native Visualization in Databricks**

✅ **Use Cases:**

* **Quick Exploratory Data Analysis (EDA)**: Ideal for quick data exploration while working within a notebook.
* **Lightweight & Embedded in Notebooks**: No need to switch between tools; everything is available inside Databricks.
* **For Ad-Hoc Analysis**: Suitable when a data scientist or analyst needs quick visual insights.
* **For Small to Medium Datasets**: Works well when analyzing small to medium-sized datasets directly within Databricks.

🚫 **Limitations:**

* **Not Ideal for Enterprise-Level Reporting**: Lacks advanced dashboarding and data storytelling capabilities.
* **Limited Customization**: While useful, customization is limited compared to tools like Power BI or Tableau.
* **Not Ideal for Large Datasets**: Visualizing massive datasets inside a Databricks notebook can be slow and inefficient.

## **2. Partner Connect Visualization Tools (Tableau, Power BI, etc.)**

Databricks **Partner Connect** allows seamless integration with leading BI tools such as **Tableau, Power BI, Qlik, Looker, and others** for more advanced visualization and reporting.

### **Key Features & Benefits**

1. **Enterprise-Grade Dashboards & Reports**
   1. Create complex, interactive dashboards for enterprise reporting needs.
2. **Live Connection to Databricks**
   1. Supports direct and live connections to Databricks without needing data extracts.
3. **Advanced Data Transformations & Blending**
   1. Tools like Power BI and Tableau allow blending data from multiple sources for richer insights.
4. **Collaboration & Sharing**
   1. BI tools provide built-in sharing, publishing, and collaboration features for large teams.
5. **Security & Governance**
   1. Leverage Azure AD authentication, role-based access control (RBAC), and row-level security.
6. **Performance Optimization**
   1. Optimized for handling large datasets using Databricks SQL Warehouse and query acceleration.

### **Comparison of Key BI Tools**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Feature** | **Tableau** | **Power BI** | **Qlik** | **Looker** |
| **Best For** | Advanced visual analytics | Microsoft ecosystem users | Associative data models | Embedded analytics |
| **Live Connection** | Yes | Yes | Yes | Yes |
| **Data Blending** | Yes | Yes | Yes | Yes |
| **AI & ML Integration** | Moderate | Strong (AI Insights) | Moderate | Strong |
| **Ease of Use** | Moderate | Easy for Microsoft users | Moderate | Moderate |
| **Collaboration** | Strong | Strong | Moderate | Strong |

### **When to Use Partner Connect Tools**

✅ **Use Cases:**

* **Enterprise Dashboards & Reporting**: When you need to create shareable and interactive dashboards for stakeholders.
* **Cross-Source Data Analysis**: When blending data from multiple sources beyond Databricks.
* **Advanced Visualizations & Customization**: For sophisticated visualizations that require high customization.
* **Performance Optimization for Large Datasets**: Best when working with large datasets where native visualizations struggle.
* **Collaboration & Distribution**: When reports need to be shared with a wider audience within the organization.

🚫 **Limitations:**

* **Learning Curve**: Requires learning a new BI tool, especially for non-technical users.
* **Licensing Costs**: Most BI tools have additional licensing costs, unlike Databricks’ native tools.
* **External Dependencies**: Needs external tools to be set up and maintained separately.

## **3. Choosing Between Databricks Native Visualization & Partner Connect BI Tools**

### **Decision Guide: When to Use What?**

|  |  |  |
| --- | --- | --- |
| **Scenario** | **Use Databricks Native Visualization** | **Use Partner BI Tools (Tableau, Power BI, etc.)** |
| **Quick Exploratory Data Analysis** | ✅ | ❌ |
| **Building Enterprise Dashboards** | ❌ | ✅ |
| **Handling Large Datasets** | ❌ (Limited Performance) | ✅ |
| **Highly Customizable & Interactive Visuals** | ❌ | ✅ |
| **Ad-hoc Data Exploration in Notebooks** | ✅ | ❌ |
| **Collaboration & Report Sharing** | ❌ | ✅ |
| **Blending Multiple Data Sources** | ❌ | ✅ |
| **Automated Reporting** | ❌ | ✅ |

### **Final Thoughts**

* **Use Databricks Native Visualization** for **quick insights, EDA, and interactive exploration** in a notebook.
* **Use Power BI, Tableau, or Qlik** when **creating enterprise-grade dashboards, large-scale reporting, and collaboration**.
* **If you need advanced analytics and ML integration**, Databricks combined with a BI tool like Power BI or Tableau offers the best of both worlds.

## **Azure Databricks Native Visualizations & Dashboards: Detailed Guide**

Azure Databricks offers **native visualization** and **dashboarding** capabilities directly within notebooks and Databricks SQL, supporting both **batch and streaming data processing**. Below is a **comprehensive breakdown** of these features, how to create dashboards, and how to refresh them for **batch and streaming workflows**.

# **1. Databricks Native Visualization Features**

Azure Databricks provides **built-in visualization tools** within both **Databricks notebooks** and **Databricks SQL**.

## **1.1 Notebook-Based Visualizations**

Databricks notebooks allow users to create interactive visualizations directly within **SQL, Python, Scala, and R** cells.

### **Key Features:**

* **Auto-Generated Visualizations**:
  + Run a SELECT query in a notebook and click on the "Visualization" button to choose from **bar, line, scatter, area, pie, and other charts**.
* **Customizable Axes & Formatting**:
  + Modify chart types, axis labels, colors, aggregations, and display settings.
* **Dynamic Filtering & Interactivity**:
  + Apply filters to refine data without modifying the SQL query.
* **Pivot Tables**:
  + Create pivot tables from tabular datasets for better summarization.
* **Geospatial Charts**:
  + Map visualizations for geospatial data (e.g., plotting coordinates on a map).

### **Example: Creating a Native Visualization in a Notebook**

sql

CopyEdit

SELECT country, SUM(sales) as total\_sales  
FROM sales\_data  
GROUP BY country  
ORDER BY total\_sales DESC

1. Run the query.
2. Click on the **“+” (Visualization)** button.
3. Choose a chart type (e.g., bar, line, scatter).
4. Customize as needed.

## **1.2 Databricks SQL-Based Dashboards & Visualization**

Databricks **SQL** enables **dashboard creation** with auto-refresh and scheduling capabilities.

### **Key Features**

* **Create Dashboards** with multiple visualizations from different SQL queries.
* **Drag-and-Drop Layout** to organize visualizations.
* **Filters & Parameters** for interactive exploration.
* **Scheduled Query Execution** for refreshing data (batch mode).
* **Alerts & Notifications** based on query results.
* **Streaming Data Visualization** for real-time updates.

### **Creating a Databricks SQL Dashboard**

1. **Navigate to Databricks SQL → Dashboards**.
2. **Run a SQL Query**:

sql

CopyEdit

SELECT product\_category, SUM(revenue) AS total\_revenue  
FROM sales\_data  
GROUP BY product\_category  
ORDER BY total\_revenue DESC

1. Click **“New Visualization”**, choose a chart type, and configure settings.
2. Save the visualization and **Add it to a Dashboard**.
3. Repeat the process for multiple queries to build the dashboard.

# **2. Refreshing Databricks Dashboards & SQL Queries**

### **Refreshing Queries & Dashboards for Batch and Streaming Data**

Azure Databricks supports two main refresh strategies:

1. **Batch Processing (Scheduled Execution)**
2. **Streaming (Auto-Refreshing)**

## **2.1 Refreshing Dashboards for Batch Data Processing**

### **Method 1: Manual Refresh**

* Users can manually refresh dashboards by clicking the **"Refresh"** button.
* Useful when ad-hoc analysis is required.

### **Method 2: Scheduled Refresh in Databricks SQL**

* Schedule SQL queries to **run at a fixed interval** to refresh dashboard data.
* Useful for **daily/weekly/monthly** reporting.

#### **Steps to Schedule a Refresh**

1. **Open a Query in Databricks SQL.**
2. Click on **“Schedule”** (clock icon).
3. **Set the Execution Frequency** (e.g., every 15 minutes, hourly, daily).
4. **Choose a Warehouse** to run the query.
5. Save the schedule.

🔥 **Example: Scheduling a Query to Run Every Hour**

sql

CopyEdit

SELECT product\_category, SUM(revenue) AS total\_revenue  
FROM sales\_data  
GROUP BY product\_category  
ORDER BY total\_revenue DESC

* Schedule it **every 1 hour**.
* The dashboard will automatically update based on this schedule.

## **2.2 Refreshing Dashboards for Streaming Data Processing**

Streaming data requires **continuous updates** instead of scheduled batch execution.

### **Method 1: Auto-Refresh (Real-Time Streaming)**

* Databricks SQL dashboards support **auto-refresh** for streaming data sources.
* Can be configured to refresh every **few seconds**.

#### **Steps to Enable Auto-Refresh:**

1. **Open the Databricks SQL Dashboard.**
2. Click on **"Refresh Settings"**.
3. Set **Auto-Refresh Interval** (e.g., every 10 seconds).
4. Save the settings.

⚡ **Use Case:**

* Monitoring real-time event streams (e.g., IoT sensor data, stock prices).

### **Method 2: Using Structured Streaming with Auto-Updates**

For advanced streaming scenarios, use **Databricks Structured Streaming**.

**Example: Creating a Streaming Table & Connecting to a Dashboard**

python

CopyEdit

from pyspark.sql.functions import \*  
from pyspark.sql.types import \*  
  
# Define Streaming Source (e.g., Kafka)  
streaming\_data = spark.readStream \  
 .format("kafka") \  
 .option("kafka.bootstrap.servers", "your\_kafka\_server") \  
 .option("subscribe", "your\_topic") \  
 .load()  
  
# Transform Data  
processed\_data = streaming\_data.selectExpr("CAST(value AS STRING)")  
  
# Write to a Temporary Table (Delta Lake)  
processed\_data.writeStream \  
 .format("delta") \  
 .option("checkpointLocation", "/mnt/checkpoints/") \  
 .outputMode("append") \  
 .table("real\_time\_data")

Now, create a **Databricks SQL Dashboard** using:

sql

CopyEdit

SELECT \* FROM real\_time\_data

* Enable **Auto-Refresh** to visualize new data as it arrives.

🔥 **Best Practices for Streaming Dashboards**

* Use **Delta Tables** for **real-time analytics**.
* Keep the **refresh interval low** (e.g., 10s–30s) to avoid performance issues.
* Optimize with **Databricks SQL Warehouse** for fast query execution.

# **3. Comparison: Batch vs. Streaming Refresh for Dashboards**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Batch Processing** | **Streaming Processing** |
| **Update Frequency** | Scheduled (e.g., hourly/daily) | Continuous (real-time updates) |
| **Data Size Handling** | Large historical data | Real-time event streams |
| **Performance Impact** | Lower CPU usage | Higher CPU usage (frequent updates) |
| **Use Cases** | Business reporting, sales trends | IoT monitoring, stock price tracking |
| **Dashboard Refresh** | Manual or Scheduled | Auto-refresh every few seconds |

# **4. Best Practices for Dashboard Performance Optimization**

1. **Use Delta Lake for Faster Queries**
   1. Optimizes query execution for large datasets.
2. **Optimize Queries with Indexing & Caching**
   1. Cache frequently accessed tables.
3. **Set Reasonable Refresh Intervals**
   1. Avoid very frequent refreshes for large dashboards.
4. **Use Databricks SQL Warehouses**
   1. Provides **better query performance** than traditional Spark queries.
5. **Partition Data for Faster Processing**
   1. Helps in reducing query execution time for large datasets.

# **5. Summary: Choosing the Right Approach**

|  |  |
| --- | --- |
| **Scenario** | **Best Approach** |
| **Static Reports (e.g., daily sales reports)** | Batch processing with scheduled execution |
| **Real-Time Monitoring (e.g., stock prices, IoT sensors)** | Streaming dashboards with auto-refresh |
| **High-Performance Dashboards** | Use Databricks SQL Warehouses + optimized queries |
| **Ad-hoc Data Exploration** | Use Databricks notebook visualizations |

# **📌 PART 1: Setting Up a Batch Processing Dashboard**

### **Objective:**

Create a **batch-processing dashboard** using **Databricks SQL** and **scheduled queries** with sample sales data.

## **🔹 Step 1: Load Sample Batch Data**

We'll use **Delta Tables** to store batch data.

### **1️⃣ Create a New Notebook & Load Sample Sales Data**

Run the following **Python (PySpark) script** to create a **sample Delta table**:

python

CopyEdit

from pyspark.sql import SparkSession  
from pyspark.sql.functions import col  
from pyspark.sql.types import StructType, StructField, IntegerType, StringType, DoubleType  
  
# Define schema  
schema = StructType([  
 StructField("sale\_id", IntegerType(), True),  
 StructField("product", StringType(), True),  
 StructField("category", StringType(), True),  
 StructField("amount", DoubleType(), True),  
 StructField("date", StringType(), True)  
])  
  
# Sample data  
data = [  
 (1, "Laptop", "Electronics", 1200.50, "2024-02-10"),  
 (2, "Phone", "Electronics", 800.75, "2024-02-10"),  
 (3, "Tablet", "Electronics", 450.00, "2024-02-11"),  
 (4, "TV", "Appliances", 999.99, "2024-02-12"),  
 (5, "Fridge", "Appliances", 1500.00, "2024-02-12")  
]  
  
# Create DataFrame  
df = spark.createDataFrame(data, schema=schema)  
  
# Write to Delta Table  
df.write.mode("overwrite").format("delta").saveAsTable("sales\_data\_batch")  
  
print("Batch sales data loaded successfully!")

* This creates a **Delta Table (sales\_data\_batch)**.

## **🔹 Step 2: Create a SQL Query in Databricks SQL**

1. Go to **Databricks SQL → Queries**.
2. Click **“New Query”**.
3. Run the following **SQL query**:

sql

CopyEdit

SELECT category, SUM(amount) AS total\_sales  
FROM sales\_data\_batch  
GROUP BY category  
ORDER BY total\_sales DESC;

1. Click **"New Visualization"** → Choose **Bar Chart**.

## **🔹 Step 3: Create & Schedule the Dashboard**

1. Click **“Save & Add to Dashboard”** → Create a new **"Sales Report Dashboard"**.
2. Go to **Dashboards** → Open **"Sales Report Dashboard"**.
3. Click **"Schedule" (Clock Icon) → Set Refresh Every 1 Hour**.
4. Click **“Save”**.

🎉 **Your batch-processing dashboard is now set up!**  
Every hour, it will **refresh with new sales data**.

# **📌 PART 2: Setting Up a Streaming Dashboard**

### **Objective:**

Create a **real-time streaming dashboard** using **Databricks Structured Streaming** and **Auto-Refresh SQL Dashboards**.

## **🔹 Step 1: Simulate Streaming Data (Kafka Alternative)**

We'll simulate streaming data using a **Python notebook**.

### **1️⃣ Create a Streaming Table**

Run this **Python script**:

python

CopyEdit

from pyspark.sql.types import \*  
from pyspark.sql.functions import \*  
import time  
  
# Define schema  
schema = StructType([  
 StructField("sale\_id", IntegerType(), True),  
 StructField("product", StringType(), True),  
 StructField("category", StringType(), True),  
 StructField("amount", DoubleType(), True),  
 StructField("timestamp", StringType(), True)  
])  
  
# Function to continuously insert streaming data  
def generate\_streaming\_data():  
 sample\_data = [  
 (6, "Smartwatch", "Electronics", 299.99, "2024-02-12 10:15:00"),  
 (7, "Microwave", "Appliances", 120.00, "2024-02-12 10:16:00"),  
 (8, "Headphones", "Electronics", 150.00, "2024-02-12 10:17:00"),  
 (9, "Washing Machine", "Appliances", 700.00, "2024-02-12 10:18:00")  
 ]  
   
 df = spark.createDataFrame(sample\_data, schema=schema)  
 df.write.mode("append").format("delta").saveAsTable("sales\_data\_stream")  
 print("Inserted new streaming data...")  
  
# Simulate new data every 5 seconds  
for i in range(5):  
 generate\_streaming\_data()  
 time.sleep(5)

* This **continuously adds new sales transactions** every **5 seconds**.

## **🔹 Step 2: Set Up a Structured Streaming Query**

Run this **SQL query in Databricks SQL**:

sql

CopyEdit

SELECT category, SUM(amount) AS total\_sales  
FROM sales\_data\_stream  
GROUP BY category  
ORDER BY total\_sales DESC;

* Click **"New Visualization"** → Choose **Real-Time Line Chart**.

## **🔹 Step 3: Auto-Refresh Streaming Dashboard**

1. Click **"Save & Add to Dashboard"** → Create a **"Real-Time Sales Dashboard"**.
2. Open **Dashboards** → Select **"Real-Time Sales Dashboard"**.
3. Click **"Refresh Settings"** → Set **Refresh Interval = 10 seconds**.
4. Click **“Save”**.

🎉 **Your streaming dashboard is now live!**

* It refreshes every **10 seconds** as new data arrives.
* Works great for **real-time sales monitoring, IoT, or stock prices**.

# **📊 Summary: Comparing Batch vs. Streaming Dashboards**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Batch Processing Dashboard** | **Streaming Dashboard** |
| **Data Source** | Static/periodic updates (e.g., daily sales) | Real-time event data |
| **Update Frequency** | Scheduled (e.g., every 1 hour) | Continuous (auto-refresh every 10s) |
| **Best For** | Monthly/weekly reports, aggregated data | Live dashboards, IoT data, stock prices |
| **Performance Impact** | Lower (less frequent updates) | Higher (continuous updates) |
| **Implementation** | Scheduled SQL query | Structured Streaming |

# **🎯 Final Thoughts**

* ✅ **Batch dashboards** are ideal for **historical reports** and **aggregated insights**.
* ✅ **Streaming dashboards** are best for **real-time monitoring** of **fast-changing data**.
* ✅ **Both** can be combined for a **hybrid solution** (e.g., daily reports + live metrics).

## **🚀 Deployment Plan**

### **✅ Step 1: Set Up Your Azure Databricks Environment**

Before running any code, ensure:

* You have an **Azure Databricks workspace**.
* A **Databricks SQL Warehouse** is available for running queries.
* The necessary **cluster is running** (for running notebooks).
* You have the correct **permissions** to create tables, dashboards, and schedules.

## **🛠 Step 2: Deploy the Batch Processing Dashboard**

### **1️⃣ Load Sample Data into a Delta Table**

#### **Run this in a Databricks Notebook (Python Cell):**

python

CopyEdit

from pyspark.sql import SparkSession  
from pyspark.sql.types import StructType, StructField, IntegerType, StringType, DoubleType  
  
# Define schema  
schema = StructType([  
 StructField("sale\_id", IntegerType(), True),  
 StructField("product", StringType(), True),  
 StructField("category", StringType(), True),  
 StructField("amount", DoubleType(), True),  
 StructField("date", StringType(), True)  
])  
  
# Sample batch data  
data = [  
 (1, "Laptop", "Electronics", 1200.50, "2024-02-10"),  
 (2, "Phone", "Electronics", 800.75, "2024-02-10"),  
 (3, "Tablet", "Electronics", 450.00, "2024-02-11"),  
 (4, "TV", "Appliances", 999.99, "2024-02-12"),  
 (5, "Fridge", "Appliances", 1500.00, "2024-02-12")  
]  
  
# Create DataFrame  
df = spark.createDataFrame(data, schema=schema)  
  
# Write to a Delta Table  
df.write.mode("overwrite").format("delta").saveAsTable("sales\_data\_batch")  
  
print("Batch sales data loaded successfully!")

📌 **This will create a Delta Table (sales\_data\_batch) for batch reporting.**

### **2️⃣ Create a SQL Query for the Batch Dashboard**

#### **Go to Databricks SQL → Click "New Query" → Run:**

sql

CopyEdit

SELECT category, SUM(amount) AS total\_sales  
FROM sales\_data\_batch  
GROUP BY category  
ORDER BY total\_sales DESC;

* Click **“New Visualization”** → Choose **Bar Chart**.
* Customize chart **titles, colors, and labels**.

### **3️⃣ Create a Scheduled Batch Dashboard**

1. Click **"Save & Add to Dashboard"** → Create a new **"Sales Report Dashboard"**.
2. Open **Dashboards** → Select **"Sales Report Dashboard"**.
3. Click **"Schedule" (Clock Icon)** → **Set Refresh Interval to 1 Hour**.
4. Click **“Save”**.

🎉 **Your batch dashboard is live!**  
✅ It **automatically updates every hour** with new batch data.

## **⚡ Step 3: Deploy the Streaming Dashboard**

### **1️⃣ Create a Streaming Table**

We will simulate streaming sales transactions.

#### **Run this in a Databricks Notebook (Python Cell):**

python

CopyEdit

from pyspark.sql.types import StructType, StructField, IntegerType, StringType, DoubleType  
from pyspark.sql.functions import \*  
import time  
  
# Define schema  
schema = StructType([  
 StructField("sale\_id", IntegerType(), True),  
 StructField("product", StringType(), True),  
 StructField("category", StringType(), True),  
 StructField("amount", DoubleType(), True),  
 StructField("timestamp", StringType(), True)  
])  
  
# Function to continuously insert streaming data  
def generate\_streaming\_data():  
 sample\_data = [  
 (6, "Smartwatch", "Electronics", 299.99, "2024-02-12 10:15:00"),  
 (7, "Microwave", "Appliances", 120.00, "2024-02-12 10:16:00"),  
 (8, "Headphones", "Electronics", 150.00, "2024-02-12 10:17:00"),  
 (9, "Washing Machine", "Appliances", 700.00, "2024-02-12 10:18:00")  
 ]  
   
 df = spark.createDataFrame(sample\_data, schema=schema)  
 df.write.mode("append").format("delta").saveAsTable("sales\_data\_stream")  
 print("Inserted new streaming data...")  
  
# Simulate new data every 5 seconds  
for i in range(5):  
 generate\_streaming\_data()  
 time.sleep(5)

📌 **This script continuously inserts new transactions into sales\_data\_stream.**

### **2️⃣ Set Up a Streaming SQL Query**

#### **Go to Databricks SQL → Click "New Query" → Run:**

sql

CopyEdit

SELECT category, SUM(amount) AS total\_sales  
FROM sales\_data\_stream  
GROUP BY category  
ORDER BY total\_sales DESC;

* Click **“New Visualization”** → Choose **Real-Time Line Chart**.

### **3️⃣ Configure Auto-Refreshing Streaming Dashboard**

1. Click **"Save & Add to Dashboard"** → Create **"Real-Time Sales Dashboard"**.
2. Open **Dashboards** → Select **"Real-Time Sales Dashboard"**.
3. Click **"Refresh Settings"** → Set **Auto-Refresh Every 10 Seconds**.
4. Click **“Save”**.

🎉 **Your streaming dashboard is live!**  
✅ **Automatically updates every 10 seconds** as new data arrives.

# **📊 Final Deployment Summary**

|  |  |  |
| --- | --- | --- |
| **Step** | **Batch Dashboard** | **Streaming Dashboard** |
| **Data Source** | Static, scheduled updates (Delta Table) | Real-time stream (Delta Table) |
| **Update Frequency** | Every 1 hour (Scheduled) | Every 10 seconds (Auto-Refresh) |
| **Use Case** | Monthly reports, business insights | Live monitoring, IoT data |
| **SQL Query** | Aggregates static data | Processes real-time data |
| **Dashboard Type** | Bar charts, scheduled updates | Real-time charts, continuous refresh |

# **🔗 Setting Up Partner Connect for Azure Databricks (Power BI & Tableau)**

Partner Connect allows you to easily integrate **BI tools like Power BI and Tableau** with Azure Databricks for **seamless data visualization**. Below is a **detailed step-by-step guide** for connecting both tools.

# **📌 PART 1: Connecting Power BI to Azure Databricks**

## **🔹 Step 1: Enable Partner Connect in Databricks**

1. **Log in** to your Azure Databricks workspace.
2. Go to **Partner Connect** (**Left Sidebar → Partner Connect**).
3. Click **Power BI** from the available partners.

## **🔹 Step 2: Configure Power BI Connection**

1. Select **your Databricks SQL Warehouse** (or create a new one if needed).
2. Click **“Connect”**.
3. Power BI will open with a **Databricks Connector** pre-configured.
4. **Sign in to Power BI** and allow the connection.

## **🔹 Step 3: Manually Connect Power BI to Databricks (Optional)**

If the automatic setup fails, you can **manually connect**:

### **1️⃣ Open Power BI Desktop**

1. Open **Power BI Desktop**.
2. Click **Get Data → Azure → Azure Databricks**.
3. Click **Connect**.

### **2️⃣ Get Databricks Connection Details**

1. In **Databricks**, go to **SQL Warehouses**.
2. Click on the **Warehouse you want to use**.
3. Copy the **JDBC connection string**.

### **3️⃣ Enter Connection Details in Power BI**

1. In Power BI, paste the **server hostname** and **HTTP path**.
2. Use **Personal Access Token (PAT)** for authentication:
   1. Generate a **Token** in Databricks (Settings → User Settings → Generate New Token).
   2. Paste the token in Power BI.
3. Click **Connect** → Select your Databricks table.

✅ **Power BI is now connected to Databricks!** You can start building reports and dashboards.

## **🔹 Step 4: Publish Reports to Power BI Service**

1. In Power BI, click **Publish**.
2. Select a **Power BI Workspace**.
3. The report is now available in **Power BI Online**.
4. Set **scheduled refresh** (if needed) under **Dataset Settings**.

📌 **Power BI works well for enterprise BI needs** with scheduled reporting.

# **📌 PART 2: Connecting Tableau to Azure Databricks**

## **🔹 Step 1: Enable Partner Connect**

1. Open **Azure Databricks**.
2. Go to **Partner Connect** (**Left Sidebar → Partner Connect**).
3. Click **Tableau**.

## **🔹 Step 2: Configure Tableau Connection**

1. Select your **Databricks SQL Warehouse**.
2. Click **Connect**.
3. Tableau will open with a **pre-configured Databricks Connector**.

## **🔹 Step 3: Manually Connect Tableau to Databricks (Optional)**

If the automatic setup fails:

### **1️⃣ Open Tableau Desktop**

1. Open **Tableau Desktop**.
2. Click **Connect to a Server → Databricks**.

### **2️⃣ Get Databricks Connection Details**

1. In Databricks, go to **SQL Warehouses**.
2. Copy:
   1. **Server Hostname**
   2. **HTTP Path**
   3. **Personal Access Token (PAT)** (Generate under User Settings)

### **3️⃣ Enter Connection Details in Tableau**

1. Paste the **Server Hostname** & **HTTP Path** in Tableau.
2. Select **Personal Access Token (PAT)** as the authentication method.
3. Paste the **PAT token**.
4. Click **Sign In**.

✅ **Tableau is now connected to Databricks!** You can start creating visualizations.

## **🔹 Step 4: Publish Dashboards to Tableau Server**

1. In Tableau, click **Server → Publish Workbook**.
2. Select **Tableau Server or Tableau Online**.
3. Choose **your workspace**.
4. Set up **scheduled extract refresh** (if needed).

📌 **Tableau works well for advanced visualizations and self-service analytics.**

# **📊 Power BI vs. Tableau with Databricks**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Power BI** | **Tableau** |
| **Ease of Setup** | Easier (Direct Azure Databricks Connector) | Requires more manual setup |
| **Best For** | Enterprise BI, Scheduled Reporting | Advanced Visuals, Self-Service BI |
| **Performance** | Optimized for Microsoft stack | High-speed interactive analytics |
| **Licensing** | Lower cost for Azure users | Higher cost (per user licensing) |

# **🎯 Final Steps**

* ✅ **Power BI** is best for **enterprise reports** and **scheduled insights**.
* ✅ **Tableau** is better for **interactive, self-service dashboards**.
* ✅ Both can be used **together** for different use cases.

### **📊 Azure Databricks Native Visualization Features & Capabilities**

Azure Databricks provides built-in visualization tools for **quick, interactive data exploration** without needing third-party BI tools like Power BI or Tableau. Below is a **detailed list of features** and their capabilities.

## **🔹 1. Built-in Visualizations**

Databricks SQL and Notebooks support **native visualizations** that allow users to convert query results into interactive charts and graphs.

### **📈 Available Chart Types**

|  |  |  |
| --- | --- | --- |
| **Chart Type** | **Description** | **Best For** |
| **Bar Chart** | Horizontal/Vertical bars to compare values | Category-wise sales, trends |
| **Line Chart** | Lines connecting data points over time | Time series analysis, forecasting |
| **Area Chart** | Similar to a line chart but shaded | Cumulative trends |
| **Pie Chart** | Displays proportions in a circular chart | Market share, category distribution |
| **Scatter Plot** | Dots representing relationships between variables | Correlation, trend analysis |
| **Histogram** | Distribution of numerical data | Frequency analysis, data distribution |
| **Box Plot** | Visualizes statistical distribution | Outlier detection, data spread |
| **Heatmap** | Color-coded matrix for large data comparison | Trend spotting, clustering |
| **Pivot Table** | Interactive table to summarize data | Aggregated views, filtering |

## **🔹 2. Key Features of Databricks Visualizations**

### **✅ SQL-Based Visualization**

* Query execution in **Databricks SQL** automatically provides visualization options.
* **One-click conversion** of SQL query results into a chart.
* **Drag-and-drop functionality** to configure visuals.

### **✅ Notebook-Based Visualization**

* Supports **Python (Matplotlib, Seaborn, Plotly)** & **Scala visualization libraries**.
* **Magic Commands (%sql)** to create charts from SQL queries.
* **Interactive widgets** to dynamically filter visualizations.

### **✅ Dashboard Creation & Management**

* Save multiple visualizations in a **single dashboard**.
* Supports **real-time updates** for streaming data.
* Ability to **share dashboards** with team members.
* **Scheduled Refresh**: Dashboards can refresh automatically at custom intervals.

### **✅ Interactivity & Customization**

* **Filters & Parameters**: Create **interactive dashboards** with user-driven filters.
* **Color Schemes**: Custom colors, labels, and themes for different chart types.
* **Sorting & Aggregation**: Modify chart behavior with **grouping, sorting, and aggregations**.

### **✅ Streaming Data Visualization**

* Supports **real-time dashboards** with auto-refreshing data.
* Compatible with **Delta Live Tables** and **Structured Streaming**.
* **Auto-refresh intervals**: 10 seconds, 30 seconds, 1 minute, etc.

### **✅ Export & Sharing**

* **Download charts** as PNG, JPG, or CSV.
* **Embed dashboards** in external applications.
* **Public/Private sharing** for team collaboration.

## **🔹 3. Dashboard Refreshing for Batch & Streaming Data**

### **🔄 Batch Data**

* Uses **static Delta Tables**.
* Supports **scheduled refresh (hourly, daily, weekly, etc.)**.
* Best for **historical trends, summary reports**.

### **⚡ Streaming Data**

* Uses **Delta Live Tables** or **Structured Streaming**.
* Supports **auto-refresh every few seconds**.
* Best for **real-time monitoring, live business intelligence**.

## **🔹 4. When to Use Databricks Native Visualizations?**

|  |  |  |
| --- | --- | --- |
| **Scenario** | **Use Databricks Native Visuals?** | **Why?** |
| **Quick Data Exploration** | ✅ Yes | Fast, built-in, no setup required |
| **Ad-hoc Analysis** | ✅ Yes | Easy SQL + Visuals in one place |
| **Large Datasets** | ✅ Yes | Optimized for big data queries |
| **Scheduled Reports** | ✅ Yes | Supports auto-refresh |
| **Advanced BI Dashboards** | ❌ No | Use Power BI/Tableau for better customization |
| **Executive Reporting** | ❌ No | Power BI/Tableau offer more formatting |

# **🎯 Summary**

* **Azure Databricks provides built-in visualizations** for quick **data analysis and dashboards**.
* Supports **batch & streaming data**, **auto-refresh**, and **interactive filtering**.
* **Great for internal analysis**, but for **complex BI needs**, use **Power BI/Tableau**.

# **📊 Step-by-Step Guide: Creating Interactive Dashboards in Azure Databricks**

This guide will walk you through **creating interactive dashboards** in Databricks using **native visualizations**, including **batch and streaming data visualization**, **auto-refresh settings**, and **filters**.

## **✅ Step 1: Prepare Your Data in Databricks**

Before creating a dashboard, you need to **load your data into a Delta Table**.

### **🔹 1. Create Sample Batch Data**

Run the following Python code in a Databricks notebook to create a sample dataset:

python

CopyEdit

from pyspark.sql import SparkSession  
from pyspark.sql.types import StructType, StructField, IntegerType, StringType, DoubleType  
  
# Define schema  
schema = StructType([  
 StructField("sale\_id", IntegerType(), True),  
 StructField("product", StringType(), True),  
 StructField("category", StringType(), True),  
 StructField("amount", DoubleType(), True),  
 StructField("date", StringType(), True)  
])  
  
# Sample batch data  
data = [  
 (1, "Laptop", "Electronics", 1200.50, "2024-02-10"),  
 (2, "Phone", "Electronics", 800.75, "2024-02-10"),  
 (3, "Tablet", "Electronics", 450.00, "2024-02-11"),  
 (4, "TV", "Appliances", 999.99, "2024-02-12"),  
 (5, "Fridge", "Appliances", 1500.00, "2024-02-12")  
]  
  
# Create DataFrame  
df = spark.createDataFrame(data, schema=schema)  
  
# Write to a Delta Table  
df.write.mode("overwrite").format("delta").saveAsTable("sales\_data\_batch")  
  
print("Batch sales data loaded successfully!")

✅ **This creates a Delta Table (sales\_data\_batch) with sales data.**  
Now, we can visualize it in **Databricks SQL**.

## **✅ Step 2: Create Visualizations**

1. **Go to Databricks SQL** (Click on the SQL icon in the left sidebar).
2. **Click "New Query"** and run the following SQL query:

sql

CopyEdit

SELECT category, SUM(amount) AS total\_sales  
FROM sales\_data\_batch  
GROUP BY category  
ORDER BY total\_sales DESC;

1. Click **"New Visualization"** and select a **Bar Chart**.
2. **Customize the visualization:**
   1. **Title:** "Sales by Category"
   2. **X-Axis:** "Category"
   3. **Y-Axis:** "Total Sales"
   4. **Sort:** Descending
3. Click **"Save"**.

✅ **Your first visualization is ready!**

## **✅ Step 3: Create a Dashboard**

1. Go to **Dashboards** (left sidebar).
2. Click **"New Dashboard"** → Name it **"Sales Performance Dashboard"**.
3. Click **"Add Visualization"** and select the **Bar Chart** you just created.

### **🔹 Add More Charts**

Repeat **Step 2** to create additional charts:

* **Line Chart:** Sales trend over time.
* **Pie Chart:** Sales contribution by product.
* **Pivot Table:** Detailed sales breakdown.

📌 **Drag & resize charts** inside the dashboard for a better layout.

## **✅ Step 4: Add Interactivity (Filters & Widgets)**

1. **Click “Edit Dashboard”**.
2. Click **"Add a Filter"**.
3. Choose **"Category"** as a filter.
4. Set the filter type to **Dropdown**.
5. Click **"Save"**.

📌 **Now users can select categories to filter dashboard results dynamically!**

## **✅ Step 5: Auto-Refresh for Batch Data**

1. Open **your dashboard**.
2. Click **"Schedule Refresh" (Clock Icon)**.
3. Set refresh to **every 1 hour**.
4. Click **"Save"**.

✅ **Your dashboard now updates every hour with the latest batch data.**

## **✅ Step 6: Create a Streaming Dashboard**

For real-time updates, let’s create a **streaming table**.

### **🔹 1. Simulate Streaming Data**

Run the following Python code in Databricks:

python

CopyEdit

from pyspark.sql.functions import \*  
import time  
  
# Function to continuously insert streaming data  
def generate\_streaming\_data():  
 sample\_data = [  
 (6, "Smartwatch", "Electronics", 299.99, "2024-02-12 10:15:00"),  
 (7, "Microwave", "Appliances", 120.00, "2024-02-12 10:16:00"),  
 (8, "Headphones", "Electronics", 150.00, "2024-02-12 10:17:00"),  
 (9, "Washing Machine", "Appliances", 700.00, "2024-02-12 10:18:00")  
 ]  
   
 df = spark.createDataFrame(sample\_data, schema=schema)  
 df.write.mode("append").format("delta").saveAsTable("sales\_data\_stream")  
 print("Inserted new streaming data...")  
  
# Simulate new data every 5 seconds  
for i in range(5):  
 generate\_streaming\_data()  
 time.sleep(5)

📌 **This script inserts new rows into sales\_data\_stream every 5 seconds.**

### **🔹 2. Create a Real-Time Visualization**

1. **Run the following SQL query** in **Databricks SQL**:

sql

CopyEdit

SELECT category, SUM(amount) AS total\_sales  
FROM sales\_data\_stream  
GROUP BY category  
ORDER BY total\_sales DESC;

1. Click **"New Visualization"** and select **"Real-Time Line Chart"**.
2. Click **"Save & Add to Dashboard"**, choose **"Real-Time Sales Dashboard"**.

### **🔹 3. Enable Auto-Refreshing for Streaming Data**

1. Open **the "Real-Time Sales Dashboard"**.
2. Click **"Refresh Settings"**.
3. Set **Auto-Refresh to Every 10 Seconds**.
4. Click **"Save"**.

✅ **Now your dashboard updates every 10 seconds with live sales data!**

# **🎯 Final Summary**

|  |  |  |
| --- | --- | --- |
| **Step** | **Batch Dashboard** | **Streaming Dashboard** |
| **Data Source** | Static Delta Table | Delta Table with Structured Streaming |
| **Update Frequency** | Hourly Scheduled Refresh | Auto-refresh every 10 sec |
| **Best Visualization Type** | Bar, Line, Pie, Pivot Table | Real-Time Line, Heatmap |
| **Use Case** | Monthly reports, trend analysis | Live monitoring, real-time insights |

# **🚀 Next Steps**

* ✅ **Would you like to integrate this with Power BI or Tableau?**
* ✅ **Need help with Databricks SQL performance tuning?**
* ✅ **Want to explore advanced BI use cases like predictive analytics?**