# Load data in Spark notebooks

**Note:**

You are not required to complete the processes, tasks, activities, or steps presented in this example. The various samples provided are for illustrative purposes only and it's likely that if you try this out you will encounter issues in your system.

## Ingest and explore Parquet files from a data lake with Apache Spark for Azure Synapse

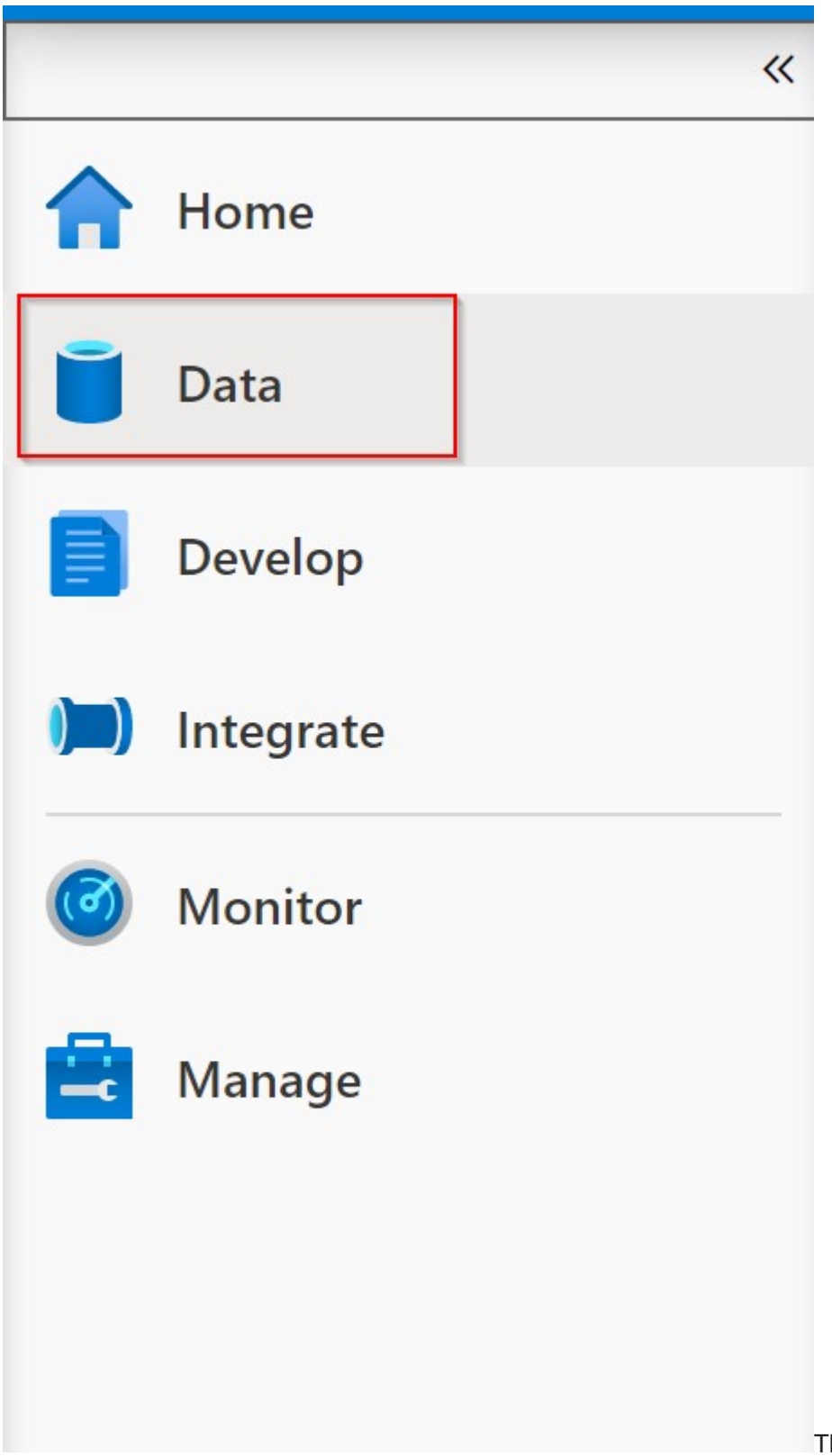
Tailwind Traders has Parquet files stored in their data lake. They need to quickly access the files and explore them using Apache Spark.

You recommend that they use the Data hub to view the Parquet files in the connected storage account, and then use the *new notebook* context menu to create a new Synapse notebook that loads a Spark DataFrame with the contents of a selected Parquet file.

You can complete this task using the following steps:

1. Open the Azure Synapse Studio.

2. Select the **Data** hub.

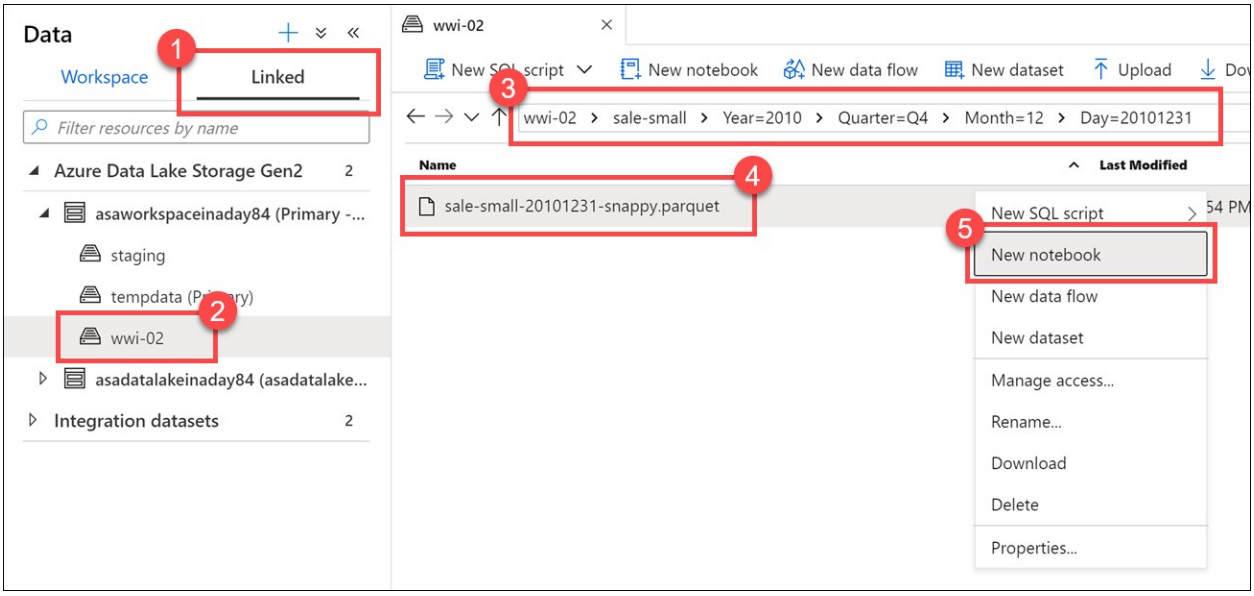


The data hub is highlighted.

3. Select the **Linked** tab **(1)** and expand the primary data lake storage account (*the name may differ from what you see here; it is the first storage account listed*).

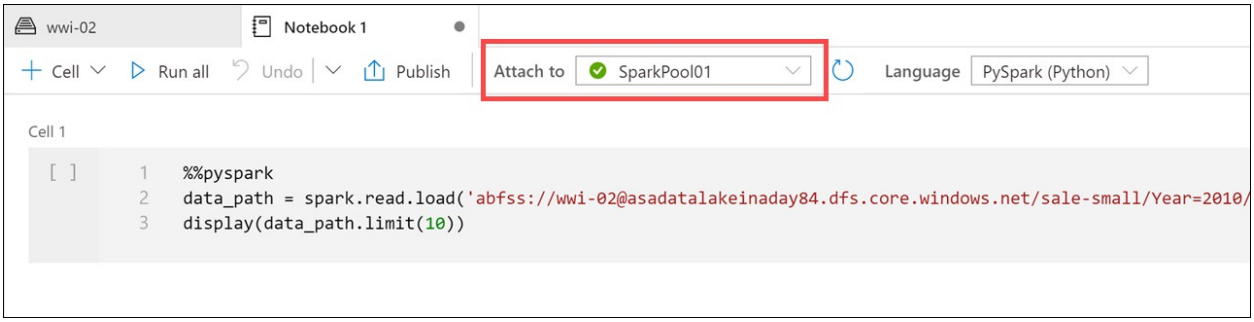
Select the **wwi-02** container **(2)** and browser to the *sale-small/Year=2010/Quarter=Q4/Month=12/Day=20101231* folder **(3)**.

Right-click the Parquet file **(4)** and select **New notebook (5)**.




The Parquet file is displayed as described.
A new notebook is generated with PySpark code to load the data in a Spark DataFrame and display 100 rows with the header.

4. Ensure the Spark pool is attached to the notebook.



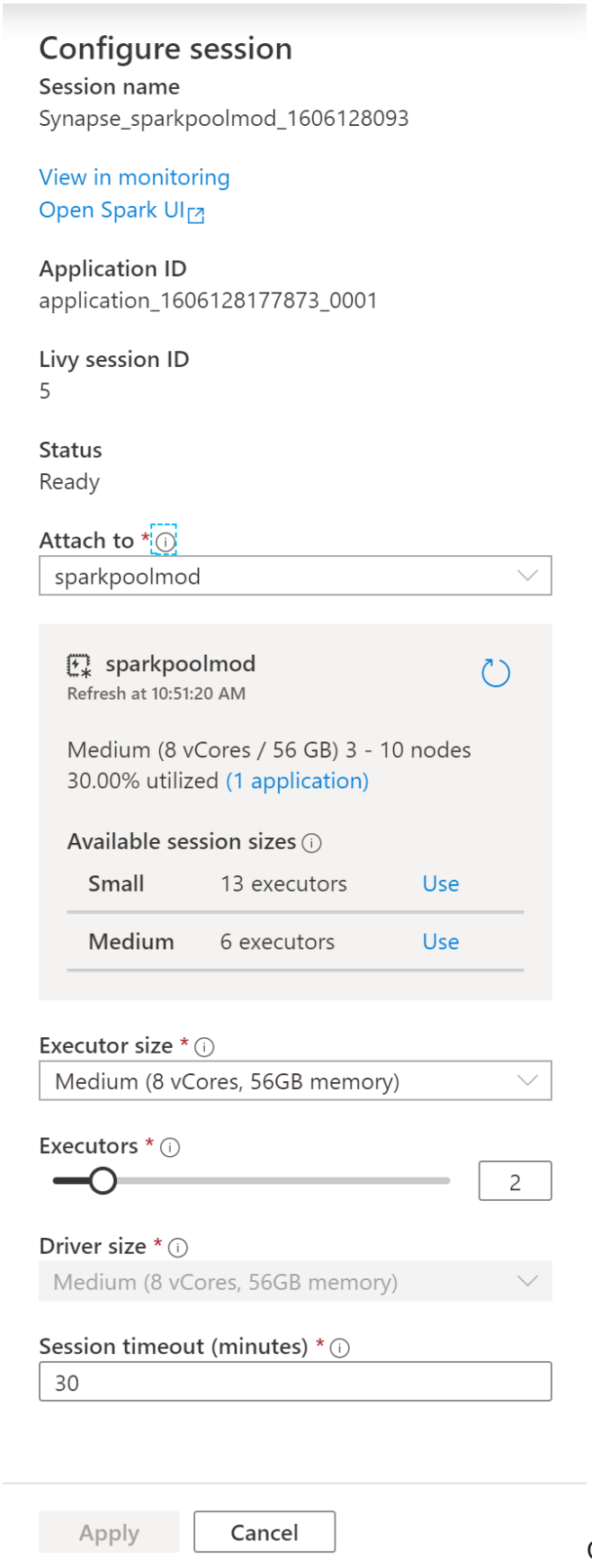

The Spark pool is highlighted.
The Spark pool provides the compute for all notebook operations. If you look at the bottom of the notebook, you can see that the pool has not started.

When you run a cell in the notebook while the pool is idle, the pool will start and allocate resources. It is a one-time operation until the pool autopauses from being idle for too long.


The Spark pool is in a paused state.

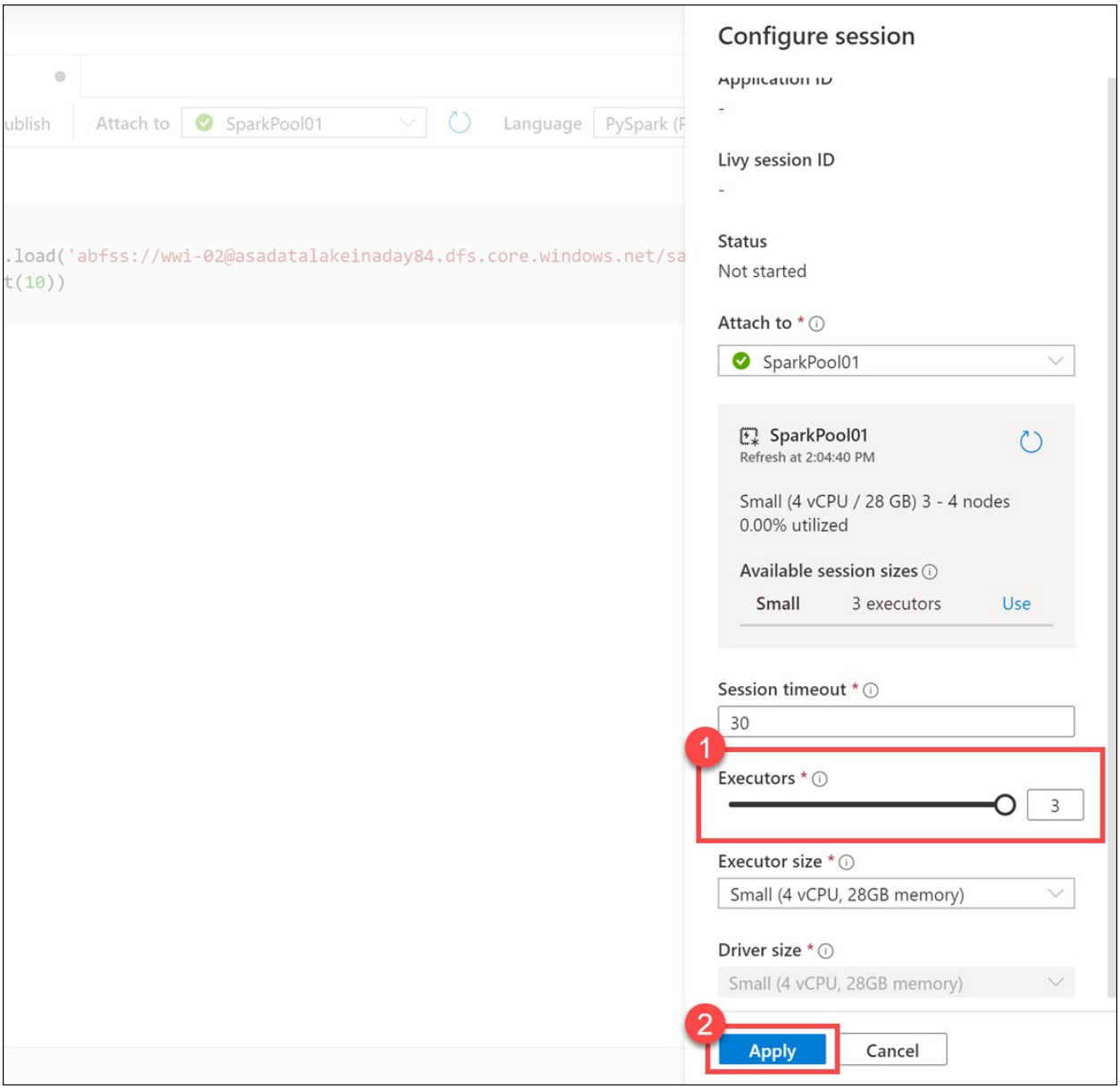
**Note:** *The auto-pause settings are configured on the Spark pool configuration in the Manage hub.*

5. Select **Configure session** at the bottom-left of the notebook to change the Spark configuration for this session.




Configure session.

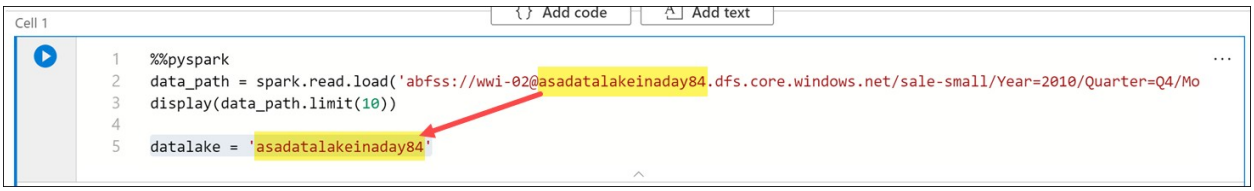
6. Set the number of **Executors** to **3 (1)**, then select **Apply (2)**.




The form is displayed.
We have just set the number of executors allocated to **SparkPool01** for the session.

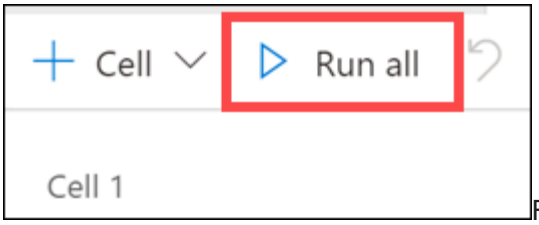
7. Add the following code beneath the code in the cell to define a variable named *datalake* whose value is the name of the primary storage account (replace the *REPLACE_WITH_YOUR_DATALAKE_NAME* value with the name of the storage account in line **2)**:

```
1
datalake = 'REPLACE_WITH_YOUR_DATALAKE_NAME'
```

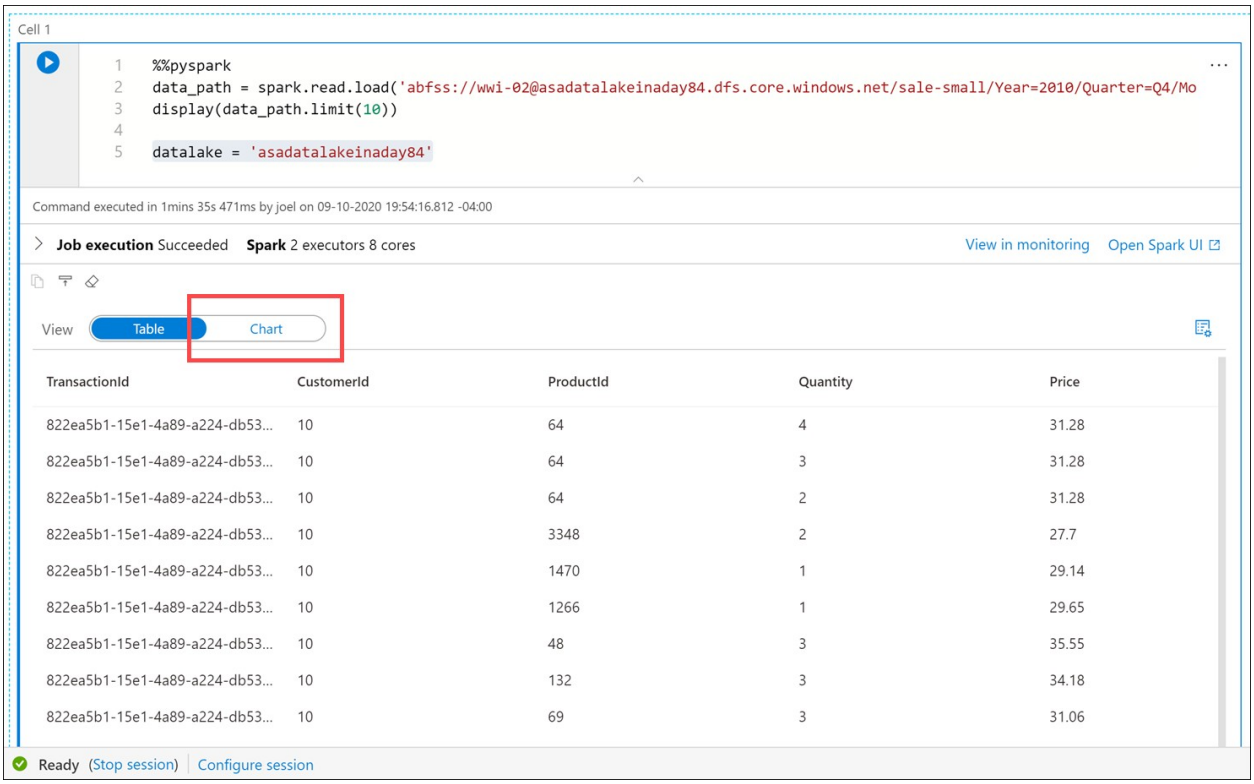



The variable value is updated with the storage account name.
This variable will be used in a couple cells later on.

8. Select **Run all** on the notebook toolbar to execute the notebook.

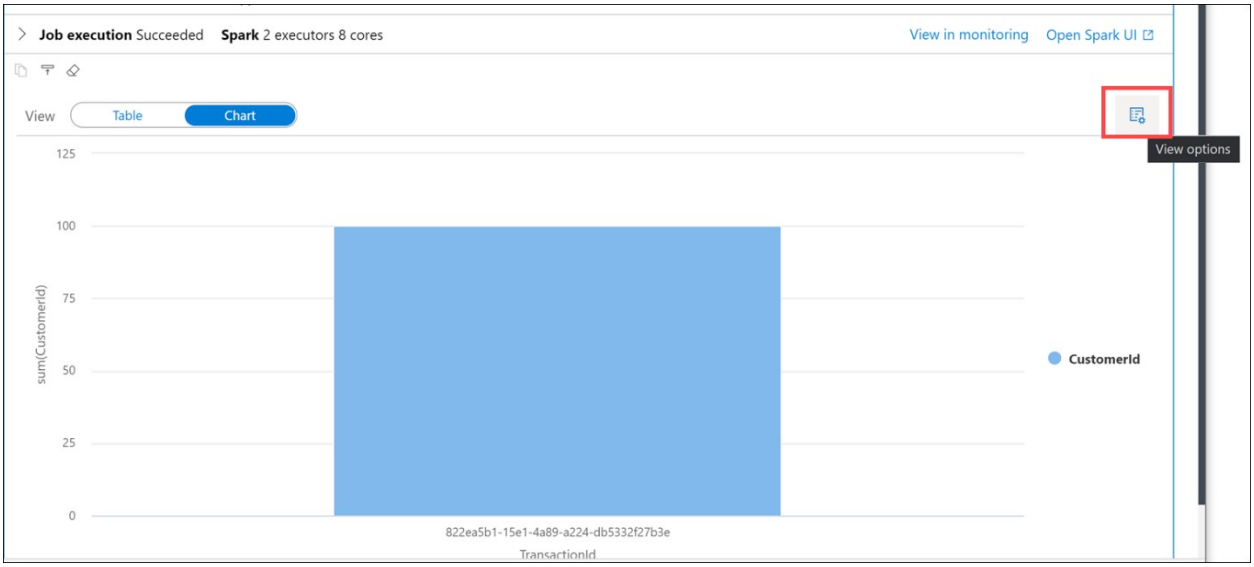



Run all is highlighted.
**Note:** The first time you run a notebook in a Spark pool, Synapse creates a new session. This can take approximately 3-5 minutes. To run just the cell, either hover over the cell and select the *Run cell* icon to the left of the cell, or select the cell and then type **Ctrl+Enter** on your keyboard. 9. After the cell run is complete, change the View to **Chart** in the cell output.



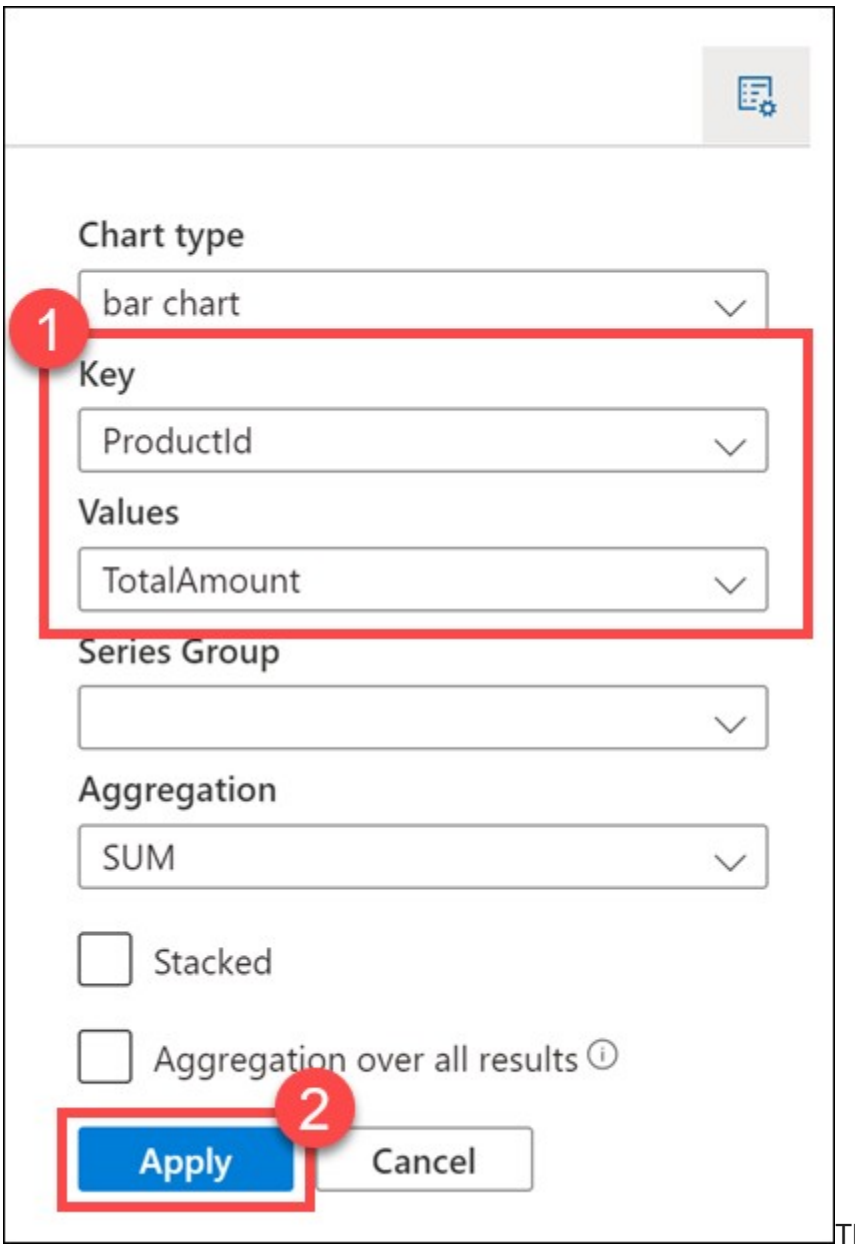

The Chart view is highlighted.
By default, the cell outputs to a table view when you use the **display()** function. You can see in the output the sales transaction data stored in the Parquet file for December 31, 2010. Select the **Chart** visualization to see a different view of the data.

10. Select the **View options** button to the right.



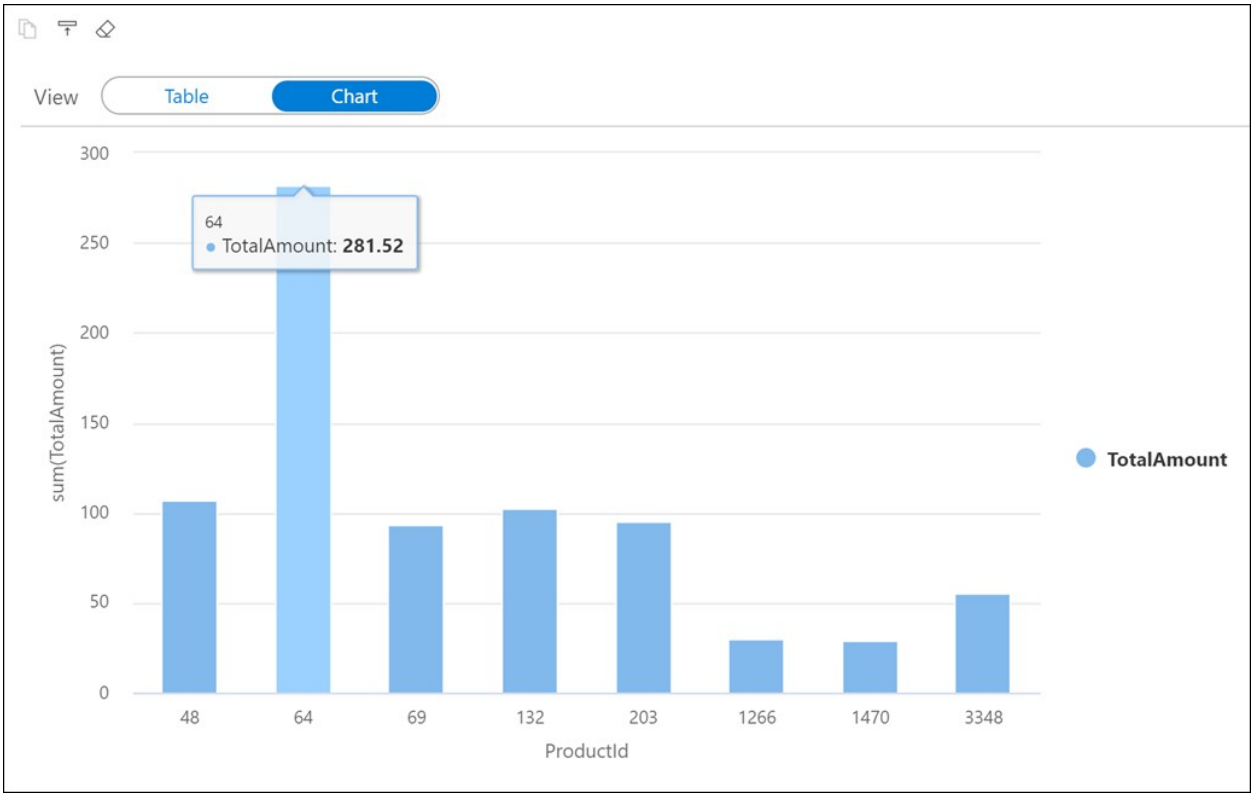
The button is highlighted.

11. Set Key to *ProductId* and Values to *TotalAmount* **(1)**, then select **Apply**.



The options are configured as described.

12. The chart visualization displays. Hover over the bars to view more detail.




The configured chart is displayed.

13. Create a new cell underneath by selecting **{} Add code** when hovering over the blank space at the bottom of the notebook.


The Add code button is highlighted underneath the chart.

14. The Apache Spark engine can analyze the Parquet files and infer the schema. To do so, enter the following code in the new cell and **run** it:

```
data_path.printSchema()
```

Your output should appear as follows:

```
root
    |-- TransactionId: string (nullable = true)
    |-- CustomerId: integer (nullable = true)
    |-- ProductId: short (nullable = true)
    |-- Quantity: short (nullable = true)
    |-- Price: decimal(29,2) (nullable = true)
    |-- TotalAmount: decimal(29,2) (nullable = true)
    |-- TransactionDate: integer (nullable = true)
    |-- ProfitAmount: decimal(29,2) (nullable = true)
    |-- Hour: byte (nullable = true)
    |-- Minute: byte (nullable = true)
    |-- StoreId: short (nullable = true)
```

Apache Spark evaluates the file contents to infer the schema. This automatic inference is sufficient for data exploration and most transformation tasks. However, when you load data to an external resource like a SQL pool table, you may need to declare your own schema and apply that to the dataset. For now, the schema looks good.

15. Next, use the dataframe to use aggregates and grouping operations in order to better understand the data. Create a new cell and enter the following, then **run** the cell:

```
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *
```

```
profitByDateProduct = (data_path.groupBy("TransactionDate","ProductId")
    .agg(
        sum("ProfitAmount").alias("(sum)ProfitAmount"),
        round(avg("Quantity"), 4).alias("(avg)Quantity"),
        sum("Quantity").alias("(sum)Quantity"))
    .orderBy("TransactionDate"))
display(profitByDateProduct.limit(100))
```

**Note:** We import required Python libraries to use aggregation functions and types defined in the schema to successfully execute the query.

The output shows the same data we saw in the chart above, but now with **sum** and **avg** aggregates **(1)**. Notice that we use the **alias** method **(2)** to change the column names.