|  |  |  |
| --- | --- | --- |
| **Lab** | **Topic** | **Link** |
| 1 (day5: Lab1-5) | Run your first ETL workload on Azure Databricks | https://learn.microsoft.com/en- us/azure/databricks/getting-started/etl-quick- start |
| 2 | Delta Lake | https://learn.microsoft.com/en- us/azure/databricks/delta/tutorial |
| 3 | Tutorial: Load and transform data in PySpark DataFrames | https://learn.microsoft.com/en- us/azure/databricks/getting- started/dataframes-python |
| 4 | Use SQL Warehouses in Azure Databricks | https://learn.microsoft.com/en- us/training/modules/use-sql-warehouses- azure-databricks/ |
| 5 | Build an end-to-end data pipeline in Databricks | https://learn.microsoft.com/en- us/azure/databricks/getting-started/data- pipeline-get-started |
| 6 (day4: Lab 1) | Sample Dashboards in Databricks+ Spark SQL | https://learn.microsoft.com/en- us/azure/databricks/sql/get-started/sample- dashboards |
| 7 (day4: Lab 2) | Visualize queries and create a dashboard in Databricks SQL | https://learn.microsoft.com/en- us/azure/databricks/sql/get- started/visualize-data-tutorial |

Lab 1: Run your first ETL workload on Azure Databrick - <https://learn.microsoft.com/en-us/azure/databricks/getting-started/etl-quick-start>

**Run your first ETL workload on Azure Databricks**

* Article
* 03/29/2024
* 5 contributors

**In this article**

1. [Requirements](https://learn.microsoft.com/en-us/azure/databricks/getting-started/etl-quick-start#requirements)
2. [Step 1: Create a cluster](https://learn.microsoft.com/en-us/azure/databricks/getting-started/etl-quick-start#--step-1-create-a-cluster)
3. [Step 2: Create a Databricks notebook](https://learn.microsoft.com/en-us/azure/databricks/getting-started/etl-quick-start#--step-2-create-a-databricks-notebook)
4. [Step 3: Configure Auto Loader to ingest data to Delta Lake](https://learn.microsoft.com/en-us/azure/databricks/getting-started/etl-quick-start#--step-3-configure-auto-loader-to-ingest-data-to-delta-lake)
5. [Step 4: Process and interact with data](https://learn.microsoft.com/en-us/azure/databricks/getting-started/etl-quick-start#--step-4-process-and-interact-with-data)
6. [Step 5: Schedule a job](https://learn.microsoft.com/en-us/azure/databricks/getting-started/etl-quick-start#--step-5-schedule-a-job)
7. [Additional Integrations](https://learn.microsoft.com/en-us/azure/databricks/getting-started/etl-quick-start#additional-integrations)

Learn how to use production-ready tools from Azure Databricks to develop and deploy your first extract, transform, and load (ETL) pipelines for data orchestration.

By the end of this article, you will feel comfortable:

1. [Launching a Databricks all-purpose compute cluster](https://learn.microsoft.com/en-us/azure/databricks/getting-started/etl-quick-start#cluster).
2. [Creating a Databricks notebook](https://learn.microsoft.com/en-us/azure/databricks/getting-started/etl-quick-start#notebook).
3. [Configuring incremental data ingestion to Delta Lake with Auto Loader](https://learn.microsoft.com/en-us/azure/databricks/getting-started/etl-quick-start#auto-loader).
4. [Executing notebook cells to process, query, and preview data](https://learn.microsoft.com/en-us/azure/databricks/getting-started/etl-quick-start#process).
5. [Scheduling a notebook as a Databricks job](https://learn.microsoft.com/en-us/azure/databricks/getting-started/etl-quick-start#schedule).

This tutorial uses interactive notebooks to complete common ETL tasks in Python or Scala.

You can also use Delta Live Tables to build ETL pipelines. Databricks created Delta Live Tables to reduce the complexity of building, deploying, and maintaining production ETL pipelines. See [Tutorial: Run your first Delta Live Tables pipeline](https://learn.microsoft.com/en-us/azure/databricks/delta-live-tables/tutorial-pipelines).

You can also use the [Databricks Terraform provider](https://learn.microsoft.com/en-us/azure/databricks/dev-tools/terraform/) to create this article’s resources. See [Create clusters, notebooks, and jobs with Terraform](https://learn.microsoft.com/en-us/azure/databricks/dev-tools/terraform/cluster-notebook-job).

**Requirements**

* You are logged into a Azure Databricks workspace.
* You have [permission to create a cluster](https://learn.microsoft.com/en-us/azure/databricks/compute/use-compute).

**Note**

If you do not have cluster control privileges, you can still complete most of the steps below as long as you have [**access to a cluster**](https://learn.microsoft.com/en-us/azure/databricks/compute/use-compute#permissions).

**Step 1: Create a cluster**

To do exploratory data analysis and data engineering, create a cluster to provide the compute resources needed to execute commands.

1. Click compute icon **Compute** in the sidebar.
2. On the Compute page, click **Create Cluster**. This opens the New Cluster page.
3. Specify a unique name for the cluster, leave the remaining values in their default state, and click **Create Cluster**.

To learn more about Databricks clusters, see [Compute](https://learn.microsoft.com/en-us/azure/databricks/compute/).

**Step 2: Create a Databricks notebook**

To get started writing and executing interactive code on Azure Databricks, create a notebook.

1. Click New Icon **New** in the sidebar, then click **Notebook**.
2. On the Create Notebook page:
   * Specify a unique name for your notebook.
   * Make sure the default language is set to **Python** or **Scala**.
   * Select the cluster you created in step 1 from the **Cluster** dropdown.
   * Click **Create**.

A notebook opens with an empty cell at the top.

To learn more about creating and managing notebooks, see [Manage notebooks](https://learn.microsoft.com/en-us/azure/databricks/notebooks/notebooks-manage).

**Step 3: Configure Auto Loader to ingest data to Delta Lake**

Databricks recommends using [Auto Loader](https://learn.microsoft.com/en-us/azure/databricks/ingestion/auto-loader/) for incremental data ingestion. Auto Loader automatically detects and processes new files as they arrive in cloud object storage.

Databricks recommends storing data with [Delta Lake](https://learn.microsoft.com/en-us/azure/databricks/delta/). Delta Lake is an open source storage layer that provides ACID transactions and enables the data lakehouse. Delta Lake is the default format for tables created in Databricks.

To configure Auto Loader to ingest data to a Delta Lake table, copy and paste the following code into the empty cell in your notebook:

**Python**

PythonCopy

# Import functions

from pyspark.sql.functions import col, current\_timestamp

# Define variables used in code below

file\_path = "/databricks-datasets/structured-streaming/events"

username = spark.sql("SELECT regexp\_replace(current\_user(), '[^a-zA-Z0-9]', '\_')").first()[0]

table\_name = f"{username}\_etl\_quickstart"

checkpoint\_path = f"/tmp/{username}/\_checkpoint/etl\_quickstart"

# Clear out data from previous demo execution

spark.sql(f"DROP TABLE IF EXISTS {table\_name}")

dbutils.fs.rm(checkpoint\_path, True)

# Configure Auto Loader to ingest JSON data to a Delta table

(spark.readStream

.format("cloudFiles")

.option("cloudFiles.format", "json")

.option("cloudFiles.schemaLocation", checkpoint\_path)

.load(file\_path)

.select("\*", col("\_metadata.file\_path").alias("source\_file"), current\_timestamp().alias("processing\_time"))

.writeStream

.option("checkpointLocation", checkpoint\_path)

.trigger(availableNow=True)

.toTable(table\_name))

**Scala**

ScalaCopy

// Imports

import org.apache.spark.sql.functions.current\_timestamp

import org.apache.spark.sql.streaming.Trigger

import spark.implicits.\_

// Define variables used in code below

val file\_path = "/databricks-datasets/structured-streaming/events"

val username = spark.sql("SELECT regexp\_replace(current\_user(), '[^a-zA-Z0-9]', '\_')").first.get(0)

val table\_name = s"${username}\_etl\_quickstart"

val checkpoint\_path = s"/tmp/${username}/\_checkpoint"

// Clear out data from previous demo execution

spark.sql(s"DROP TABLE IF EXISTS ${table\_name}")

dbutils.fs.rm(checkpoint\_path, true)

// Configure Auto Loader to ingest JSON data to a Delta table

spark.readStream

.format("cloudFiles")

.option("cloudFiles.format", "json")

.option("cloudFiles.schemaLocation", checkpoint\_path)

.load(file\_path)

.select($"\*", $"\_metadata.file\_path".as("source\_file"), current\_timestamp.as("processing\_time"))

.writeStream

.option("checkpointLocation", checkpoint\_path)

.trigger(Trigger.AvailableNow)

.toTable(table\_name)

**Note**

The variables defined in this code should allow you to safely execute it without risk of conflicting with existing workspace assets or other users. Restricted network or storage permissions will raise errors when executing this code; contact your workspace administrator to troubleshoot these restrictions.

To learn more about Auto Loader, see [What is Auto Loader?](https://learn.microsoft.com/en-us/azure/databricks/ingestion/auto-loader/).

**Step 4: Process and interact with data**

Notebooks execute logic cell-by-cell. To execute the logic in your cell:

1. To run the cell you completed in the previous step, select the cell and press **SHIFT+ENTER**.
2. To query the table you’ve just created, copy and paste the following code into an empty cell, then press **SHIFT+ENTER** to run the cell.

**Python**

PythonCopy

df = spark.read.table(table\_name)

**Scala**

ScalaCopy

val df = spark.read.table(table\_name)

1. To preview the data in your DataFrame, copy and paste the following code into an empty cell, then press **SHIFT+ENTER** to run the cell.

**Python**

PythonCopy

display(df)

**Scala**

ScalaCopy

display(df)

To learn more about interactive options for visualizing data, see [Visualizations in Databricks notebooks](https://learn.microsoft.com/en-us/azure/databricks/visualizations/).

**Step 5: Schedule a job**

You can run Databricks notebooks as production scripts by adding them as a task in a Databricks job. In this step, you will create a new job that you can trigger manually.

To schedule your notebook as a task:

1. Click **Schedule** on the right side of the header bar.
2. Enter a unique name for the **Job name**.
3. Click **Manual**.
4. In the **Cluster** drop-down, select the cluster you created in step 1.
5. Click **Create**.
6. In the window that appears, click **Run now**.
7. To see the job run results, click the External Link icon next to the **Last run** timestamp.

For more information on jobs, see [What is Azure Databricks Jobs?](https://learn.microsoft.com/en-us/azure/databricks/workflows/#what-is-jobs).

**Additional Integrations**

Learn more about integrations and tools for data engineering with Azure Databricks:

* [Connect your favorite IDE](https://learn.microsoft.com/en-us/azure/databricks/dev-tools/index-ide)
* [Use dbt with Databricks](https://learn.microsoft.com/en-us/azure/databricks/partners/prep/dbt)
* [Learn about the Databricks Command Line Interface (CLI)](https://learn.microsoft.com/en-us/azure/databricks/dev-tools/cli/)
* [Learn about the Databricks Terraform Provider](https://learn.microsoft.com/en-us/azure/databricks/dev-tools/terraform/)

Lab 2: Delta Lake - <https://learn.microsoft.com/en-us/azure/databricks/delta/tutorial>

**Tutorial: Delta Lake**

* Article
* 03/07/2024
* 4 contributors

**In this article**

1. [Create a table](https://learn.microsoft.com/en-us/azure/databricks/delta/tutorial#--create-a-table)
2. [Upsert to a table](https://learn.microsoft.com/en-us/azure/databricks/delta/tutorial#--upsert-to-a-table)
3. [Read a table](https://learn.microsoft.com/en-us/azure/databricks/delta/tutorial#--read-a-table)
4. [Write to a table](https://learn.microsoft.com/en-us/azure/databricks/delta/tutorial#--write-to-a-table)
5. [Update a table](https://learn.microsoft.com/en-us/azure/databricks/delta/tutorial#--update-a-table)
6. [Delete from a table](https://learn.microsoft.com/en-us/azure/databricks/delta/tutorial#--delete-from-a-table)
7. [Display table history](https://learn.microsoft.com/en-us/azure/databricks/delta/tutorial#--display-table-history)
8. [Query an earlier version of the table (time travel)](https://learn.microsoft.com/en-us/azure/databricks/delta/tutorial#--query-an-earlier-version-of-the-table-time-travel)
9. [Optimize a table](https://learn.microsoft.com/en-us/azure/databricks/delta/tutorial#--optimize-a-table)
10. [Z-order by columns](https://learn.microsoft.com/en-us/azure/databricks/delta/tutorial#--z-order-by-columns)
11. [Clean up snapshots with VACUUM](https://learn.microsoft.com/en-us/azure/databricks/delta/tutorial#--clean-up-snapshots-with-vacuum)

Show less

This tutorial introduces common Delta Lake operations on Azure Databricks, including the following:

* [Create a table.](https://learn.microsoft.com/en-us/azure/databricks/delta/tutorial#create)
* [Upsert to a table.](https://learn.microsoft.com/en-us/azure/databricks/delta/tutorial#upsert)
* [Read from a table.](https://learn.microsoft.com/en-us/azure/databricks/delta/tutorial#read)
* [Display table history.](https://learn.microsoft.com/en-us/azure/databricks/delta/tutorial#display-history)
* [Query an earlier version of a table.](https://learn.microsoft.com/en-us/azure/databricks/delta/tutorial#time-travel)
* [Optimize a table.](https://learn.microsoft.com/en-us/azure/databricks/delta/tutorial#optimize)
* [Add a Z-order index.](https://learn.microsoft.com/en-us/azure/databricks/delta/tutorial#z-order)
* [Vacuum unreferenced files.](https://learn.microsoft.com/en-us/azure/databricks/delta/tutorial#vacuum)

You can run the example Python, R, Scala, and SQL code in this article from within a [notebook](https://learn.microsoft.com/en-us/azure/databricks/notebooks/notebooks-manage) attached to an Azure Databricks [cluster](https://learn.microsoft.com/en-us/azure/databricks/compute/). You can also run the SQL code in this article from within a [query](https://learn.microsoft.com/en-us/azure/databricks/sql/user/sql-editor/) associated with a [SQL warehouse](https://learn.microsoft.com/en-us/azure/databricks/compute/sql-warehouse/create) in [Databricks SQL](https://learn.microsoft.com/en-us/azure/databricks/sql/).

**Note**

Some of the following code examples use a two-level namespace notation consisting of a schema (also called a database) and a table or view (for example, default.people10m). To use these examples with [**Unity Catalog**](https://learn.microsoft.com/en-us/azure/databricks/data-governance/unity-catalog/), replace the two-level namespace with Unity Catalog three-level namespace notation consisting of a catalog, schema, and table or view (for example, main.default.people10m).

**Create a table**

All tables created on Azure Databricks use Delta Lake by default.

**Note**

Delta Lake is the default for all reads, writes, and table creation commands Azure Databricks.

**Python**

PythonCopy

# Load the data from its source.

df = spark.read.load("/databricks-datasets/learning-spark-v2/people/people-10m.delta")

# Write the data to a table.

table\_name = "people\_10m"

df.write.saveAsTable(table\_name)

**R**

RCopy

library(SparkR)

sparkR.session()

# Load the data from its source.

df = read.df(path = "/databricks-datasets/learning-spark-v2/people/people-10m.delta")

# Write the data to a table.

table\_name = "people\_10m"

saveAsTable(

df = df,

tableName = table\_name

)

**Scala**

ScalaCopy

// Load the data from its source.

val people = spark.read.load("/databricks-datasets/learning-spark-v2/people/people-10m.delta")

// Write the data to a table.

val table\_name = "people\_10m"

people.write.saveAsTable("people\_10m")

**SQL**

SQLCopy

DROP TABLE IF EXISTS people\_10m;

CREATE TABLE IF NOT EXISTS people\_10m

AS SELECT \* FROM delta.`/databricks-datasets/learning-spark-v2/people/people-10m.delta`;

The preceding operations create a new [managed table](https://learn.microsoft.com/en-us/azure/databricks/lakehouse/data-objects#managed-table) by using the schema that was inferred from the data. For information about available options when you create a Delta table, see [CREATE TABLE](https://learn.microsoft.com/en-us/azure/databricks/sql/language-manual/sql-ref-syntax-ddl-create-table).

For managed tables, Azure Databricks determines the location for the data. To get the location, you can use the [DESCRIBE DETAIL](https://learn.microsoft.com/en-us/azure/databricks/delta/table-details) statement, for example:

**Python**

PythonCopy

display(spark.sql('DESCRIBE DETAIL people\_10m'))

**R**

RCopy

display(sql("DESCRIBE DETAIL people\_10m"))

**Scala**

ScalaCopy

display(spark.sql("DESCRIBE DETAIL people\_10m"))

**SQL**

SQLCopy

DESCRIBE DETAIL people\_10m;

Sometimes you may want to create a table by specifying the schema before inserting data. You can complete this with the following SQL commands:

SQLCopy

CREATE TABLE IF NOT EXISTS people10m (

id INT,

firstName STRING,

middleName STRING,

lastName STRING,

gender STRING,

birthDate TIMESTAMP,

ssn STRING,

salary INT

)

CREATE OR REPLACE TABLE people10m (

id INT,

firstName STRING,

middleName STRING,

lastName STRING,

gender STRING,

birthDate TIMESTAMP,

ssn STRING,

salary INT

)

In Databricks Runtime 13.0 and above, you can use CREATE TABLE LIKE to create a new empty Delta table that duplicates the schema and table properties for a source Delta table. This can be especially useful when promoting tables from a development environment into production, such as in the following code example:

SQLCopy

CREATE TABLE prod.people10m LIKE dev.people10m

You can also use the DeltaTableBuilder API in Delta Lake to create tables. Compared to the DataFrameWriter APIs, this API makes it easier to specify additional information like column comments, table properties, and [generated columns](https://learn.microsoft.com/en-us/azure/databricks/delta/generated-columns).

**Important**

This feature is in [**Public Preview**](https://learn.microsoft.com/en-us/azure/databricks/release-notes/release-types).

**Python**

PythonCopy

# Create table in the metastore

DeltaTable.createIfNotExists(spark) \

.tableName("default.people10m") \

.addColumn("id", "INT") \

.addColumn("firstName", "STRING") \

.addColumn("middleName", "STRING") \

.addColumn("lastName", "STRING", comment = "surname") \

.addColumn("gender", "STRING") \

.addColumn("birthDate", "TIMESTAMP") \

.addColumn("ssn", "STRING") \

.addColumn("salary", "INT") \

.execute()

# Create or replace table with path and add properties

DeltaTable.createOrReplace(spark) \

.addColumn("id", "INT") \

.addColumn("firstName", "STRING") \

.addColumn("middleName", "STRING") \

.addColumn("lastName", "STRING", comment = "surname") \

.addColumn("gender", "STRING") \

.addColumn("birthDate", "TIMESTAMP") \

.addColumn("ssn", "STRING") \

.addColumn("salary", "INT") \

.property("description", "table with people data") \

.location("/tmp/delta/people10m") \

.execute()

**Scala**

ScalaCopy

// Create table in the metastore

DeltaTable.createOrReplace(spark)

.tableName("default.people10m")

.addColumn("id", "INT")

.addColumn("firstName", "STRING")

.addColumn("middleName", "STRING")

.addColumn(

DeltaTable.columnBuilder("lastName")

.dataType("STRING")

.comment("surname")

.build())

.addColumn("lastName", "STRING", comment = "surname")

.addColumn("gender", "STRING")

.addColumn("birthDate", "TIMESTAMP")

.addColumn("ssn", "STRING")

.addColumn("salary", "INT")

.execute()

// Create or replace table with path and add properties

DeltaTable.createOrReplace(spark)

.addColumn("id", "INT")

.addColumn("firstName", "STRING")

.addColumn("middleName", "STRING")

.addColumn(

DeltaTable.columnBuilder("lastName")

.dataType("STRING")

.comment("surname")

.build())

.addColumn("lastName", "STRING", comment = "surname")

.addColumn("gender", "STRING")

.addColumn("birthDate", "TIMESTAMP")

.addColumn("ssn", "STRING")

.addColumn("salary", "INT")

.property("description", "table with people data")

.location("/tmp/delta/people10m")

.execute()

**Upsert to a table**

To merge a set of updates and insertions into an existing Delta table, you use the [MERGE INTO](https://learn.microsoft.com/en-us/azure/databricks/sql/language-manual/delta-merge-into) statement. For example, the following statement takes data from the source table and merges it into the target Delta table. When there is a matching row in both tables, Delta Lake updates the data column using the given expression. When there is no matching row, Delta Lake adds a new row. This operation is known as an *upsert*.

SQLCopy

CREATE OR REPLACE TEMP VIEW people\_updates (

id, firstName, middleName, lastName, gender, birthDate, ssn, salary

) AS VALUES

(9999998, 'Billy', 'Tommie', 'Luppitt', 'M', '1992-09-17T04:00:00.000+0000', '953-38-9452', 55250),

(9999999, 'Elias', 'Cyril', 'Leadbetter', 'M', '1984-05-22T04:00:00.000+0000', '906-51-2137', 48500),

(10000000, 'Joshua', 'Chas', 'Broggio', 'M', '1968-07-22T04:00:00.000+0000', '988-61-6247', 90000),

(20000001, 'John', '', 'Doe', 'M', '1978-01-14T04:00:00.000+000', '345-67-8901', 55500),

(20000002, 'Mary', '', 'Smith', 'F', '1982-10-29T01:00:00.000+000', '456-78-9012', 98250),

(20000003, 'Jane', '', 'Doe', 'F', '1981-06-25T04:00:00.000+000', '567-89-0123', 89900);

MERGE INTO people\_10m

USING people\_updates

ON people\_10m.id = people\_updates.id

WHEN MATCHED THEN UPDATE SET \*

WHEN NOT MATCHED THEN INSERT \*;

If you specify \*, this updates or inserts all columns in the target table. This assumes that the source table has the same columns as those in the target table, otherwise the query will throw an analysis error.

You must specify a value for every column in your table when you perform an INSERT operation (for example, when there is no matching row in the existing dataset). However, you do not need to update all values.

To see the results, query the table.

SQLCopy

SELECT \* FROM people\_10m WHERE id >= 9999998

**Read a table**

You access data in Delta tables by the table name or the table path, as shown in the following examples:

**Python**

PythonCopy

people\_df = spark.read.table(table\_name)

display(people\_df)

## or

people\_df = spark.read.load(table\_path)

display(people\_df)

**R**

RCopy

people\_df = tableToDF(table\_name)

display(people\_df)

**Scala**

ScalaCopy

val people\_df = spark.read.table(table\_name)

display(people\_df)

\\ or

val people\_df = spark.read.load(table\_path)

display(people\_df)

**SQL**

SQLCopy

SELECT \* FROM people\_10m;

SELECT \* FROM delta.`<path-to-table`;

**Write to a table**

Delta Lake uses standard syntax for writing data to tables.

To atomically add new data to an existing Delta table, use append mode as in the following examples:

**SQL**

SQLCopy

INSERT INTO people10m SELECT \* FROM more\_people

**Python**

PythonCopy

df.write.mode("append").saveAsTable("people10m")

**Scala**

ScalaCopy

df.write.mode("append").saveAsTable("people10m")

To atomically replace all the data in a table, use overwrite mode as in the following examples:

**SQL**

SQLCopy

INSERT OVERWRITE TABLE people10m SELECT \* FROM more\_people

**Python**

PythonCopy

df.write.mode("overwrite").saveAsTable("people10m")

**Scala**

ScalaCopy

df.write.mode("overwrite").saveAsTable("people10m")

**Update a table**

You can update data that matches a predicate in a Delta table. For example, in a table named people10m or a path at /tmp/delta/people-10m, to change an abbreviation in the gender column from M or F to Male or Female, you can run the following:

**SQL**

SQLCopy

UPDATE people10m SET gender = 'Female' WHERE gender = 'F';

UPDATE people10m SET gender = 'Male' WHERE gender = 'M';

UPDATE delta.`/tmp/delta/people-10m` SET gender = 'Female' WHERE gender = 'F';

UPDATE delta.`/tmp/delta/people-10m` SET gender = 'Male' WHERE gender = 'M';

**Python**

PythonCopy

from delta.tables import \*

from pyspark.sql.functions import \*

deltaTable = DeltaTable.forPath(spark, '/tmp/delta/people-10m')

# Declare the predicate by using a SQL-formatted string.

deltaTable.update(

condition = "gender = 'F'",

set = { "gender": "'Female'" }

)

# Declare the predicate by using Spark SQL functions.

deltaTable.update(

condition = col('gender') == 'M',

set = { 'gender': lit('Male') }

)

**Scala**

ScalaCopy

import io.delta.tables.\_

val deltaTable = DeltaTable.forPath(spark, "/tmp/delta/people-10m")

// Declare the predicate by using a SQL-formatted string.

deltaTable.updateExpr(

"gender = 'F'",

Map("gender" -> "'Female'")

import org.apache.spark.sql.functions.\_

import spark.implicits.\_

// Declare the predicate by using Spark SQL functions and implicits.

deltaTable.update(

col("gender") === "M",

Map("gender" -> lit("Male")));

**Delete from a table**

You can remove data that matches a predicate from a Delta table. For instance, in a table named people10m or a path at /tmp/delta/people-10m, to delete all rows corresponding to people with a value in the birthDatecolumn from before 1955, you can run the following:

**SQL**

SQLCopy

DELETE FROM people10m WHERE birthDate < '1955-01-01'

DELETE FROM delta.`/tmp/delta/people-10m` WHERE birthDate < '1955-01-01'

**Python**

PythonCopy

from delta.tables import \*

from pyspark.sql.functions import \*

deltaTable = DeltaTable.forPath(spark, '/tmp/delta/people-10m')

# Declare the predicate by using a SQL-formatted string.

deltaTable.delete("birthDate < '1955-01-01'")

# Declare the predicate by using Spark SQL functions.

deltaTable.delete(col('birthDate') < '1960-01-01')

**Scala**

ScalaCopy

import io.delta.tables.\_

val deltaTable = DeltaTable.forPath(spark, "/tmp/delta/people-10m")

// Declare the predicate by using a SQL-formatted string.

deltaTable.delete("birthDate < '1955-01-01'")

import org.apache.spark.sql.functions.\_

import spark.implicits.\_

// Declare the predicate by using Spark SQL functions and implicits.

deltaTable.delete(col("birthDate") < "1955-01-01")

**Important**

delete removes the data from the latest version of the Delta table but does not remove it from the physical storage until the old versions are explicitly vacuumed. See [**vacuum**](https://learn.microsoft.com/en-us/azure/databricks/delta/vacuum) for details.

**Display table history**

To view the history of a table, use the [DESCRIBE HISTORY](https://learn.microsoft.com/en-us/azure/databricks/delta/history) statement, which provides provenance information, including the table version, operation, user, and so on, for each write to a table.

SQLCopy

DESCRIBE HISTORY people\_10m

**Query an earlier version of the table (time travel)**

Delta Lake time travel allows you to query an older snapshot of a Delta table.

To query an older version of a table, specify a version or timestamp in a SELECT statement. For example, to query version 0 from the history above, use:

SQLCopy

SELECT \* FROM people\_10m VERSION AS OF 0

or

SQLCopy

SELECT \* FROM people\_10m TIMESTAMP AS OF '2019-01-29 00:37:58'

For timestamps, only date or timestamp strings are accepted, for example, "2019-01-01" and "2019-01-01'T'00:00:00.000Z".

DataFrameReader options allow you to create a DataFrame from a Delta table that is fixed to a specific version of the table, for example in Python:

PythonCopy

df1 = spark.read.format('delta').option('timestampAsOf', '2019-01-01').table("people\_10m")

display(df1)

or, alternately:

PythonCopy

df2 = spark.read.format('delta').option('versionAsOf', 0).table("people\_10m")

display(df2)

For details, see [Work with Delta Lake table history](https://learn.microsoft.com/en-us/azure/databricks/delta/history).

**Optimize a table**

Once you have performed multiple changes to a table, you might have a lot of small files. To improve the speed of read queries, you can use OPTIMIZE to collapse small files into larger ones:

SQLCopy

OPTIMIZE people\_10m

**Z-order by columns**

To improve read performance further, you can co-locate related information in the same set of files by Z-Ordering. This co-locality is automatically used by Delta Lake data-skipping algorithms to dramatically reduce the amount of data that needs to be read. To Z-Order data, you specify the columns to order on in the ZORDER BY clause. For example, to co-locate by gender, run:

SQLCopy

OPTIMIZE people\_10m

ZORDER BY (gender)

For the full set of options available when running OPTIMIZE, see [Compact data files with optimize on Delta Lake](https://learn.microsoft.com/en-us/azure/databricks/delta/optimize).

**Clean up snapshots with VACUUM**

Delta Lake provides snapshot isolation for reads, which means that it is safe to run OPTIMIZE even while other users or jobs are querying the table. Eventually however, you should clean up old snapshots. You can do this by running the VACUUM command:

SQLCopy

VACUUM people\_10m

For details on using VACUUM effectively, see [Remove unused data files with vacuum](https://learn.microsoft.com/en-us/azure/databricks/delta/vacuum).

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Lab 3: Tutorial: Load and transform data in PySpark DataFrames -** <https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#language-Python>

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Tutorial: Load and transform data using Apache Spark DataFrames**

* Article
* 04/15/2024
* 2 contributors

**In this article**

1. [Python](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#python)
2. [Scala](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#scala)
3. [R](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#r)
4. [What is a DataFrame?](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#--what-is-a-dataframe)
5. [Requirements](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#requirements)
6. [Step 1: Define variables and load CSV file](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#--step-1-define-variables-and-load-csv-file)
7. [Step 2: Create a DataFrame](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#--step-2-create-a-dataframe)
8. [Step 3: Load data into a DataFrame from CSV file](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#--step-3-load-data-into-a-dataframe-from-csv-file)
9. [Step 4: View and interact with your DataFrame](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#--step-4-view-and-interact-with-your-dataframe)
10. [Step 5: Save the DataFrame](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#--step-5-save-the-dataframe)
11. [Additional tasks: Run SQL queries in PySpark, Scala, and R](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#--additional-tasks-run-sql-queries-in-pyspark-scala-and-r)
12. [DataFrame tutorial notebook](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#dataframe-tutorial-notebook)
13. [Additional resources](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#additional-resources)

This tutorial shows you how to load and transform data using the Apache Spark Python (PySpark) DataFrame API, the Apache Spark Scala DataFrame API, and the SparkR SparkDataFrame API in Azure Databricks.

By the end of this tutorial, you will understand what a DataFrame is and be familiar with the following tasks:

**Python**

* [Define variables and copy public data into a Unity Catalog volume](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#define-variables)
* [Create a DataFrame with Python](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#create-dataframe)
* [Load data into a DataFrame from CSV file](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#create-dataframe-from-csv)
* [View and interact with a DataFrame](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#interact-with-dataframe)
* [Save the DataFrame](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#save-dataframe)
* [Run SQL queries in PySpark](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#run-sql)

See also [Apache Spark PySpark API reference](https://api-docs.databricks.com/python/pyspark/latest/pyspark.sql/api/pyspark.sql.DataFrame.html#pyspark-sql-dataframe).

**Scala**

* [Define variables and copy public data into a Unity Catalog volume](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#define-variables)
* [Create a DataFrame with Scala](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#create-dataframe)
* [Load data into a DataFrame from CSV file](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#create-dataframe-from-csv)
* [View and interacting with a DataFrame](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#interact-with-dataframe)
* [Save the DataFrame](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#save-dataframe)
* [Run SQL queries in Apache Spark](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#run-sql)

See also [Apache Spark Scala API reference](https://api-docs.databricks.com/scala/spark/latest/org/apache/spark/index.html).

**R**

* [Define variables and copy public data into a Unity Catalog volume](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#define-variables)
* [Create a SparkR SparkDataFrames](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#create-dataframe)
* [Load data into a DataFrame from CSV file](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#create-dataframe-from-csv)
* [View and interact with a DataFrame](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#interact-with-dataframe)
* [Save the DataFrame](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#save-dataframe)
* [Run SQL queries in SparkR](https://learn.microsoft.com/en-us/azure/databricks/getting-started/dataframes#run-sql)

See also [Apache SparkR API reference](https://spark.apache.org/docs/latest/sparkr.html#sparkdataframe).

**What is a DataFrame?**

A DataFrame is a two-dimensional labeled data structure with columns of potentially different types. You can think of a DataFrame like a spreadsheet, a SQL table, or a dictionary of series objects. Apache Spark DataFrames provide a rich set of functions (select columns, filter, join, aggregate) that allow you to solve common data analysis problems efficiently.

Apache Spark DataFrames are an abstraction built on top of Resilient Distributed Datasets (RDDs). Spark DataFrames and Spark SQL use a unified planning and optimization engine, allowing you to get nearly identical performance across all supported languages on Azure Databricks (Python, SQL, Scala, and R).

**Requirements**

To complete the following tutorial, you must meet the following requirements:

* To use the examples in this tutorial, your workspace must have [Unity Catalog](https://learn.microsoft.com/en-us/azure/databricks/data-governance/unity-catalog/) enabled.
* The examples in this tutorial use a Unity Catalog [volume](https://learn.microsoft.com/en-us/azure/databricks/connect/unity-catalog/volumes) to store sample data. To use these examples, create a volume and use that volume’s catalog, schema, and volume names to set the volume path used by the examples.
* You must have the following permissions in Unity Catalog:
  + READ VOLUME and WRITE VOLUME, or ALL PRIVILEGES for the volume used for this tutorial.
  + USE SCHEMA or ALL PRIVILEGES for the schema used for this tutorial.
  + USE CATALOG or ALL PRIVILEGES for the catalog used for this tutorial.

To set these permissions, see your Databricks administrator or [Unity Catalog privileges and securable objects](https://learn.microsoft.com/en-us/azure/databricks/data-governance/unity-catalog/manage-privileges/privileges).

**Step 1: Define variables and load CSV file**

This step defines variables for use in this tutorial and then loads a CSV file containing baby name data from [health.data.ny.gov](https://health.data.ny.gov/api/views/jxy9-yhdk/rows.csv) into your Unity Catalog volume.

1. Open a new notebook by clicking the New Icon icon. To learn how to navigate Azure Databricks notebooks, see [Databricks notebook interface and controls](https://learn.microsoft.com/en-us/azure/databricks/notebooks/notebook-ui).
2. Copy and paste the following code into the new empty notebook cell. Replace <catalog-name>, <schema-name>, and <volume-name> with the catalog, schema, and volume names for a Unity Catalog volume. Replace <table\_name> with a table name of your choice. You will load baby name data into this table later in this tutorial.
3. Press Shift+Enter to run the cell and create a new blank cell.

**Python**

PythonCopy

catalog = "<catalog\_name>"

schema = "<schema\_name>"

volume = "<volume\_name>"

download\_url = "https://health.data.ny.gov/api/views/jxy9-yhdk/rows.csv"

file\_name = "rows.csv"

table\_name = "<table\_name>"

path\_volume = "/Volumes/" + catalog + "/" + schema + "/" + volume

path\_tables = catalog + "." + schema

print(path\_tables) # Show the complete path

print(path\_volume) # Show the complete path

**Scala**

ScalaCopy

val catalog = "<catalog\_name>"

val schema = "<schema\_name>"

val volume = "<volume\_name>"

val download\_url = "https://health.data.ny.gov/api/views/jxy9-yhdk/rows.csv"

val file\_name = "rows.csv"

val table\_name = "<table\_name>"

val path\_volume = s"/Volumes/$catalog/$schema/$volume"

val path\_tables = s"$catalog.$schema.$table\_name"

print(path\_volume) // Show the complete path

print(path\_tables) // Show the complete path

**R**

RCopy

catalog <- "<catalog\_name>"

schema <- "<schema\_name>"

volume <- "<volume\_name>"

download\_url <- "https://health.data.ny.gov/api/views/jxy9-yhdk/rows.csv"

file\_name <- "rows.csv"

table\_name <- "<table\_name>"

path\_volume <- paste("/Volumes/", catalog, "/", schema, "/", volume, sep = "")

path\_tables <- paste(catalog, ".", schema, sep = "")

print(path\_volume) # Show the complete path

print(path\_tables) # Show the complete path

1. Copy and paste the following code into the new empty notebook cell. This code copies the rows.csv file from [health.data.ny.gov](https://health.data.ny.gov/api/views/jxy9-yhdk/rows.csv) into your Unity Catalog volume using the [Databricks dbutuils](https://learn.microsoft.com/en-us/azure/databricks/dev-tools/databricks-utils#cp-command-dbutilsfscp) command.
2. Press Shift+Enter to run the cell and then move to the next cell.

**Python**

PythonCopy

dbutils.fs.cp(f"{download\_url}", f"{path\_volume}" + "/" + f"{file\_name}")

**Scala**

ScalaCopy

dbutils.fs.cp(download\_url, s"$path\_volume/$file\_name")

**R**

RCopy

dbutils.fs.cp(download\_url, paste(path\_volume, "/", file\_name, sep = ""))

**Step 2: Create a DataFrame**

This step creates a DataFrame named df1 with test data and then displays its contents.

1. Copy and paste the following code into the new empty notebook cell. This code creates the Dataframe with test data, and then displays the contents and the schema of the DataFrame.
2. Press Shift+Enter to run the cell and then move to the next cell.

**Python**

PythonCopy

data = [[2021, "test", "Albany", "M", 42]]

columns = ["Year", "First\_Name", "County", "Sex", "Count"]

df1 = spark.createDataFrame(data,

schema="Year int,

First\_Name STRING,

County STRING,

Sex STRING,

Count int")

display(df1) # The display() method is specific to Databricks notebooks and provides a richer visualization.

# df1.show() The show() method is a part of the Apache Spark DataFrame API and provides basic visualization.

**Scala**

ScalaCopy

val data = Seq((2021, "test", "Albany", "M", 42))

val columns = Seq("Year", "First\_Name", "County", "Sex", "Count")

val df1 = data.toDF(columns: \_\*)

display(df1) // The display() method is specific to Databricks notebooks and provides a richer visualization.

// df1.show() The show() method is a part of the Apache Spark DataFrame API and provides basic visualization.

**R**

RCopy

# Load the SparkR package that is already preinstalled on the cluster.

library(SparkR)

data <- data.frame(

Year = c(2021),

First\_Name = c("test"),

County = c("Albany"),

Sex = c("M"),

Count = c(42)

)

df1 <- createDataFrame(data)

display(df1) # The display() method is specific to Databricks notebooks and provides a richer visualization.

# head(df1) The head() method is a part of the Apache SparkR DataFrame API and provides basic visualization.

**Step 3: Load data into a DataFrame from CSV file**

This step creates a DataFrame named df\_csv from the CSV file that you previously loaded into your Unity Catalog volume. See [spark.read.csv](https://spark.apache.org/docs/latest/sql-data-sources-csv.html).

1. Copy and paste the following code into the new empty notebook cell. This code loads baby name data into DataFrame df\_csv from the CSV file and then displays the contents of the DataFrame.
2. Press Shift+Enter to run the cell and then move to the next cell.

**Python**

PythonCopy

df\_csv = spark.read.csv(f"{path\_volume}/{file\_name}",

header=True,

inferSchema=True,

sep=",")

display(df\_csv)

**Scala**

ScalaCopy

val df\_csv = spark.read

.option("header", "true")

.option("inferSchema", "true")

.option("delimiter", ",")

.csv(s"$path\_volume/$file\_name")

display(df\_csv)

**R**

RCopy

df\_csv <- read.df(paste(path\_volume, "/", file\_name, sep=""),

source="csv",

header = TRUE,

inferSchema = TRUE,

delimiter = ",")

display(df\_csv)

You can load data from many [supported file formats](https://learn.microsoft.com/en-us/azure/databricks/query/formats/).

**Step 4: View and interact with your DataFrame**

View and interact with your baby names DataFrames using the following methods.

**Print the DataFrame schema**

Learn how to display the schema of an Apache Spark DataFrame. Apache Spark uses the term *schema* to refer to the names and data types of the columns in the DataFrame.

Copy and paste the following code into an empty notebook cell. This code shows the schema of your DataFrames with the .printSchema() method to view the schemas of the two DataFrames - to prepare to union the two DataFrames.

**Python**

PythonCopy

df\_csv.printSchema()

df1.printSchema()

**Scala**

ScalaCopy

df\_csv.printSchema()

df1.printSchema()

**R**

RCopy

printSchema(df\_csv)

printSchema(df1)

**Note**

Azure Databricks also uses the term schema to describe a collection of tables registered to a catalog.

**Rename column in the DataFrame**

Learn how to rename a column in a DataFrame.

Copy and paste the following code into an empty notebook cell. This code renames a column in the df1\_csvDataFrame to match the respective column in the df1 DataFrame. This code uses the Apache Spark withColumnRenamed() method.

**Python**

PythonCopy

df\_csv = df\_csv.withColumnRenamed("First Name", "First\_Name")

df\_csv.printSchema

**Scala**

ScalaCopy

val df\_csvRenamed = df\_csv.withColumnRenamed("First Name", "First\_Name")

// when modifying a DataFrame in Scala, you must assign it to a new variable

df\_csv\_renamed.printSchema()

**R**

RCopy

df\_csv <- withColumnRenamed(df\_csv, "First Name", "First\_Name")

printSchema(df\_csv)

**Combine DataFrames**

Learn how to create a new DataFrame that adds the rows of one DataFrame to another.

Copy and paste the following code into an empty notebook cell. This code uses the Apache Spark union()method to combine the contents of your first DataFrame df with DataFrame df\_csv containing the baby names data loaded from the CSV file.

**Python**

PythonCopy

df = df1.union(df\_csv)

display(df)

**Scala**

ScalaCopy

val df = df1.union(df\_csv\_renamed)

display(df)

**R**

RCopy

display(df <- union(df1, df\_csv))

**Filter rows in a DataFrame**

Discover the most popular baby names in your data set by filtering rows, using the Apache Spark .filter() or .where() methods. Use filtering to select a subset of rows to return or modify in a DataFrame. There is no difference in performance or syntax, as seen in the following examples.

**Using .filter() method**

Copy and paste the following code into an empty notebook cell. This code uses the the Apache Spark .filter()method to display those rows in the DataFrame with a count of more than 50.

**Python**

PythonCopy

display(df.filter(df["Count"] > 50))

**Scala**

ScalaCopy

display(df.filter(df("Count") > 50))

**R**

RCopy

display(filteredDF <- filter(df, df$Count > 50))

**Using .where() method**

Copy and paste the following code into an empty notebook cell. This code uses the the Apache Spark .where()method to display those rows in the DataFrame with a count of more than 50.

**Python**

PythonCopy

display(df.where(df["Count"] > 50))

**Scala**

ScalaCopy

display(df.where(df("Count") > 50))

**R**

RCopy

display(filtered\_df <- where(df, df$Count > 50))

**Select columns from a DataFrame and order by frequency**

Learn about which baby name frequency with the select() method to specify the columns from the DataFrame to return. Use the Apache Spark orderby and desc functions to order the results.

The [pyspark.sql](https://spark.apache.org/docs/2.4.0/api/python/pyspark.sql.html) module for Apache Spark provides support for SQL functions. Among these functions that we use in this tutorial are the the Apache Spark orderBy(), desc(), and expr() functions. You enable the use of these functions by importing them into your session as needed.

Copy and paste the following code into an empty notebook cell. This code imports the desc() function and then uses the Apache Spark select() method and Apache Spark orderBy() and desc() functions to display the most common names and their counts in descending order.

**Python**

PythonCopy

from pyspark.sql.functions import desc

display(df.select("First\_Name", "Count").orderBy(desc("Count")))

**Scala**

ScalaCopy

import org.apache.spark.sql.functions.desc

display(df.select("First\_Name", "Count").orderBy(desc("Count")))

**R**

RCopy

display(arrange(select(df, df$First\_Name, df$Count), desc(df$Count)))

**Create a subset DataFrame**

Learn how to create a subset DataFrame from an existing DataFrame.

Copy and paste the following code into an empty notebook cell. This code uses the Apache Spark filtermethod to create a new DataFrame restricting the data by year, count, and sex. It uses the Apache Spark select() method to limit the columns. It also uses the Apache Spark orderBy() and desc() functions to sort the new DataFrame by count.

**Python**

PythonCopy

subsetDF = df.filter((df["Year"] == 2009) & (df["Count"] > 100) & (df["Sex"] == "F")).select("First\_Name", "County", "Count").orderBy(desc("Count"))

display(subsetDF)

**Scala**

ScalaCopy

val subsetDF = df.filter((df("Year") == 2009) && (df("Count") > 100) && (df("Sex") == "F")).select("First\_Name", "County", "Count").orderBy(desc("Count"))

display(subsetDF)

**R**

RCopy

subsetDF <- select(filter(df, (df$Count > 100) & (df$year == 2009) & df["Sex"] == "F")), "First\_Name", "County", "Count")

display(subsetDF)

**Step 5: Save the DataFrame**

Learn how to save a DataFrame,. You can either save your DataFrame to a table or write the DataFrame to a file or multiple files.

**Save the DataFrame to a table**

Azure Databricks uses the Delta Lake format for all tables by default. To save your DataFrame, you must have CREATE table privileges on the catalog and schema.

Copy and paste the following code into an empty notebook cell. This code saves the contents of the DataFrame to a table using the variable you defined at the start of this tutorial.

**Python**

PythonCopy

df.write.saveAsTable(s"$path\_tables" + "." + s"$table\_name")

# To overwrite an existing table, use the following code:

# df.write.mode("overwrite").saveAsTable(fs"$path\_tables" + "." + s"$table\_name")

**Scala**

ScalaCopy

df.write.saveAsTable(s"$path\_tables" + "." + s"$table\_name")

// To overwrite an existing table, use the following code:

// df.write.mode("overwrite").saveAsTable(s"$path\_volume" + "." + s"$table\_name")

**R**

RCopy

saveAsTable(df, paste(path\_tables, ".", table\_name))

# To overwrite an existing table, use the following code:

# saveAsTable(df, paste(path\_tables, ".", table\_name), mode = "overwrite")

Most Apache Spark applications work on large data sets and in a distributed fashion. Apache Spark writes out a directory of files rather than a single file. Delta Lake splits the Parquet folders and files. Many data systems can read these directories of files. Azure Databricks recommends using tables over file paths for most applications.

**Save the DataFrame to JSON files**

Copy and paste the following code into an empty notebook cell. This code saves the DataFrame to a directory of JSON files.

**Python**

PythonCopy

df.write.format("json").save("/tmp/json\_data")

# To overwrite an existing file, use the following code:

# df.write.format("json").mode("overwrite").save("/tmp/json\_data")

**Scala**

ScalaCopy

df.write.format("json").save("/tmp/json\_data")

// To overwrite an existing file, use the following code:

// df.write.format("json").mode("overwrite").save("/tmp/json\_data")

**R**

RCopy

write.df(df, path = "/tmp/json\_data", source = "json")

# To overwrite an existing file, use the following code:

# write.df(df, path = "/tmp/json\_data", source = "json", mode = "overwrite")

**Read the DataFrame from a JSON file**

Learn how to use the Apache Spark spark.read.format() method to read JSON data from a directory into a DataFrame.

Copy and paste the following code into an empty notebook cell. This code displays the JSON files you saved in the previous example.

**Python**

PythonCopy

display(spark.read.format("json").json("/tmp/json\_data"))

**Scala**

ScalaCopy

display(spark.read.format("json").json("/tmp/json\_data"))

**R**

RCopy

display(read.json("/tmp/json\_data"))

**Additional tasks: Run SQL queries in PySpark, Scala, and R**

Apache Spark DataFrames provide the following options to combine SQL with PySpark, Scala, and R. You can run the following code in the same notebook that you created for this tutorial.

**Specify a column as a SQL query**

Learn how to use the Apache Spark selectExpr() method. This is a variant of the select() method that accepts SQL expressions and return an updated DataFrame. This method allows you to use a SQL expression, such as upper.

Copy and paste the following code into an empty notebook cell. This code uses the Apache Spark selectExpr()method and the SQL upper expression to convert a string column to upper case (and rename the column).

**Python**

PythonCopy

display(df.selectExpr("Count", "upper(County) as big\_name"))

**Scala**

ScalaCopy

display(df.selectExpr("Count", "upper(County) as big\_name"))

**R**

RCopy

display(df\_selected <- selectExpr(df, "Count", "upper(County) as big\_name"))

**Use expr() to use SQL syntax for a column**

Learn how to import and use the Apache Spark expr() function to use SQL syntax anywhere a column would be specified.

Copy and paste the following code into an empty notebook cell. This code imports the expr() function and then uses the Apache Spark expr() function and the SQL lower expression to convert a string column to lower case (and rename the column).

**Python**

PythonCopy

from pyspark.sql.functions import expr

display(df.select("Count", expr("lower(County) as little\_name")))

**Scala**

ScalaCopy

import org.apache.spark.sql.functions.{col, expr} // Scala requires us to import the col() function as well as the expr() function

display(df.select(col("Count"), expr("lower(County) as little\_name")))

**R**

RCopy

display(df\_selected <- selectExpr(df, "Count", "lower(County) as little\_name"))

# expr() function is not supported in R, selectExpr in SparkR replicates this functionality

**Run an arbitrary SQL query using spark.sql() function**

Learn how to use the Apache Spark spark.sql() function to run arbitrary SQL queries.

Copy and paste the following code into an empty notebook cell. This code uses the Apache Spark spark.sql()function to query a SQL table using SQL syntax.

**Python**

PythonCopy

display(spark.sql(f"SELECT \* FROM {path\_tables}" + "." + f"{table\_name}"))

**Scala**

ScalaCopy

display(spark.sql(s"SELECT \* FROM $path\_tables.$table\_name"))

**R**

RCopy

display(sql(paste("SELECT \* FROM", path\_tables, ".", table\_name)))

**DataFrame tutorial notebook**

The following notebook includes the examples queries from this tutorial.

**Python**

**DataFrames tutorial notebook**

[Get notebook](https://learn.microsoft.com/en-us/azure/databricks/_extras/notebooks/source/tutorial-uc-spark-dataframe-python.html)

**Scala**

**DataFrames tutorial notebook**

[Get notebook](https://learn.microsoft.com/en-us/azure/databricks/_extras/notebooks/source/tutorial-uc-spark-dataframe-scala.html)

**R**

**DataFrames tutorial notebook**

[Get notebook](https://learn.microsoft.com/en-us/azure/databricks/_extras/notebooks/source/tutorial-uc-spark-dataframe-sparkr.html)

**Additional resources**

* [Apache Spark API reference](https://learn.microsoft.com/en-us/azure/databricks/reference/spark)
* [Convert between PySpark and pandas DataFrames](https://learn.microsoft.com/en-us/azure/databricks/pandas/pyspark-pandas-conversion)
* [Pandas API on Spark](https://learn.microsoft.com/en-us/azure/databricks/pandas/pandas-on-spark)

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Lab 4: Use SQL Warehouses in Azure Databricks**

<https://learn.microsoft.com/en-us/training/modules/use-sql-warehouses-azure-databricks/>

---------------------------------------------------------------------------------------------------------------------

**Use SQL Warehouses in Azure Databricks**

* 44 min
* Module
* 7 Units

Intermediate

Data Engineer

Azure Databricks

Azure Databricks provides SQL Warehouses that enable data analysts to work with data using familiar relational SQL queries.

**Learning objectives**

In this module, you'll learn how to:

* Create and configure SQL Warehouses in Azure Databricks.
* Create databases and tables.
* Create queries and dashboards.

**Prerequisites**

Before starting this module, you should have a basic knowledge of Azure Databricks. Consider completing the [Explore Azure Databricks](https://learn.microsoft.com/en-us/training/modules/explore-azure-databricks) module before this one.

**This module is part of these learning paths**

[Implement a Data Analytics Solution with Azure Databricks](https://learn.microsoft.com/training/paths/data-engineer-azure-databricks/)

* [Introduction](https://learn.microsoft.com/en-us/training/modules/use-sql-warehouses-azure-databricks/01-introduction)1 min
* [Get started with SQL Warehouses](https://learn.microsoft.com/en-us/training/modules/use-sql-warehouses-azure-databricks/02-sql-warehouses)3 min
* [Create databases and tables](https://learn.microsoft.com/en-us/training/modules/use-sql-warehouses-azure-databricks/03-databases-tables)3 min
* [Create queries and dashboards](https://learn.microsoft.com/en-us/training/modules/use-sql-warehouses-azure-databricks/04-queries-dashboards)3 min
* [Exercise - Use a SQL Warehouse in Azure Databricks](https://learn.microsoft.com/en-us/training/modules/use-sql-warehouses-azure-databricks/05-exercise-databricks-sql)30 min
* [Knowledge check](https://learn.microsoft.com/en-us/training/modules/use-sql-warehouses-azure-databricks/06-knowledge-check)3 min
* [Summary](https://learn.microsoft.com/en-us/training/modules/use-sql-warehouses-azure-databricks/07-summary)1 min

**Introduction**

* 1 minute

Data analysts often use SQL to query relational data and create reports and dashboards. Azure Databricks provides support for SQL-based data analytics through SQL Warehouses.

In this module, you'll learn how to:

* Create and configure SQL Warehouses in Azure Databricks.
* Create databases and tables.
* Create queries and dashboards.

**Note**

SQL Warehouses are available in *premium-tier* Azure Databricks workspaces.

**Get started with SQL Warehouses**

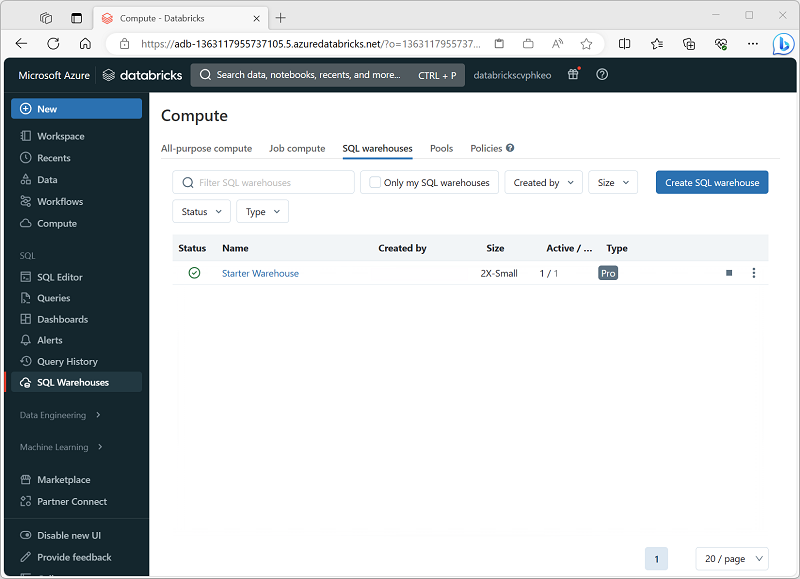
* 3 minutes

SQL Warehouses (formerly known as SQL Endpoints) provide a relational database interface for data in Azure Databricks. The data is stored in files that are abstracted by Delta tables, but from the perspective of the user or client application, the SQL Warehouse behaves like a relational database.

**Creating a SQL Warehouse**

When you create a premium-tier Azure Databricks workspace, it includes a default SQL Warehouse named **Starter Warehouse**, which you can use to explore sample data and get started with SQL-based data analytics in Azure Databricks. You can modify the configuration of the default SQL Warehouse to suit your needs, or you can create more SQL Warehouses in your workspace.

You can manage the SQL Warehouses in your Azure Databricks workspace by using the Azure Databricks portal.



**SQL Warehouse configuration settings**

When you create or configure a SQL Warehouse, you can specify the following settings:

* **Name**: A name used to identify the SQL Warehouse.
* **Cluster size**: Choose from a range of standard sizes to control the number and size of compute resources used to support the SQL Warehouse. Available sizes range from *2X-Small* (a single worker node) to *4X-Large* (256 worker nodes). For more information, see [Cluster size](https://learn.microsoft.com/en-us/azure/databricks/sql/admin/sql-endpoints#cluster-size) in the Azure Databricks documentation.
* **Auto Stop**: The amount of time the cluster will remain running when idle before being stopped. Idle clusters continue to incur charges when running.
* **Scaling**: The minimum and maximum number of clusters used to distribute query processing.
* **Type**: You can create a SQL Warehouse that uses *serverless* compute for fast, cost-effective on-demand provisioning. Alternatively, you can create a *Pro* or *Classic* SQL warehouse.

**Note**

You can create a SQL Warehouse with any available size, but if you have insufficient quota for the number of cores required to support your choice in the region where Azure Databricks is provisioned, the SQL Warehouse will fail to start.

**Create databases and tables**

* 3 minutes

After creating and starting a SQL Warehouse, you can start to work with data in tables.

**Database schema**

All SQL Warehouses contain a default database schema named **default**. You can use create tables in this schema in order to analyze data. However, if you need to work with multiple tables in a relational schema, or you have multiple analytical workloads where you want to manage the data (and access to it) separately, you can create custom database schema. To create a database, use the SQL editor to run a CREATE DATABASE or CREATE SCHEMA SQL statement. These statements are equivalent, but CREATE SCHEMA is preferred, as shown in this example:

SQLCopy

CREATE SCHEMA salesdata;

**Tip:**

For more information, see [**CREATE SCHEMA**](https://learn.microsoft.com/en-us/azure/databricks/sql/language-manual/sql-ref-syntax-ddl-create-schema) in the Azure Databricks documentation.

**Tables**

You can use the user interface in the Azure Databricks portal to upload delimited data, or import data from a wide range of common data sources. The imported data is stored in files in Databricks File System (DBFS) storage, and a Delta table is defined for it in the Hive metastore.

If the data files already exist in storage, or you need to define an explicit schema for the table, you can use a CREATE TABLE SQL statement. For example, the following code creates a table named **salesorders** in the **salesdata** database, based on the */data/sales/* folder in DBFS storage.

SQLCopy

CREATE TABLE salesdata.salesorders

(

orderid INT,

orderdate DATE,

customerid INT,

ordertotal DECIMAL

)

USING DELTA

LOCATION '/data/sales/';

**Tip:**

For more information, see [**CREATE TABLE**](https://learn.microsoft.com/en-us/azure/databricks/sql/language-manual/sql-ref-syntax-ddl-create-table) in the Azure Databricks documentation.

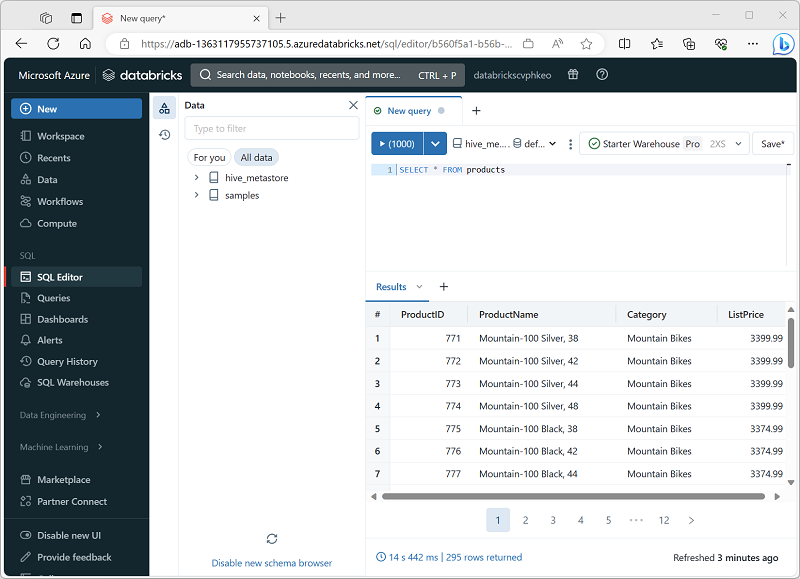
**Create queries and dashboards**

* 3 minutes

Azure Databricks SQL is primarily designed for data analytics and visualization workloads. To support these workloads, users can create *queries* to retrieve and summarize data from tables, and *dashboards* to share visualizations of the data.

**Queries**

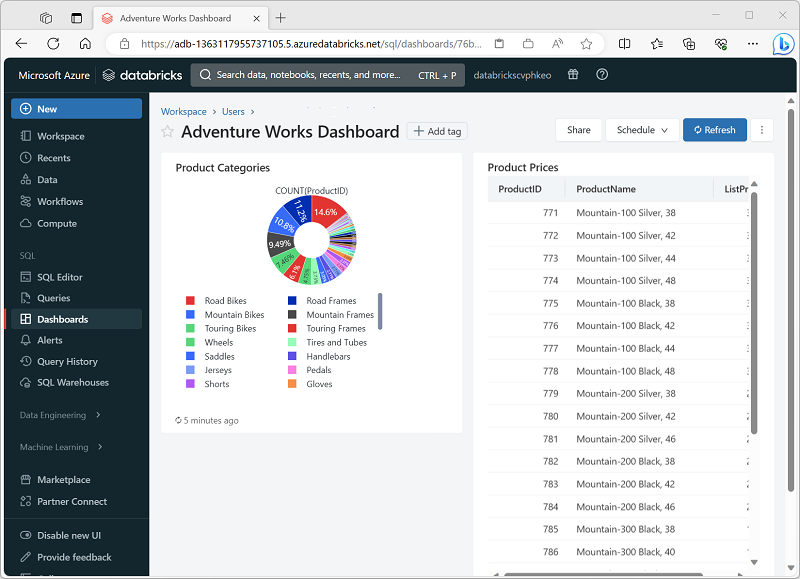
You can use the SQL Editor in the Azure Databricks portal to create a query based on any valid SQL SELECT statement, and then save the query with a meaningful name to be retrieved and run later.



After saving the query, you can schedule it to be run automatically at regular intervals to refresh the data, or you can open it and run it interactively.

**Dashboards**

Dashboards enable you to display the results of queries, either as tables of data or as graphical visualizations.



You can create multiple visualizations in a dashboard and share it with users in your organization. As with individual queries, you can schedule the dashboard to refresh is data periodically, and notify subscribers by email that new data is available.

**Exercise - Use a SQL Warehouse in Azure Databricks**

* 30 minutes

Now it's your chance to explore Azure Databricks SQL for yourself. In this exercise, you'll use a SQL Warehouse in Azure Databricks to query tables and create a dashboard.

**Note**

To complete this lab, you will need an [**Azure subscription**](https://azure.microsoft.com/free) in which you have administrative access.

Launch the exercise and follow the instructions.

[Button to launch exercise.](https://aka.ms/mslearn-databricks-sql)

Use a SQL Warehouse in Azure Databricks

SQL is an industry-standard language for querying and manipulating data. Many data analysts perform data analytics by using SQL to query tables in a relational database. Azure Databricks includes SQL functionality that builds on Spark and Delta Lake technologies to provide a relational database layer over files in a data lake.

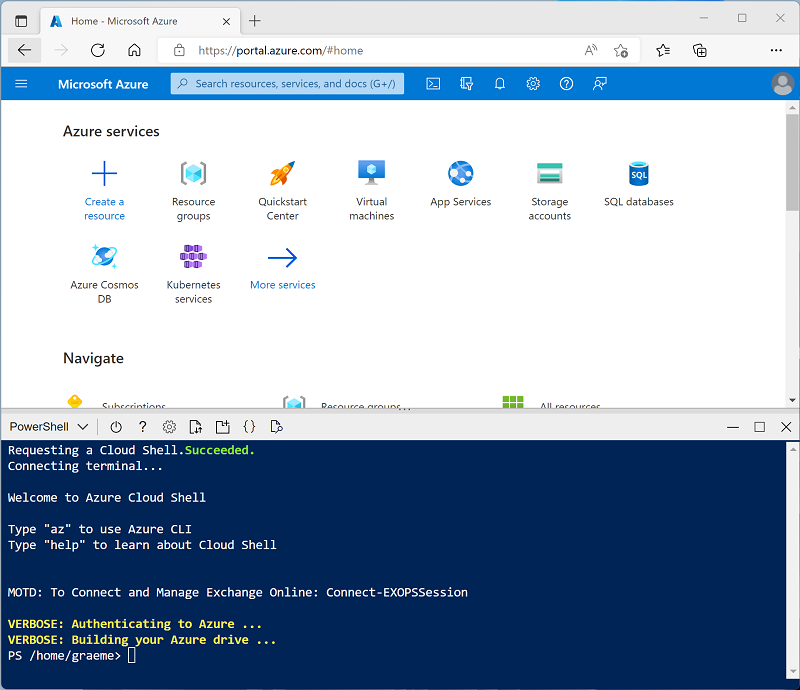
This exercise should take approximately **30** minutes to complete.

Provision an Azure Databricks workspace

**Tip**: If you already have have a *Premium* or *Trial* Azure Databricks workspace, you can skip this procedure and use your existing workspace.

This exercise includes a script to provision a new Azure Databricks workspace. The script attempts to create a *Premium* tier Azure Databricks workspace resource in a region in which your Azure subscription has sufficient quota for the compute cores required in this exercise; and assumes your user account has sufficient permissions in the subscription to create an Azure Databricks workspace resource. If the script fails due to insufficient quota or permissions, you can try to [create an Azure Databricks workspace interactively in the Azure portal](https://learn.microsoft.com/azure/databricks/getting-started/#--create-an-azure-databricks-workspace).

1. In a web browser, sign into the [Azure portal](https://portal.azure.com) at https://portal.azure.com.
2. Use the **[>\_]** button to the right of the search bar at the top of the page to create a new Cloud Shell in the Azure portal, selecting a ***PowerShell*** environment and creating storage if prompted. The cloud shell provides a command line interface in a pane at the bottom of the Azure portal, as shown here:

[](https://microsoftlearning.github.io/mslearn-databricks/Instructions/Exercises/images/cloud-shell.png)

**Note**: If you have previously created a cloud shell that uses a *Bash* environment, use the the drop-down menu at the top left of the cloud shell pane to change it to ***PowerShell***.

1. Note that you can resize the cloud shell by dragging the separator bar at the top of the pane, or by using the **—**, **◻**, and **X**icons at the top right of the pane to minimize, maximize, and close the pane. For more information about using the Azure Cloud Shell, see the [Azure Cloud Shell documentation](https://docs.microsoft.com/azure/cloud-shell/overview).
2. In the PowerShell pane, enter the following commands to clone this repo:

Code

rm -r mslearn-databricks -f

git clone https://github.com/MicrosoftLearning/mslearn-databricks

1. After the repo has been cloned, enter the following command to run the **setup.ps1** script, which provisions an Azure Databricks workspace in an available region:

Code

./mslearn-databricks/setup.ps1

1. If prompted, choose which subscription you want to use (this will only happen if you have access to multiple Azure subscriptions).
2. Wait for the script to complete - this typically takes around 5 minutes, but in some cases may take longer. While you are waiting, review the [What is data warehousing on Azure Databricks?](https://learn.microsoft.com/azure/databricks/sql/) article in the Azure Databricks documentation.

View and start a SQL Warehouse

1. When the Azure Databricks workspace resource has been deployed, go to it in the Azure portal.
2. In the **Overview** page for your Azure Databricks workspace, use the **Launch Workspace** button to open your Azure Databricks workspace in a new browser tab; signing in if prompted.

**Tip**: As you use the Databricks Workspace portal, various tips and notifications may be displayed. Dismiss these and follow the instructions provided to complete the tasks in this exercise.

1. View the Azure Databricks workspace portal and note that the sidebar on the left side contains the names of the task categories.
2. In the sidebar, under **SQL**, select **SQL Warehouses**.
3. Observe that the workspace already includes a SQL Warehouse named **Starter Warehouse**.
4. In the **Actions** (**⁝**) menu for the SQL Warehouse, select **Edit**. Then set the **Cluster size** property to **2X-Small** and save your changes.
5. Use the **Start** button to start the SQL Warehouse (which may take a minute or two).

**Note**: If your SQL Warehouse fails to start, your subscription may have insufficient quota in the region where your Azure Databricks workspace is provisioned. See [Required Azure vCPU quota](https://docs.microsoft.com/azure/databricks/sql/admin/sql-endpoints#required-azure-vcpu-quota) for details. If this happens, you can try requesting for a quota increase as detailed in the error message when the warehouse fails to start. Alternatively, you can try deleting your workspace and creating a new one in a different region. You can specify a region as a parameter for the setup script like this: ./setup.ps1 eastus

Create a database schema

1. When your SQL Warehouse is *running*, select **SQL Editor** in the sidebar.
2. In the **Schema browser** pane, observe that the *hive\_metastore* catalogue contains a database named **default**.
3. In the **New query** pane, enter the following SQL code:

Sql

CREATE DATABASE retail\_db;

1. Use the **►Run (1000)** button to run the SQL code.
2. When the code has been successfully executed, in the **Schema browser** pane, use the refresh button at the bottom of the pane to refresh the list. Then expand **hive\_metastore** and **retail\_db**, and observe that the database has been created, but contains no tables.

You can use the **default** database for your tables, but when building an analytical data store its best to create custom databases for specific data.

Create a table

1. Download the [**products.csv**](https://raw.githubusercontent.com/MicrosoftLearning/mslearn-databricks/main/data/products.csv) file to your local computer, saving it as **products.csv**.
2. In the Azure Databricks workspace portal, in the sidebar, select **(+) New** and then select **File Upload** and upload the **products.csv** file you downloaded to your computer.
3. In the **Upload data** page, select the **retail\_db** schema and set the table name to **products**. Then select **Create table** on the bottom left corner of the page.
4. When the table has been created, review its details.

The ability to create a table by importing data from a file makes it easy to populate a database. You can also use Spark SQL to create tables using code. The tables themselves are metadata definitions in the hive metastore, and the data they contain is stored in Delta format in Databricks File System (DBFS) storage.

Create a query

1. In the sidebar, select **(+) New** and then select **Query**.
2. In the **Schema browser** pane, expand **hive\_metastore** and **retail\_db**, and verify that the **products** table is listed.
3. In the **New query** pane, enter the following SQL code:

Sql

SELECT ProductID, ProductName, Category

FROM retail\_db.products;

1. Use the **►Run (1000)** button to run the SQL code.
2. When the query has completed, review the table of results.
3. Use the **Save** button at the top right of the query editor to save the query as **Products and Categories**.

Saving a query makes it easy to retrieve the same data again at a later time.

Create a dashboard

1. In the sidebar, select **(+) New** and then select **Dashboard**.
2. In the **New dashboard** dialog box, enter the name **Retail Dashboard** and select **Save**.
3. In the **Retail Dashboard** dashboard, in the **Add** drop-down list, select **Visualization**.
4. In the **Add visualization widget** dialog box, select the **Products and Categories** query. Then select **Create new visualization**, set the title to **Products Per Category**, and select **Create visualization**.
5. In the visualization editor, set the following properties:
   * **Visualization type**: bar
   * **Horizontal chart**: selected
   * **Y column**: Category
   * **X columns**: Product ID : Count
   * **Group by**: *Leave blank*
   * **Stacking**: Disabled
   * **Normalize values to percentage**: Unselected
   * **Missing and NULL values**: Do not display in chart
6. Save the visualization and view it in the dashboard.
7. Select **Done editing** to view the dashboard as users will see it.

Dashboards are a great way to share data tables and visualizations with business users. You can schedule the dashboards to be refreshed periodically, and emailed to subscribers.

Clean up

In Azure Databricks portal, on the **SQL Warehouses** page, select your SQL Warehouse and select **■ Stop** to shut it down.

If you’ve finished exploring Azure Databricks, you can delete the resources you’ve created to avoid unnecessary Azure costs and free up capacity in your subscription.

**Knowledge check**

* 3 minutes

Top of Form

**1. Which of the following workloads is best suited for Azure Databricks SQL?**

Running Scala code in notebooks to transform data.

Querying and visualizing data in relational tables.

**Correct. Azure Databricks SQL is optimized for SQL-based querying and data**

**visualization.**

Training and deploying machine learning models.

**2. Which statement should you use to create a database in a SQL warehouse?**

CREATE VIEW

CREATE SCHEMA

**Correct. The CREATE SCHEMA statement is used to create a database.**

CREATE GROUP

**3. You need to share data visualizations, including charts and tables of data, with users in your organization. What should you create?**

A table

A query

A dashboard

**Correct. A dashboard can be used to share data visualizations with other users.**

**Lab 5: Build an end-to-end data pipeline in Databricks –**

<https://learn.microsoft.com/en-us/azure/databricks/getting-started/data-pipeline-get-started>

**Build an end-to-end data pipeline in Databricks**

* Article
* 03/06/2024
* 5 contributors

**In this article**

1. [What is a data pipeline?](https://learn.microsoft.com/en-us/azure/databricks/getting-started/data-pipeline-get-started#what-is-a-data-pipeline)
2. [Data pipeline steps](https://learn.microsoft.com/en-us/azure/databricks/getting-started/data-pipeline-get-started#data-pipeline-steps)
3. [Requirements](https://learn.microsoft.com/en-us/azure/databricks/getting-started/data-pipeline-get-started#requirements)
4. [Example: Million Song dataset](https://learn.microsoft.com/en-us/azure/databricks/getting-started/data-pipeline-get-started#example-million-song-dataset)

[Step 1: Create a cluster](https://learn.microsoft.com/en-us/azure/databricks/getting-started/data-pipeline-get-started#--step-1-create-a-cluster)

[Step 2: Explore the source data](https://learn.microsoft.com/en-us/azure/databricks/getting-started/data-pipeline-get-started#step-2-explore-the-source-data)

[Step 3: Ingest the raw data](https://learn.microsoft.com/en-us/azure/databricks/getting-started/data-pipeline-get-started#--step-3-ingest-the-raw-data)

[Step 4: Prepare the raw data](https://learn.microsoft.com/en-us/azure/databricks/getting-started/data-pipeline-get-started#step-4-prepare-the-raw-data)

[Step 5: Query the transformed data](https://learn.microsoft.com/en-us/azure/databricks/getting-started/data-pipeline-get-started#step-5-query-the-transformed-data)

[Step 6: Create an Azure Databricks job to run the pipeline](https://learn.microsoft.com/en-us/azure/databricks/getting-started/data-pipeline-get-started#step-6-create-an-azure-databricks-job-to-run-the-pipeline)

[Step 7: Schedule the data pipeline job](https://learn.microsoft.com/en-us/azure/databricks/getting-started/data-pipeline-get-started#step-7-schedule-the-data-pipeline-job)

[Learn more](https://learn.microsoft.com/en-us/azure/databricks/getting-started/data-pipeline-get-started#learn-more)

This article shows you how to create and deploy an end-to-end data processing pipeline, including how to ingest raw data, transform the data, and run analyses on the processed data.

**Note**

Although this article demonstrates how to create a complete data pipeline using Databricks [**notebooks**](https://learn.microsoft.com/en-us/azure/databricks/notebooks/)and an Azure Databricks [**job**](https://learn.microsoft.com/en-us/azure/databricks/workflows/#what-is-jobs) to orchestrate a workflow, Databricks recommends using [**Delta Live Tables**](https://learn.microsoft.com/en-us/azure/databricks/delta-live-tables/), a declarative interface for building reliable, maintainable, and testable data processing pipelines.

**What is a data pipeline?**

A data pipeline implements the steps required to move data from source systems, transform that data based on requirements, and store the data in a target system. A data pipeline includes all the processes necessary to turn raw data into prepared data that users can consume. For example, a data pipeline might prepare data so data analysts and data scientists can extract value from the data through analysis and reporting.

An extract, transform, and load (ETL) workflow is a common example of a data pipeline. In ETL processing, data is ingested from source systems and written to a staging area, transformed based on requirements (ensuring data quality, deduplicating records, and so forth), and then written to a target system such as a data warehouse or data lake.

**Data pipeline steps**

To help you get started building data pipelines on Azure Databricks, the example included in this article walks through creating a data processing workflow:

* Use Azure Databricks features to explore a raw dataset.
* Create a Databricks notebook to ingest raw source data and write the raw data to a target table.
* Create a Databricks notebook to transform the raw source data and write the transformed data to a target table.
* Create a Databricks notebook to query the transformed data.
* Automate the data pipeline with an Azure Databricks job.

**Requirements**

* You’re logged into Azure Databricks and in the Data Science & Engineering workspace.
* You have [permission to create a cluster](https://learn.microsoft.com/en-us/azure/databricks/compute/use-compute) or [access to a cluster](https://learn.microsoft.com/en-us/azure/databricks/compute/use-compute#permissions).
* (Optional) To publish tables to Unity Catalog, you must create a [catalog](https://learn.microsoft.com/en-us/azure/databricks/data-governance/unity-catalog/create-catalogs) and [schema](https://learn.microsoft.com/en-us/azure/databricks/data-governance/unity-catalog/create-schemas) in Unity Catalog.

**Example: Million Song dataset**

The dataset used in this example is a subset of the [Million Song Dataset](http://labrosa.ee.columbia.edu/millionsong/), a collection of features and metadata for contemporary music tracks. This dataset is available in the [sample datasets](https://learn.microsoft.com/en-us/azure/databricks/discover/databricks-datasets#databricks-datasets-databricks-datasets) included in your Azure Databricks workspace.

**Step 1: Create a cluster**

To perform the data processing and analysis in this example, create a cluster to provide the compute resources needed to run commands.

**Note**

Because this example uses a sample dataset stored in DBFS and recommends persisting tables to [**Unity Catalog**](https://learn.microsoft.com/en-us/azure/databricks/data-governance/unity-catalog/), you create a cluster configured with *single user access* mode. Single user access mode provides full access to DBFS while also enabling access to Unity Catalog. See [**Best practices for DBFS and Unity Catalog**](https://learn.microsoft.com/en-us/azure/databricks/dbfs/unity-catalog).

1. Click **Compute** in the sidebar.
2. On the Compute page, click **Create Cluster**.
3. On the New Cluster page, enter a unique name for the cluster.
4. In **Access mode**, select **Single User**.
5. In **Single user or service principal access**, select your user name.
6. Leave the remaining values in their default state, and click **Create Cluster**.

To learn more about Databricks clusters, see [Compute](https://learn.microsoft.com/en-us/azure/databricks/compute/).

**Step 2: Explore the source data**

To learn how to use the Azure Databricks interface to explore the raw source data, see [Explore the source data for a data pipeline](https://learn.microsoft.com/en-us/azure/databricks/getting-started/data-pipeline-explore-data). If you want to go directly to ingesting and preparing the data, continue to [Step 3: Ingest the raw data](https://learn.microsoft.com/en-us/azure/databricks/getting-started/data-pipeline-get-started#ingest-prepare-data).

**Step 3: Ingest the raw data**

In this step, you load the raw data into a table to make it available for further processing. To manage data assets on the Databricks platform such as tables, Databricks recommends [Unity Catalog](https://learn.microsoft.com/en-us/azure/databricks/data-governance/unity-catalog/). However, if you don’t have permissions to create the required catalog and schema to publish tables to Unity Catalog, you can still complete the following steps by publishing tables to the Hive metastore.

To ingest data, Databricks recommends using [Auto Loader](https://learn.microsoft.com/en-us/azure/databricks/ingestion/auto-loader/). Auto Loader automatically detects and processes new files as they arrive in cloud object storage.

You can configure Auto Loader to automatically detect the schema of loaded data, allowing you to initialize tables without explicitly declaring the data schema and evolve the table schema as new columns are introduced. This eliminates the need to manually track and apply schema changes over time. Databricks recommends schema inference when using Auto Loader. However, as seen in the data exploration step, the songs data does not contain header information. Because the header is not stored with the data, you’ll need to explicitly define the schema, as shown in the next example.

1. In the sidebar, click New Icon **New** and select **Notebook** from the menu. The **Create Notebook** dialog appears.
2. Enter a name for the notebook, for example, Ingest songs data. By default:
   * **Python** is the selected language.
   * The notebook is attached to the last cluster you used. In this case, the cluster you created in [Step 1: Create a cluster](https://learn.microsoft.com/en-us/azure/databricks/getting-started/data-pipeline-get-started#create-a-cluster).
3. Enter the following into the first cell of the notebook:

PythonCopy

from pyspark.sql.types import DoubleType, IntegerType, StringType, StructType, StructField

# Define variables used in the code below

file\_path = "/databricks-datasets/songs/data-001/"

table\_name = "<table-name>"

checkpoint\_path = "/tmp/pipeline\_get\_started/\_checkpoint/song\_data"

schema = StructType(

[

StructField("artist\_id", StringType(), True),

StructField("artist\_lat", DoubleType(), True),

StructField("artist\_long", DoubleType(), True),

StructField("artist\_location", StringType(), True),

StructField("artist\_name", StringType(), True),

StructField("duration", DoubleType(), True),

StructField("end\_of\_fade\_in", DoubleType(), True),

StructField("key", IntegerType(), True),

StructField("key\_confidence", DoubleType(), True),

StructField("loudness", DoubleType(), True),

StructField("release", StringType(), True),

StructField("song\_hotnes", DoubleType(), True),

StructField("song\_id", StringType(), True),

StructField("start\_of\_fade\_out", DoubleType(), True),

StructField("tempo", DoubleType(), True),

StructField("time\_signature", DoubleType(), True),

StructField("time\_signature\_confidence", DoubleType(), True),

StructField("title", StringType(), True),

StructField("year", IntegerType(), True),

StructField("partial\_sequence", IntegerType(), True)

]

)

(spark.readStream

.format("cloudFiles")

.schema(schema)

.option("cloudFiles.format", "csv")

.option("sep","\t")

.load(file\_path)

.writeStream

.option("checkpointLocation", checkpoint\_path)

.trigger(availableNow=True)

.toTable(table\_name)

)

If you are using Unity Catalog, replace <table-name> with a catalog, schema, and table name to contain the ingested records (for example, data\_pipelines.songs\_data.raw\_song\_data). Otherwise, replace <table-name> with the name of a table to contain the ingested records, for example, raw\_song\_data.

Replace <checkpoint-path> with a path to a directory in DBFS to maintain checkpoint files, for example, /tmp/pipeline\_get\_started/\_checkpoint/song\_data.

1. Click Run Menu, and select **Run Cell**. This example defines the data schema using the information from the README, ingests the songs data from all of the files contained in file\_path, and writes the data to the table specified by table\_name.

**Step 4: Prepare the raw data**

To prepare the raw data for analysis, the following steps transform the raw songs data by filtering out unneeded columns and adding a new field containing a timestamp for the creation of the new record.

1. In the sidebar, click New Icon **New** and select **Notebook** from the menu. The **Create Notebook** dialog appears.
2. Enter a name for the notebook. For example, Prepare songs data. Change the default language to **SQL**.
3. Enter the following in the first cell of the notebook:

SQLCopy

CREATE OR REPLACE TABLE

<table-name> (

artist\_id STRING,

artist\_name STRING,

duration DOUBLE,

release STRING,

tempo DOUBLE,

time\_signature DOUBLE,

title STRING,

year DOUBLE,

processed\_time TIMESTAMP

);

INSERT INTO

<table-name>

SELECT

artist\_id,

artist\_name,

duration,

release,

tempo,

time\_signature,

title,

year,

current\_timestamp()

FROM

<raw-songs-table-name>

If you are using Unity Catalog, replace <table-name> with a catalog, schema, and table name to contain the filtered and transformed records (for example, data\_pipelines.songs\_data.prepared\_song\_data). Otherwise, replace <table-name> with the name of a table to contain the filtered and transformed records (for example, prepared\_song\_data).

Replace <raw-songs-table-name> with the name of the table containing the raw songs records ingested in the previous step.

1. Click Run Menu, and select **Run Cell**.

**Step 5: Query the transformed data**

In this step, you extend the processing pipeline by adding queries to analyze the songs data. These queries use the prepared records created in the previous step.

1. In the sidebar, click New Icon **New** and select **Notebook** from the menu. The **Create Notebook** dialog appears.
2. Enter a name for the notebook. For example, Analyze songs data. Change the default language to **SQL**.
3. Enter the following in the first cell of the notebook:

SQLCopy

-- Which artists released the most songs each year?

SELECT

artist\_name,

count(artist\_name)

AS

num\_songs,

year

FROM

<prepared-songs-table-name>

WHERE

year > 0

GROUP BY

artist\_name,

year

ORDER BY

num\_songs DESC,

year DESC

Replace <prepared-songs-table-name> with the name of the table containing prepared data. For example, data\_pipelines.songs\_data.prepared\_song\_data.

1. Click Down Caret in the cell actions menu, select **Add Cell Below** and enter the following in the new cell:

SQLCopy

-- Find songs for your DJ list

SELECT

artist\_name,

title,

tempo

FROM

<prepared-songs-table-name>

WHERE

time\_signature = 4

AND

tempo between 100 and 140;

Replace <prepared-songs-table-name> with the name of the prepared table created in the previous step. For example, data\_pipelines.songs\_data.prepared\_song\_data.

1. To run the queries and view the output, click **Run all**.

**Step 6: Create an Azure Databricks job to run the pipeline**

You can create a workflow to automate running the data ingestion, processing, and analysis steps using an Azure Databricks job.

1. In your Data Science & Engineering workspace, do one of the following:
   * Click Jobs Icon **Workflows** in the sidebar and click Create Job Button.
   * In the sidebar, click New Icon **New** and select **Job**.
2. In the task dialog box on the **Tasks** tab, replace **Add a name for your job…** with your job name. For example, “Songs workflow”.
3. In **Task name**, enter a name for the first task, for example, Ingest\_songs\_data.
4. In **Type**, select the **Notebook** task type.
5. In **Source**, select **Workspace**.
6. Use the file browser to find the data ingestion notebook, click the notebook name, and click **Confirm**.
7. In **Cluster**, select **Shared\_job\_cluster** or the cluster you created in the Create a cluster step.
8. Click **Create**.
9. Click Add Task Button below the task you just created and select **Notebook**.
10. In **Task name**, enter a name for the task, for example, Prepare\_songs\_data.
11. In **Type**, select the **Notebook** task type.
12. In **Source**, select **Workspace**.
13. Use the file browser to find the data preparation notebook, click the notebook name, and click **Confirm**.
14. In **Cluster**, select **Shared\_job\_cluster** or the cluster you created in the Create a cluster step.
15. Click **Create**.
16. Click Add Task Button below the task you just created and select **Notebook**.
17. In **Task name**, enter a name for the task, for example, Analyze\_songs\_data.
18. In **Type**, select the **Notebook** task type.
19. In **Source**, select **Workspace**.
20. Use the file browser to find the data analysis notebook, click the notebook name, and click **Confirm**.
21. In **Cluster**, select **Shared\_job\_cluster** or the cluster you created in the Create a cluster step.
22. Click **Create**.
23. To run the workflow, Click Run Now Button. To view [details for the run](https://learn.microsoft.com/en-us/azure/databricks/workflows/jobs/monitor-job-runs#job-run-details), click the link in the **Start time**column for the run in the [job runs](https://learn.microsoft.com/en-us/azure/databricks/workflows/jobs/monitor-job-runs#view-job-run-list) view. Click each task to view details for the task run.
24. To view the results when the workflow completes, click the final data analysis task. The **Output** page appears and displays the query results.

**Step 7: Schedule the data pipeline job**

**Note**

To demonstrate using an Azure Databricks job to orchestrate a scheduled workflow, this getting started example separates the ingestion, preparation, and analysis steps into separate notebooks, and each notebook is then used to create a task in the job. If all of the processing is contained in a single notebook, you can easily schedule the notebook directly from the Azure Databricks notebook UI. See [**Create and manage scheduled notebook jobs**](https://learn.microsoft.com/en-us/azure/databricks/notebooks/schedule-notebook-jobs).

A common requirement is to run a data pipeline on a scheduled basis. To define a schedule for the job that runs the pipeline:

1. Click Jobs Icon **Workflows** in the sidebar.
2. In the **Name** column, click the job name. The side panel displays the **Job details**.
3. Click **Add trigger** in the **Job details** panel and select **Scheduled** in **Trigger type**.
4. Specify the period, starting time, and time zone. Optionally select the **Show Cron Syntax** checkbox to display and edit the schedule in [Quartz Cron Syntax](http://www.quartz-scheduler.org/documentation/quartz-2.3.0/tutorials/crontrigger.html).
5. Click **Save**.

**Learn more**

* To learn more about Databricks notebooks, see [Introduction to Databricks notebooks](https://learn.microsoft.com/en-us/azure/databricks/notebooks/).
* To learn more about Azure Databricks Jobs, see [What is Azure Databricks Jobs?](https://learn.microsoft.com/en-us/azure/databricks/workflows/#what-is-jobs).
* To learn more about Delta Lake, see [What is Delta Lake?](https://learn.microsoft.com/en-us/azure/databricks/delta/).
* To learn more about data processing pipelines with Delta Live Tables, see [What is Delta Live Tables?](https://learn.microsoft.com/en-us/azure/databricks/delta-live-tables/).

Bottom of Form

Lab 6: Sample Dashboards in Databricks + Spark SQL –

<https://learn.microsoft.com/en-us/azure/databricks/sql/get-started/sample-dashboards>

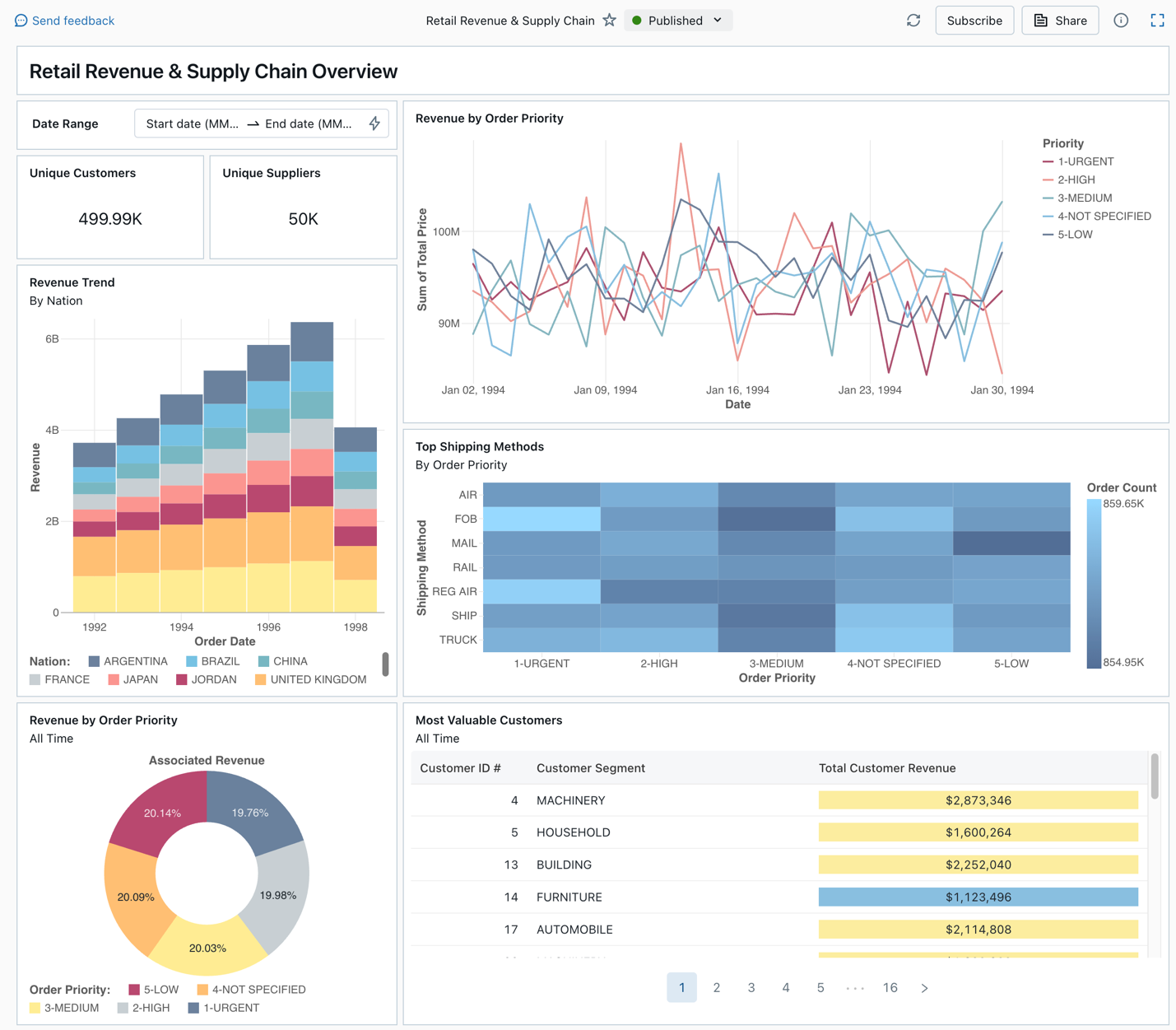
**Tutorial: Use sample dashboards**

* Article
* 04/04/2024
* 8 contributors

**In this article**

1. [Import a dashboard](https://learn.microsoft.com/en-us/azure/databricks/sql/get-started/sample-dashboards#--import-a-dashboard)
2. [Explore a visualization’s query](https://learn.microsoft.com/en-us/azure/databricks/sql/get-started/sample-dashboards#explore-a-visualizations-query)
3. [Interact with a visualization](https://learn.microsoft.com/en-us/azure/databricks/sql/get-started/sample-dashboards#interact-with-a-visualization)
4. [Publish the dashboard](https://learn.microsoft.com/en-us/azure/databricks/sql/get-started/sample-dashboards#publish-the-dashboard)
5. [Share the dashboard](https://learn.microsoft.com/en-us/azure/databricks/sql/get-started/sample-dashboards#share-the-dashboard)
6. [Schedule automatic dashboard refreshes](https://learn.microsoft.com/en-us/azure/databricks/sql/get-started/sample-dashboards#schedule-automatic-dashboard-refreshes)

This tutorial shows you how to import and use sample dashboards from the samples gallery. These dashboards illustrate some of the rich visualizations you can use to gain insights from your data. No setup is required. These dashboards use data already available in your workspace and rely on a compute resource (called a SQL warehouse) already configured. You don’t need to be an administrator to get started.



See [What are Lakeview dashboards?](https://learn.microsoft.com/en-us/azure/databricks/dashboards/lakeview) to learn about all of the visualization types and features available for Lakeview dashboards.

**Import a dashboard**

1. In the sidebar, click Dashboards Icon **Dashboards**

If your workspace has any saved dashboards, they are listed.

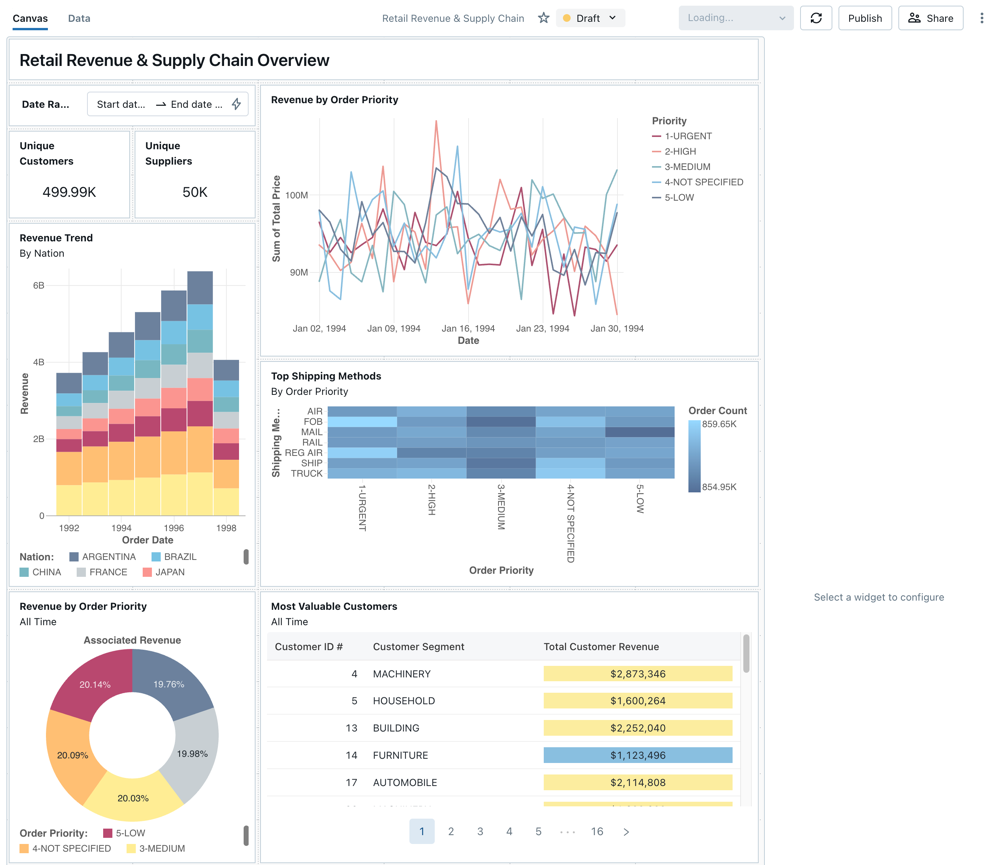
1. Click **View samples gallery**.

**Note**

You can also navigate to the Dashboard Samples Gallery by appending /sql/dashboards/samples/to your workspace URL.

1. In the **Retail Revenue & Supply Chain** tile, click **Import**. The dashboard is imported into your workspace, and you are the owner.

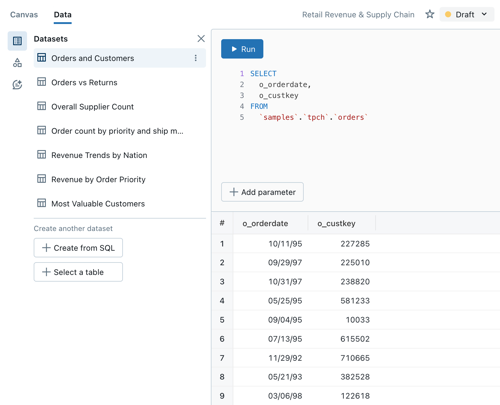
The imported draft dashboard appears, and its visualizations are refreshed.



You can import a sample dashboard multiple times, and multiple users can each import it. You can also import the **NYC Taxi Trip Analysis** dashboard.

**Explore a visualization’s query**

1. Each visualization in a dashboard is the result of a query. You can access all queries in the **Data** tab on the draft dashboard. Click **Data** in the upper-left corner of the screen. Then, click the dataset you want to view to see the associated query.



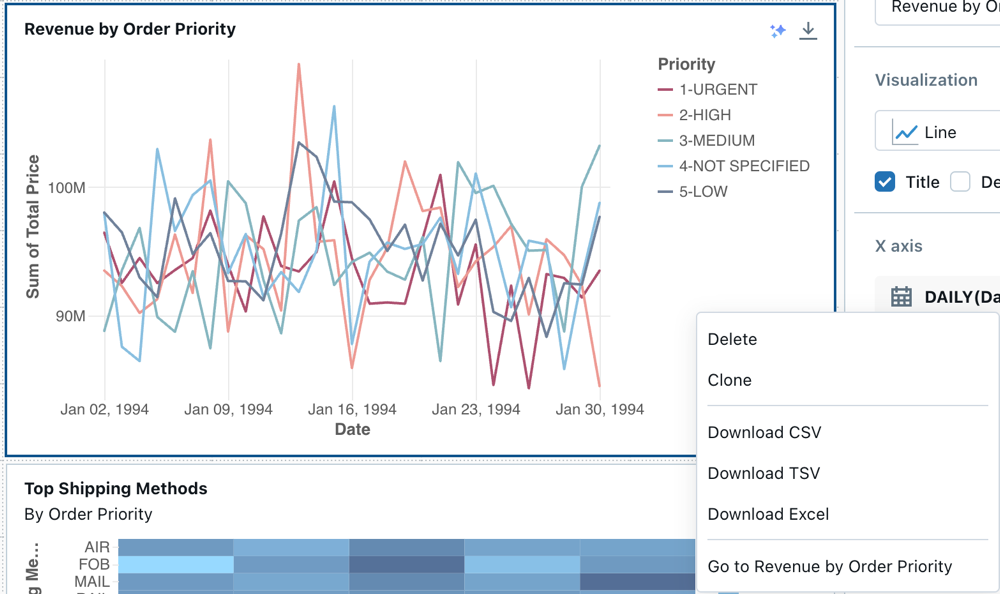
The SQL editor includes the query and results, which are shown in a table below the query.

The sample dashboards use data in the samples catalog, separate from data in your workspace. The samples catalog is available to every workspace but is read-only.

1. Click the **Canvas** tab to go back to the canvas that shows the dashboard’s visualization widgets.

**Interact with a visualization**

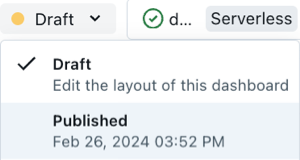
1. Hover over the **Revenue by Order Priority** visualization.
2. Click each **Priority** in the legend to focus on that group of data and hide the other lines.
3. Right-click on the visualization to see its context menu. You can delete or clone a visualization. You can also download the associated dataset as a CSV, TSV, or Excel file. Click **Go to Revenue by Order Priority** to view the associated query.



The query opens on the **Data** tab of your dashboard.

**Publish the dashboard**

* Click **Publish** at the top of the page. A **Publish** dialog appears.
* Click **Publish** in the dialog to create a sharable, non-editable version of your dashboard. This dashboard is published with your credentials embedded by default. This means that other viewers use your credentials to access the data and compute to generate visualizations on your dashboard. See [Publish a Lakeview dashboard](https://learn.microsoft.com/en-us/azure/databricks/dashboards/lakeview#publish-a-lakeview-dashboard).
* Use the switcher at the top of the page to view your published dashboard.



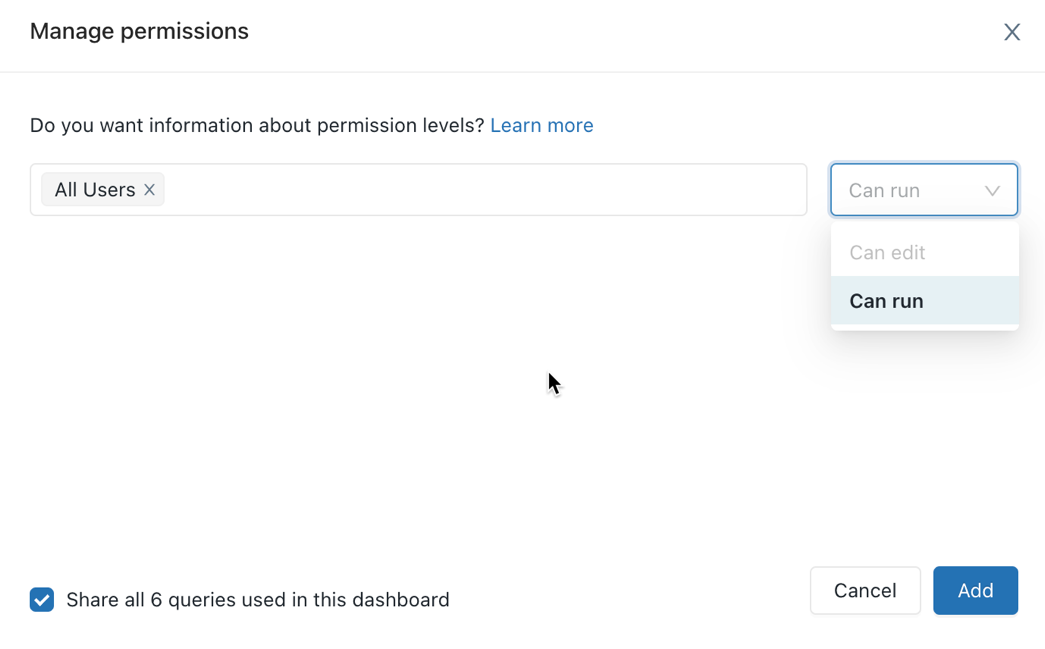
**Share the dashboard**

To share a dashboard with colleagues in your workspace:

1. Click **Share** at the top of the page.
2. Select a user or group in your workspace.

To share the dashboard with all users in the workspace, select **All users**.

1. Select the permission to grant.



To share a dashboard with account users:

1. Under **Sharing settings** at the bottom of the sharing dialog, click **Anyone in my organization can view.**

This means that anyone who is registered to your Azure Databricks account can use a link to access your dashboard. If you have embedded your credentials, account-level users don’t need workspace access to view your dashboard.

1. Close the form.

**Schedule automatic dashboard refreshes**

You can schedule the dashboard to refresh at an interval automatically.

1. At the top of the page, click **Schedule**.
2. Click **Add schedule**.
3. Select an interval, such as **Every 1 hour** at **5 minutes past the hour**. The SQL warehouse that you selected to run your queries is used to run the dashboard’s queries and generate visualizations when the dashboard is refreshed.

Workspace admin users can create, configure, and delete SQL warehouses.

1. Click **Create**.

The dialog shows all schedules associated with the dashboard.

1. Optionally, click **Subscribe** to add yourself as a subscriber and receive an email with a PDF snapshot of the dashboard after a scheduled run completes.

You can use the kebab menu Kebab menu to edit the schedule and add more subscribers. See [Schedule dashboards for periodic updates](https://learn.microsoft.com/en-us/azure/databricks/dashboards/lakeview#schedule-dashboards-for-periodic-updates).

1. To delete an existing schedule for a dashboard:
   1. Click **Subscribe**.
   2. Click the kebab menu Vertical Ellipsis on the right.
   3. Click **Delete**.

Lab 7: Visualize queries and create a dashboard in Datbricks SQL- https://learn.microsoft.com/en- us/azure/databricks/sql/get- started/visualize-data-tutorial

# Visualize queries and create a dashboard in Databricks SQL

* Article
* 03/07/2024
* 3 contributors

## In this article

1. [Connect to Databricks SQL with SQL editor](https://learn.microsoft.com/en-us/azure/databricks/sql/get-started/visualize-data-tutorial#--connect-to-databricks-sql-with-sql-editor)
2. [Query for pickup hour distribution](https://learn.microsoft.com/en-us/azure/databricks/sql/get-started/visualize-data-tutorial#query-for-pickup-hour-distribution)
3. [Create a visualization for the distribution of taxi pickups by hour.](https://learn.microsoft.com/en-us/azure/databricks/sql/get-started/visualize-data-tutorial#create-a-visualization-for-the-distribution-of-taxi-pickups-by-hour)
4. [Query for daily fare trends](https://learn.microsoft.com/en-us/azure/databricks/sql/get-started/visualize-data-tutorial#query-for-daily-fare-trends)
5. [Create a visualization for daily fare trends](https://learn.microsoft.com/en-us/azure/databricks/sql/get-started/visualize-data-tutorial#create-a-visualization-for-daily-fare-trends)
6. [Create a dashboard using these visualizations](https://learn.microsoft.com/en-us/azure/databricks/sql/get-started/visualize-data-tutorial#create-a-dashboard-using-these-visualizations)
7. [Add a pickup zip code parameter to each query](https://learn.microsoft.com/en-us/azure/databricks/sql/get-started/visualize-data-tutorial#add-a-pickup-zip-code-parameter-to-each-query)
8. [Update the dashboard to use a dashboard parameter](https://learn.microsoft.com/en-us/azure/databricks/sql/get-started/visualize-data-tutorial#update-the-dashboard-to-use-a-dashboard-parameter)

This tutorial uses the New York City taxi dataset in Samples. It shows you how to use SQL editor in Databricks SQL to create a visualization for each of several queries and then create a dashboard using these visualizations. It also shows you how to create a dashboard parameter for each of the visualizations in the dashboard.

## Connect to Databricks SQL with SQL editor

1. Click New Icon **New** in the sidebar and select **Query**.

The SQL editor opens. If you do not have access to Databricks SQL, request access to from an admin.

1. Select a warehouse.

The first time you create a query the list of available SQL warehouses displays in alphabetical order. The next time you create a query, the last used warehouse is selected.

1. Click **Serverless Starter Warehouse**. This warehouse is created for you automatically to help you get started quickly. If serverless is not enabled for your workspace, choose **Starter Warehouse**. For information on creating SQL warehouses, see [Create a SQL warehouse](https://learn.microsoft.com/en-us/azure/databricks/compute/sql-warehouse/create).

## Query for pickup hour distribution

1. In SQL editor, paste the following query in the new query window to return the distribution of taxi pickups by hour.

SQLCopy

SELECT

date\_format(tpep\_pickup\_datetime, "HH") AS `Pickup Hour`,

count(\*) AS `Number of Rides`

FROM

samples.nyctaxi.trips

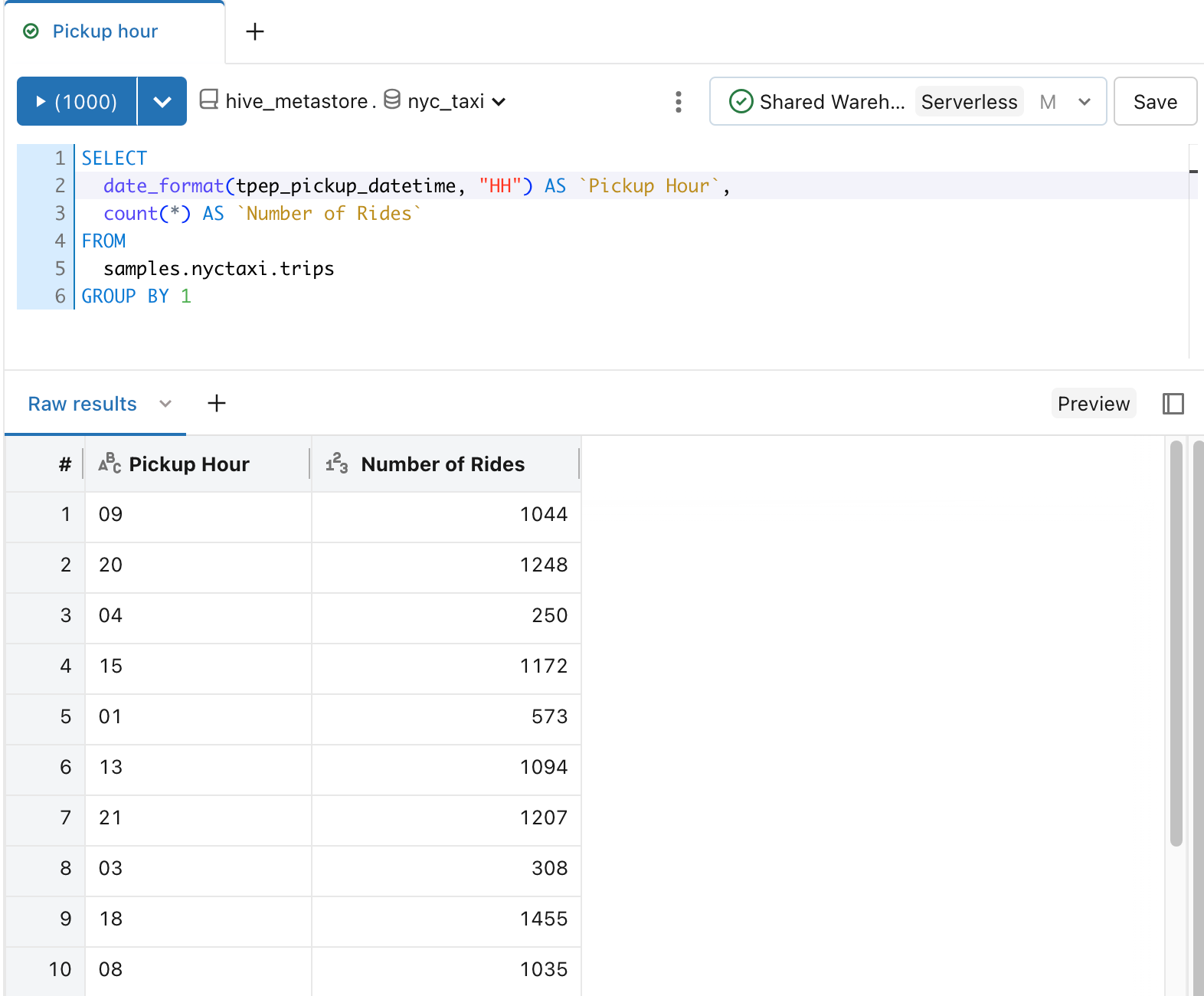
GROUP BY 1

1. Press **Ctrl/Cmd + Enter** or click **Run (1000)**. After a few seconds, the query results are shown below the query in the results pane.

**Limit 1000** is selected by default for all queries to ensure that the query returns at most 1000 rows. If a query is saved with the **Limit 1000** setting, this setting applies to all executions of the query (including within dashboards). If you want to return all rows for this query, you can unselect **LIMIT 1000** by clicking the **Run (1000)** drop-down. If you want to specify a different limit on the number of rows, you can add a LIMIT clause in your query with a value of your choice.

The query result displays in the Results tab.

1. Click **Save** and save the query as Pickup hour.

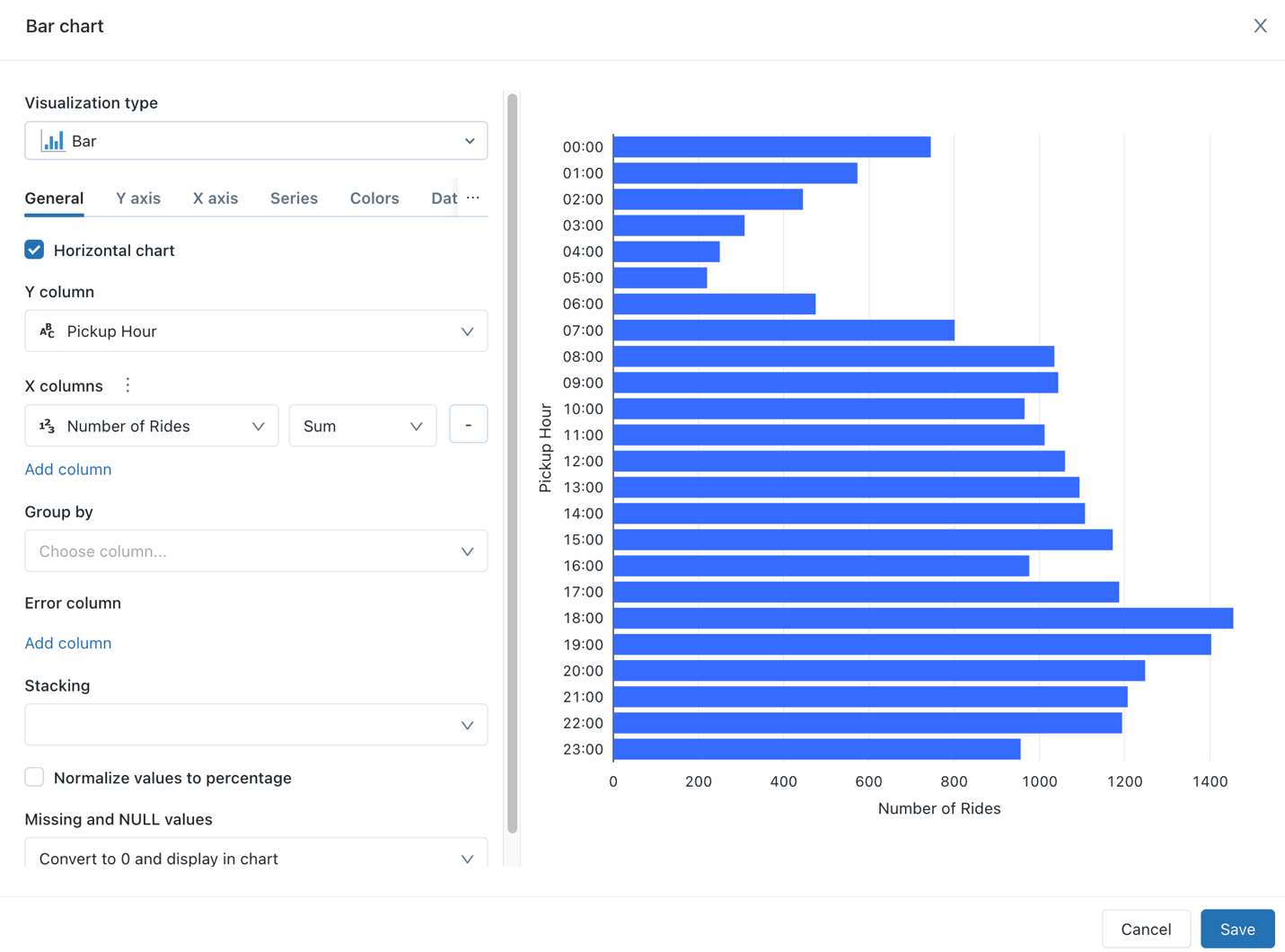


## Create a visualization for the distribution of taxi pickups by hour.

1. Next to the Results tab, click **+** and then click **Visualization**.

The visualization editor displays.

1. In the **Visualization Type** drop-down, verify that **Bar** is selected.
2. Change the visualization name to Bar chart.
3. Verify that Pickup Hour is specified for the **Y column** drop down.
4. Verify that Number of Rides and Sum are specified for the **X column** drop down.



1. Click **Save**.

The saved chart displays in the SQL editor.

## Query for daily fare trends

1. In SQL editor, click **+** and then click **Create new query**.
2. In the new query window, paste the following query to return the daily fare trends.

SQLCopy

SELECT

T.weekday,

CASE

WHEN T.weekday = 1 THEN 'Sunday'

WHEN T.weekday = 2 THEN 'Monday'

WHEN T.weekday = 3 THEN 'Tuesday'

WHEN T.weekday = 4 THEN 'Wednesday'

WHEN T.weekday = 5 THEN 'Thursday'

WHEN T.weekday = 6 THEN 'Friday'

WHEN T.weekday = 7 THEN 'Saturday'

ELSE 'N/A'

END AS day\_of\_week,

T.fare\_amount,

T.trip\_distance

FROM

(

SELECT

dayofweek(tpep\_pickup\_datetime) as weekday,

\*

FROM

`samples`.`nyctaxi`.`trips`

) T

1. Click **Save** and save the query as Daily fare to distance analysis.

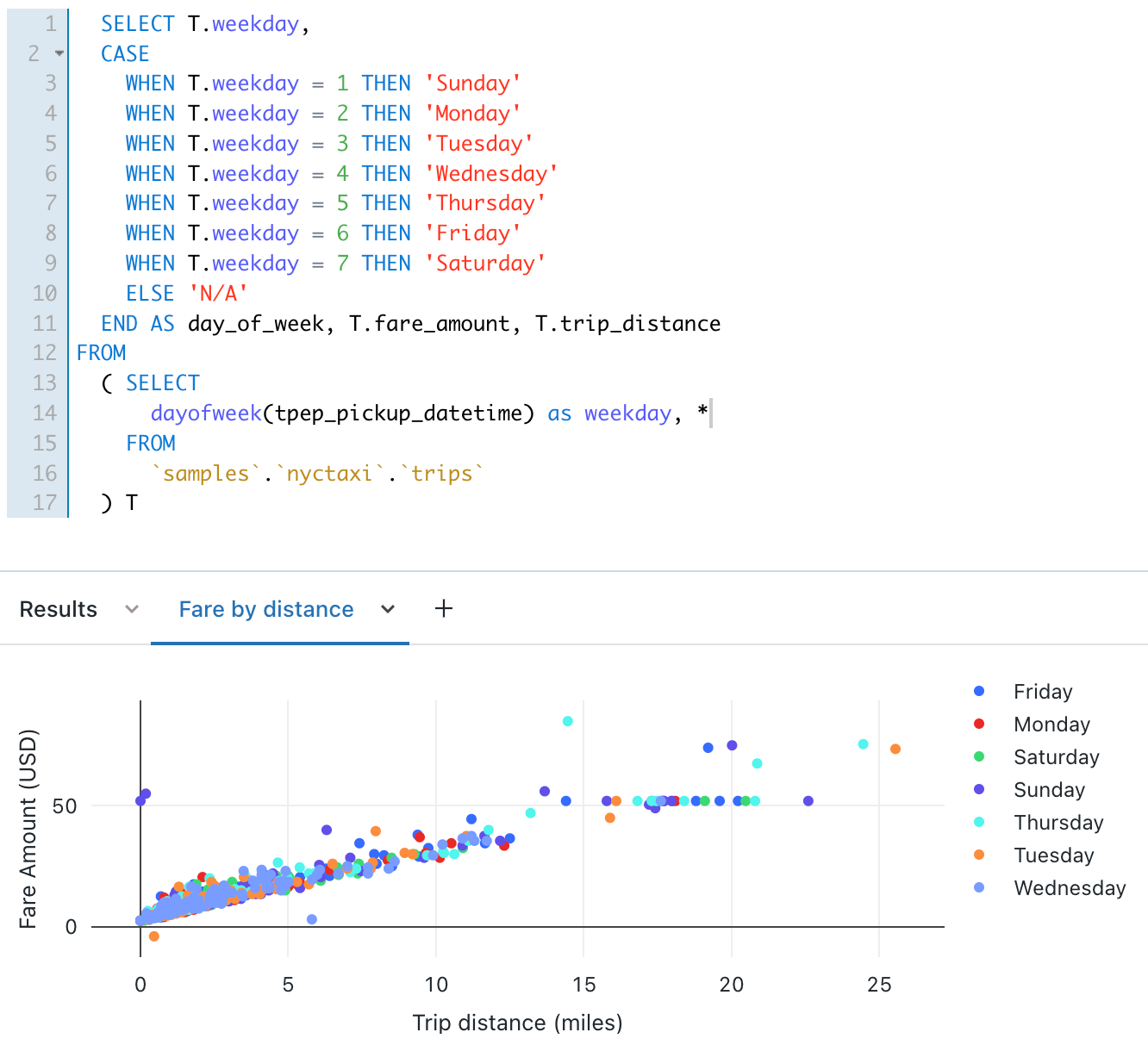
## Create a visualization for daily fare trends

1. Next to the **Results** tab, click **+** and then click **Visualization**.

The visualization editor displays.

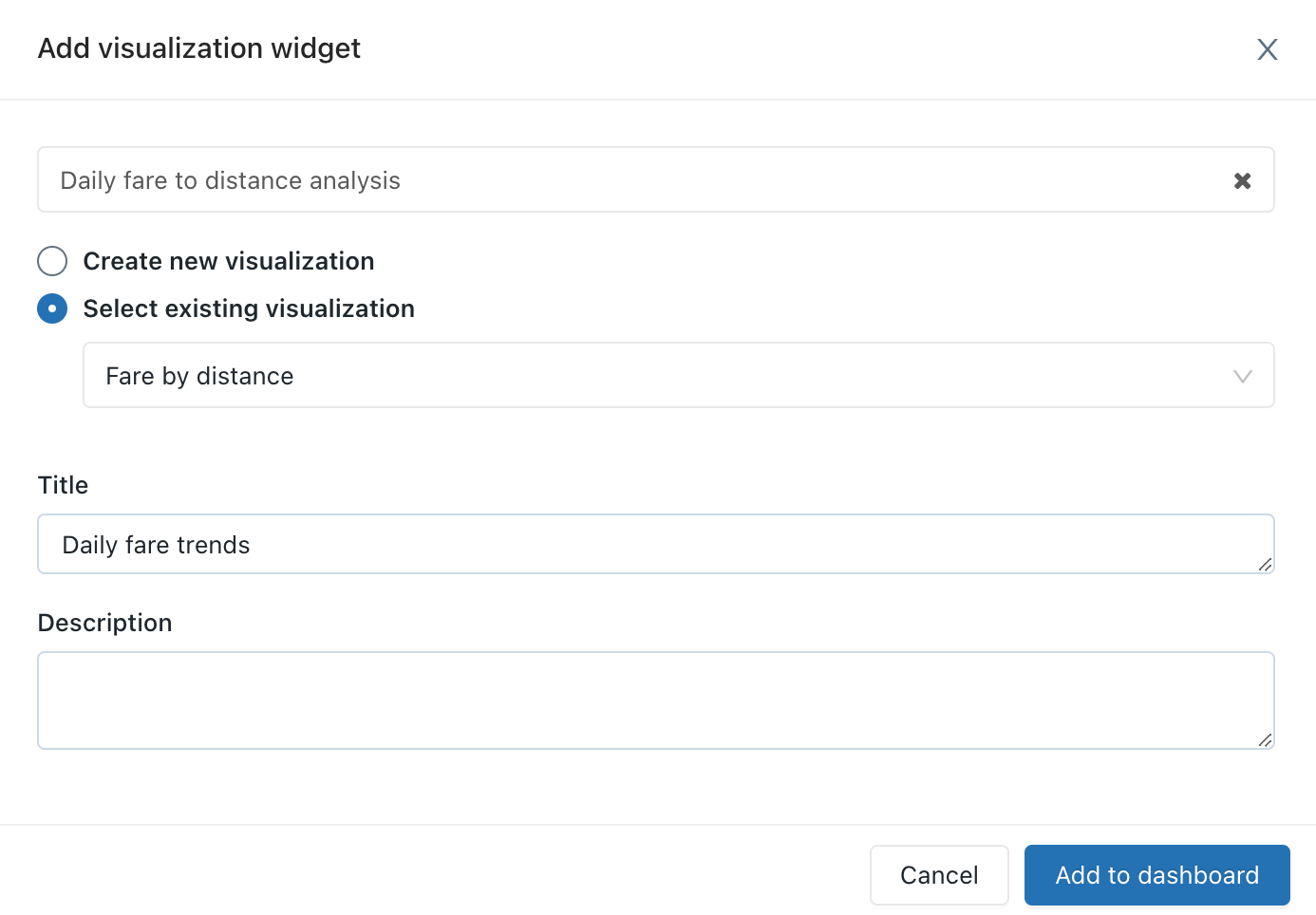
1. In the **Visualization Type** drop-down, select **Scatter**.
2. Change the visualization name to Fare by distance.
3. On the **General** tab, set the value for the **X column** to trip\_distance and set the value for the **Y columns** to fare\_amount.
4. In the **Group by** drop-down, set the value to day\_of\_week.
5. On the **X axis** tab, set the **Name** value to Trip distance (miles).
6. On the **Y axis** tab, set the **Name** value to Fare Amount (USD).
7. Click **Save**

The saved chart displays in the SQL editor.



## Create a dashboard using these visualizations

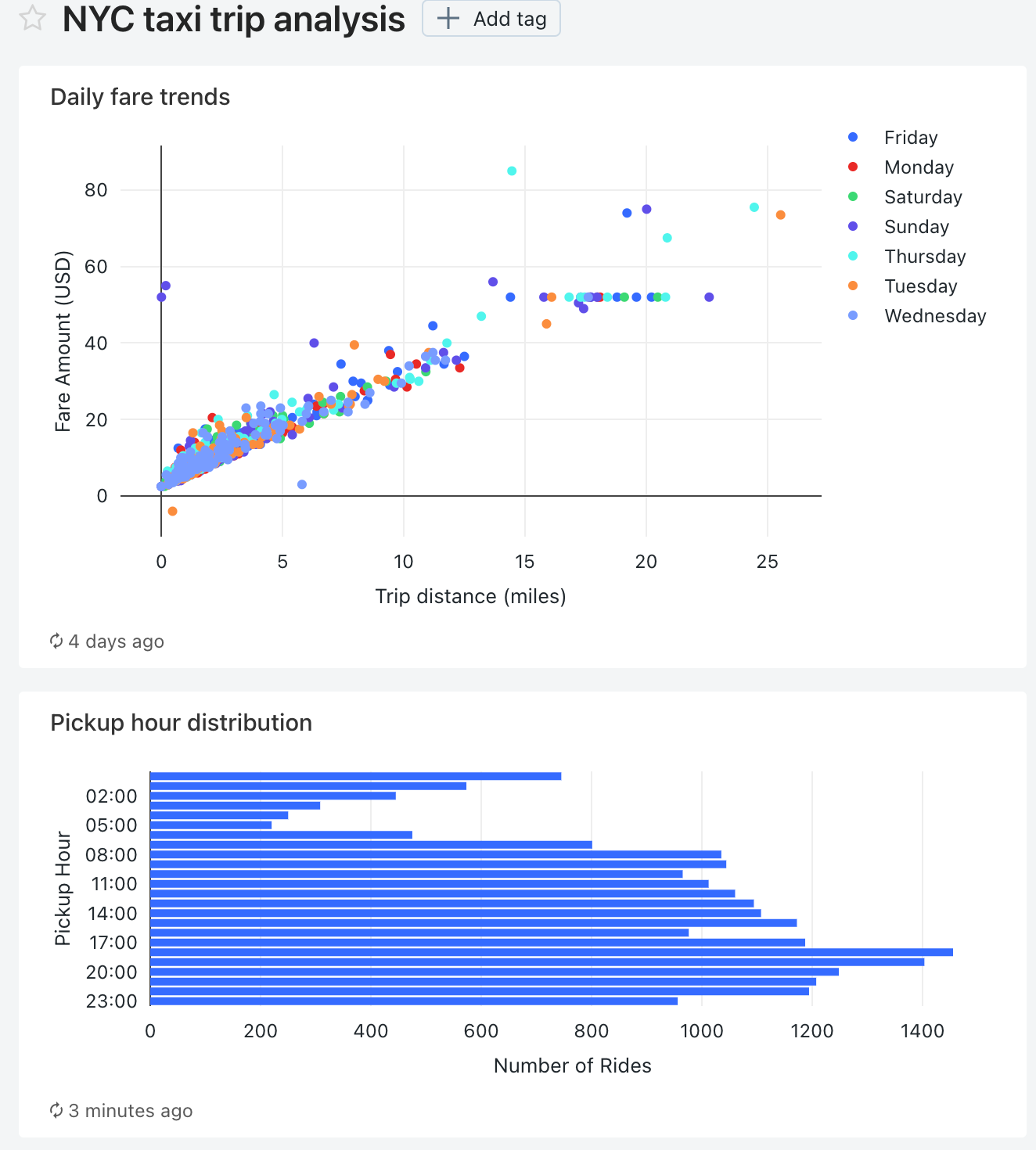
1. Click New Icon **New** in the sidebar and select **Dashboard**.
2. Set the dashboard name to NYC taxi trip analysis.
3. Click **Save**.
4. In the **Choose warehouse** drop-down list, select **Serverless Starter Warehouse**. If serverless is not enabled for your workspace, choose **Starter Warehouse**.
5. In the **Add** drop-down list, click **Visualization**.
6. In the **Add visualization widget** window, select the **Daily fare to distance analysis** query.
7. In the **Select existing visualization** list, select **Fare by distance**.
8. In the **Title** text box, enter Daily fare trends.



1. Click **Add to Dashboard**.

The Daily fare trends visualization appears on the dashbard design surface.

1. In the **Add** drop-down list to add a second widget to the dashboard, and then click **Visualization**.
2. In the **Add visualization widget** window, select the **Pickup hour** query.
3. In the **Select existing visualization** list, select **Bar chart**.
4. In the **Title** text box, enter Pickup hour distribution.
5. Click **Add to Dashboard**.
6. Resize this visualization to match the width of the first visualization in the dashboard.
7. Click **Done Editing**.



## Add a pickup zip code parameter to each query

1. In SQL editor, open the **Daily fare to distance analysis** query.
2. Add the following WHERE clause to the **Daily fare to distance analysis** query to filter the query by pickup zip code.

SQLCopy

WHERE

pickup\_zip IN ({{ pickupzip }})

1. In the **pickupzip** text box, enter 10018 and then click **Apply changes** to execute the query with the pickup zip code parameter.
2. Click **Save**.
3. Open the **Pickup hour** query.
4. Add the following WHERE clause to the **Pickup hour** query to filter the query by the pickup zip code. Add this clause before the GROUP BY clause.

SQLCopy

WHERE

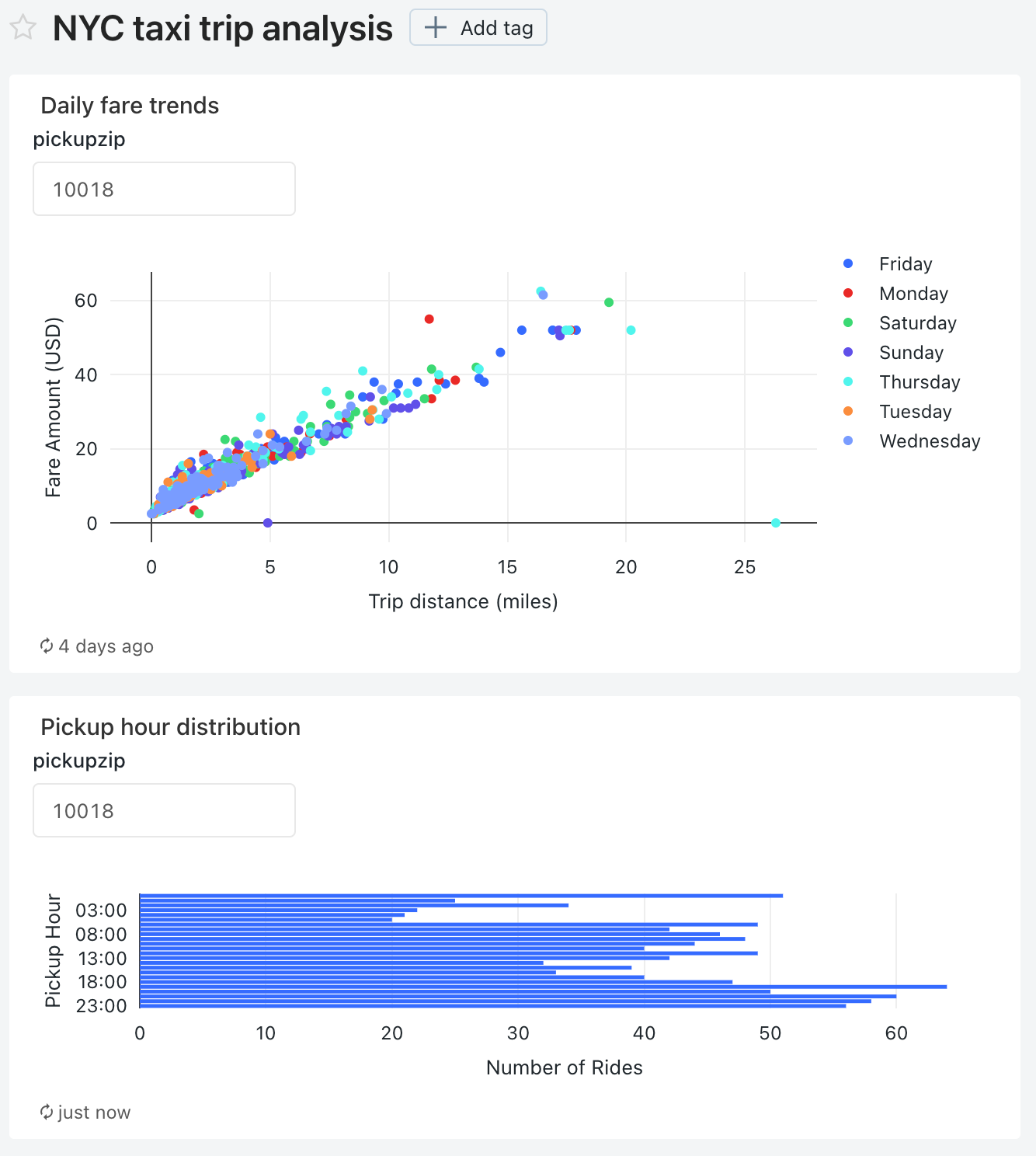
pickup\_zip IN ({{ pickupzip }})

1. In the **pickupzip** text box, enter 10018 and then click **Apply changes** to execute the query with the pickup zip code filter.
2. Click **Save**.

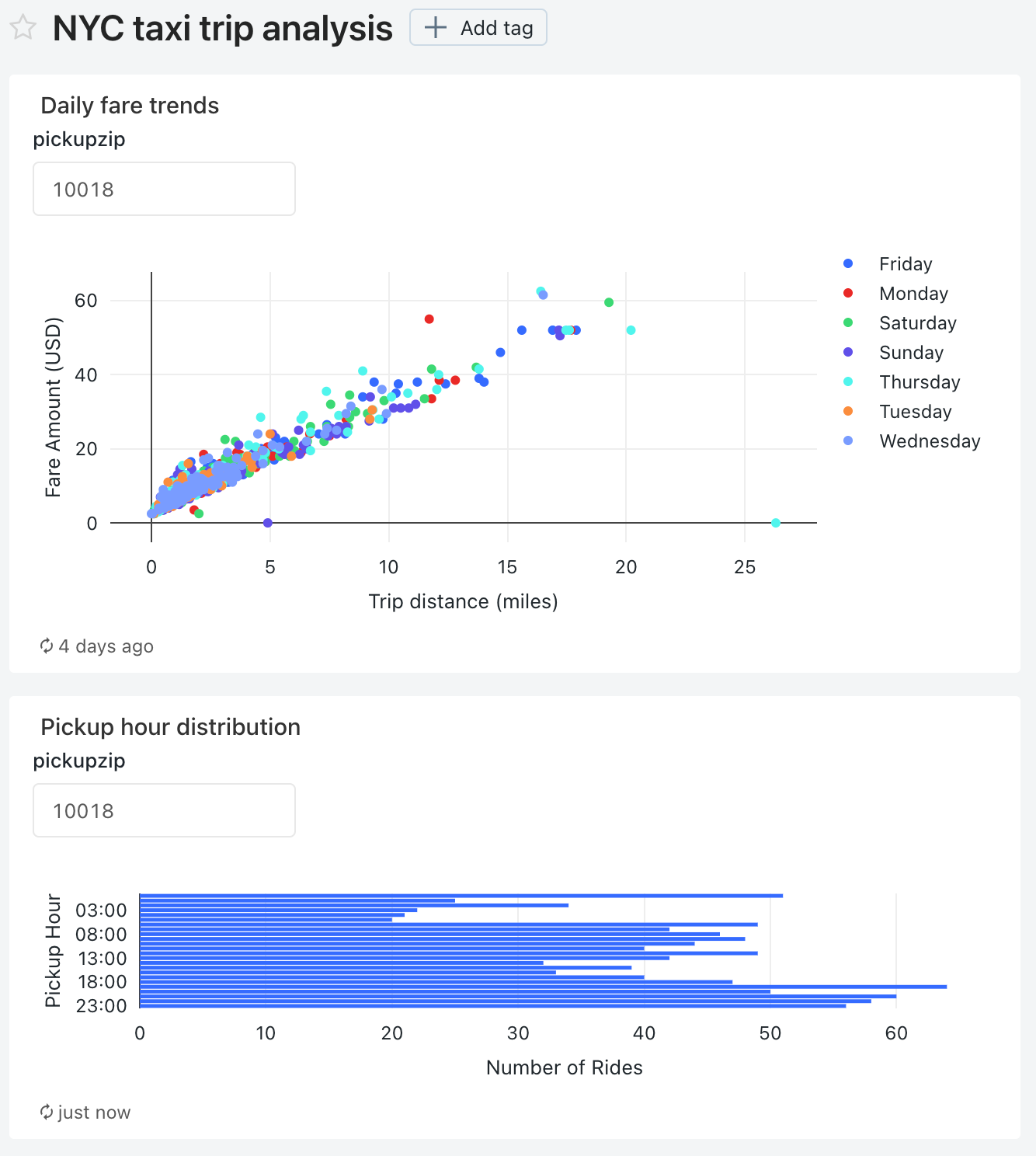
## Update the dashboard to use a dashboard parameter

1. Open the **NYC taxi trip analysis** dashboard.

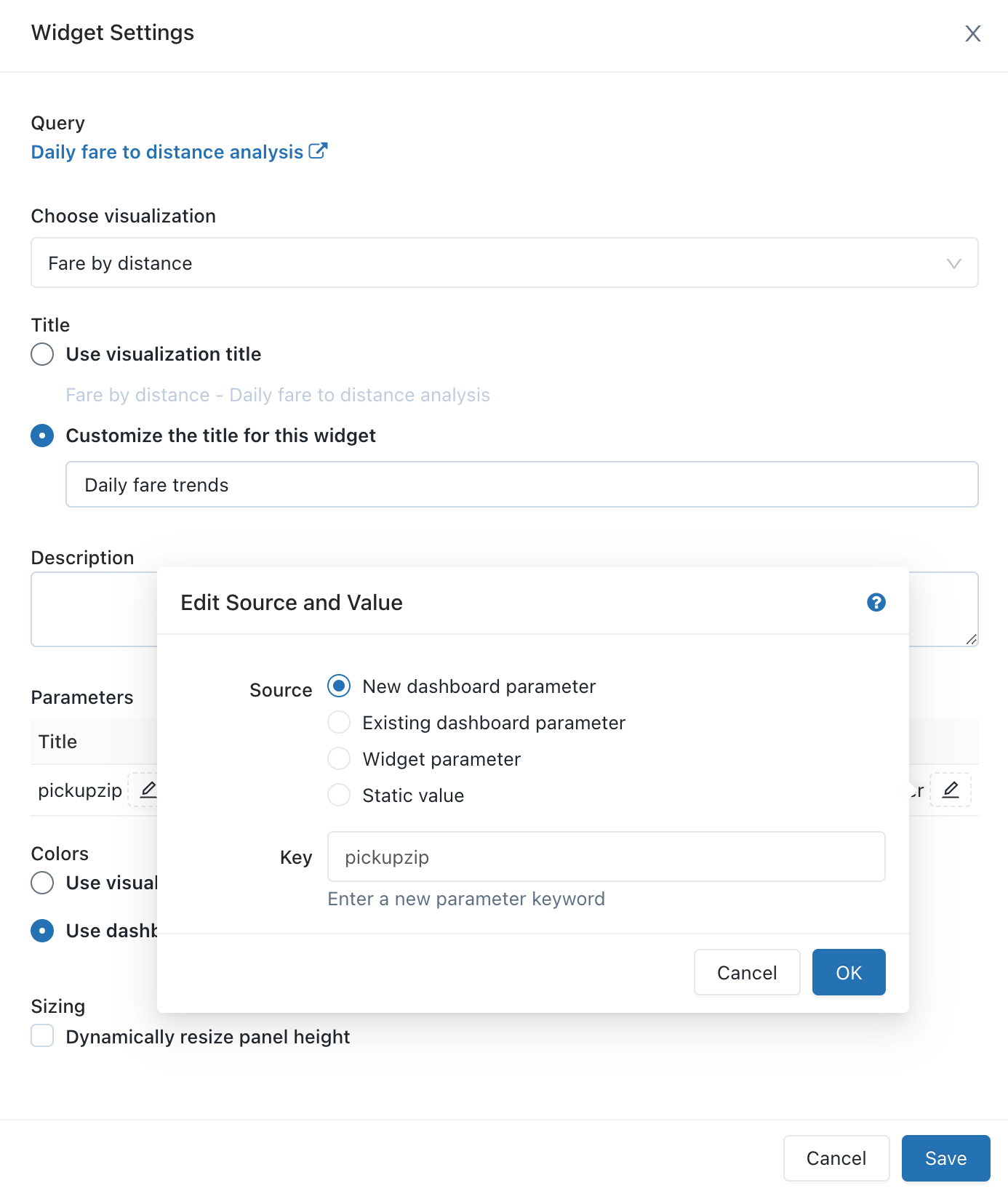
Each of the visualizations now includes a parameter for the pickup zip code.



1. Click the kebab menu Vertical Ellipsis for this dashboard and then click **Edit**.
2. Click the kebab menu Vertical Ellipsis for **Daily fare trends** visualization and then click **Change widget settings**.
3. In the **Parameters** section, click the pencil icon Edit icon for the **Widget parameter** in the **Value** field.



1. In the **Edit source and Value** window, change the **Source** to **New dashboard parameter**.



1. Click **OK** and then click **Save**.

The **pickupzip** dashboard parameter appears and the widget parameter for the **Daily fare trends** visualization no longer appears.

* 1. Click the kebab menu Vertical Ellipsis for **Pickup hour distribution** visualization and then click **Change widget settings**.

1. In the **Parameters** section, click the pencil icon Edit icon for the **Widget parameter** in the **Value** field.
2. In the **Edit source and Value** window, change the **Source** to **Existing dashboard parameter**.
3. Verify that **pickupzip** is selected as the **Key** value.
4. Click **OK** and then click **Save**.

The widget parameter for the **Pickup hour distribution** visualization no longer appears.

1. Click **Done editing**.
2. Change the value of the **pickupzip** dashboard parameter to 10017 and then click **Apply changes**.

The data in each of the vizualizations now displays the data for pickups in the 10017 zip code.

