

Data Integration Part 2

<!-- Create e2e pipeline for initial load & update

Data Prep: Handle bad data, file formats, join disparate sources

Troubleshoot: pipeline/activity failure (something beyond simply looking at the portal)

Optimize: trigger window, exec time

Triggering Pipelines

Creating schedule, tumbling window, event (blob vs ADLS)

Other cases: trigger via Function

- Native and non-standard ways to trigger a pipeline

- Set up change feed-triggered function (CSharp script via portal). Edit a document in Cosmos DB (maybe a pipeline parameter), which triggers the function

- Maybe trigger the pipeline via REST call and shown here: <https://github.com/solliancenet/azure-synapse-analytics/blob/master/infrastructure/asaexp/setup/ASAEsp%20-%20Import%20SQL%20Pool%20Tables.ps1#L114>

Spark notebook to connect to the wvi.UserTopProductPurchases Synapse database table.

- Have user right-click the table, choose notebook, then new Spark notebook

- Execute code to find the top 5 products for each user, based on which ones are both preferred and top, and have the most purchases in past 12 months

- Top 5 products overall -->

Resource naming throughout this lab

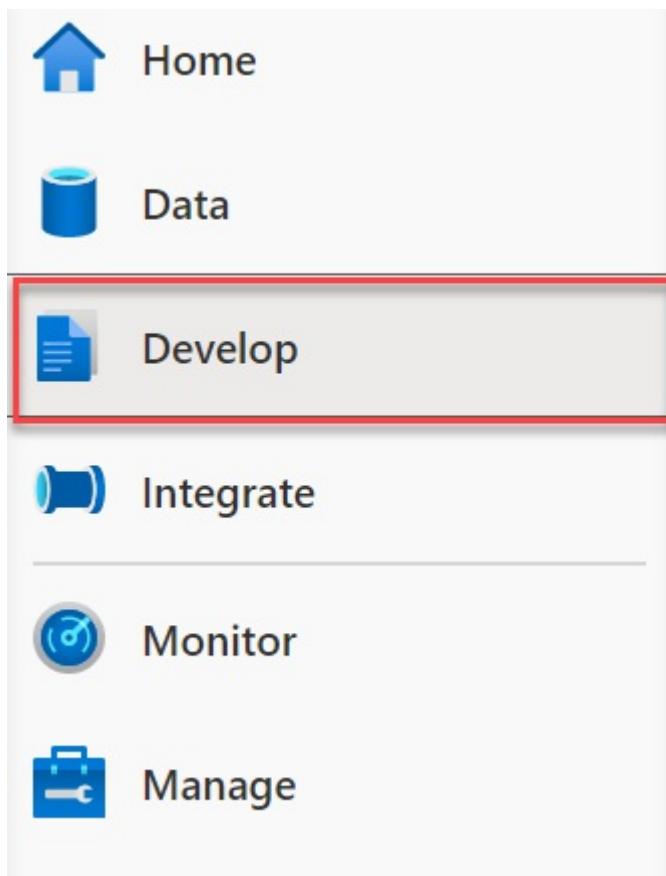
For the remainder of this guide, the following terms will be used for various ASA-related resources (make sure you replace them with actual names and values):

Azure Synapse Analytics Resource	To be referred to
Workspace resource group	WorkspaceResourceGroup
Workspace / workspace name	Workspace
Primary Storage Account	PrimaryStorage
Default file system container	DefaultFileSystem
SQL Pool	SqlPool101

Exercise 1: Create datasets and SQL tables

Task 1: Create SQL tables

1. Navigate to the **Develop** hub.



2. From the **Develop** menu, select the + button and choose **SQL Script** from the context menu.

The screenshot shows the **Develop** menu in the Azure Data Studio toolbar. The **SQL script** option is highlighted with a red box. Other options visible in the dropdown menu include:

- SQL script (highlighted)
- Notebook
- Data flow
- Spark job definition
- Power BI report
- Import

3. In the toolbar menu, connect to the **SQL Pool** assigned for your environment.



4. In the query window, replace the script with the following to create a new table for the Campaign Analytics CSV file:

```
CREATE TABLE [wwi].[CampaignAnalytics]
(
    [Region] [nvarchar](50) NOT NULL,
    [Country] [nvarchar](30) NOT NULL,
    [ProductCategory] [nvarchar](50) NOT NULL,
    [CampaignName] [nvarchar](500) NOT NULL,
    [Revenue] [decimal](10,2) NULL,
    [RevenueTarget] [decimal](10,2) NULL,
    [City] [nvarchar](50) NULL,
    [State] [nvarchar](25) NULL
)
WITH
(
    DISTRIBUTION = HASH ( [Region] ),
    CLUSTERED COLUMNSTORE INDEX
)
```

5. Select **Run** from the toolbar menu to execute the SQL command.



6. In the query window, replace the script with the following to create a new table for the Sales Parquet files:

```
CREATE TABLE [wwi].[Sale]
(
    [TransactionId] [uniqueidentifier] NOT NULL,
    [CustomerId] [int] NOT NULL,
    [ProductId] [smallint] NOT NULL,
    [Quantity] [smallint] NOT NULL,
    [Price] [decimal](9,2) NOT NULL,
    [TotalAmount] [decimal](9,2) NOT NULL,
    [TransactionDate] [int] NOT NULL,
    [ProfitAmount] [decimal](9,2) NOT NULL,
    [Hour] [tinyint] NOT NULL,
    [Minute] [tinyint] NOT NULL,
    [StoreId] [smallint] NOT NULL
)
WITH
(
    DISTRIBUTION = HASH ( [CustomerId] ),
    CLUSTERED COLUMNSTORE INDEX,
    PARTITION
    (
        [TransactionDate] RANGE RIGHT FOR VALUES (20100101, 20100201, 20100301, 20100401, 20100501,
        20100601, 20100701, 20100801, 20100901, 20101001, 20101101, 20101201, 20110101, 20110201, 20110301,
        20110401, 20110501, 20110601, 20110701, 20110801, 20110901, 20111001, 20111101, 20111201, 20120101,
        20120201, 20120301, 20120401, 20120501, 20120601, 20120701, 20120801, 20120901, 20121001, 20121101,
        20121201, 20130101, 20130201, 20130301, 20130401, 20130501, 20130601, 20130701, 20130801, 20130901,
        20131001, 20131101, 20131201, 20140101, 20140201, 20140301, 20140401, 20140501, 20140601, 20140701,
        20140801, 20140901, 20141001, 20141101, 20141201, 20150101, 20150201, 20150301, 20150401, 20150501,
        20150601, 20150701, 20150801, 20150901, 20151001, 20151101, 20151201, 20160101, 20160201, 20160301,
        20160401, 20160501, 20160601, 20160701, 20160801, 20160901, 20161001, 20161101, 20161201, 20170101,
        20170201, 20170301, 20170401, 20170501, 20170601, 20170701, 20170801, 20170901, 20171001, 20171101,
        20171201, 20180101, 20180201, 20180301, 20180401, 20180501, 20180601, 20180701, 20180801, 20180901,
        20181001, 20181101, 20181201, 20190101, 20190201, 20190301, 20190401, 20190501, 20190601, 20190701,
        20190801, 20190901, 20191001, 20191101, 20191201)
    )
)
```

7. Select **Run** from the toolbar menu to execute the SQL command.

8. In the query window, replace the script with the following to create a new table for the user reviews contained within the user profile data in Azure Cosmos DB:

```

CREATE TABLE [wwi].[UserProductReviews]
(
    [UserId] [int] NOT NULL,
    [ProductId] [int] NOT NULL,
    [ReviewText] [nvarchar](1000) NOT NULL,
    [ReviewDate] [datetime] NOT NULL
)
WITH
(
    DISTRIBUTION = HASH ( [ProductId] ),
    CLUSTERED COLUMNSTORE INDEX
)

```

9. Select **Run** from the toolbar menu to execute the SQL command.

10. In the query window, replace the script with the following to create a new table that joins users' preferred products stored in Azure Cosmos DB with top product purchases per user from the e-commerce site, stored in JSON files within the data lake:

```

CREATE TABLE [wwi].[UserTopProductPurchases]
(
    [UserId] [int] NOT NULL,
    [ProductId] [int] NOT NULL,
    [ItemsPurchasedLast12Months] [int] NULL,
    [IsTopProduct] [bit] NOT NULL,
    [IsPreferredProduct] [bit] NOT NULL
)
WITH
(
    DISTRIBUTION = HASH ( [UserId] ),
    CLUSTERED COLUMNSTORE INDEX
)

```

11. Select **Run** from the toolbar menu to execute the SQL command.

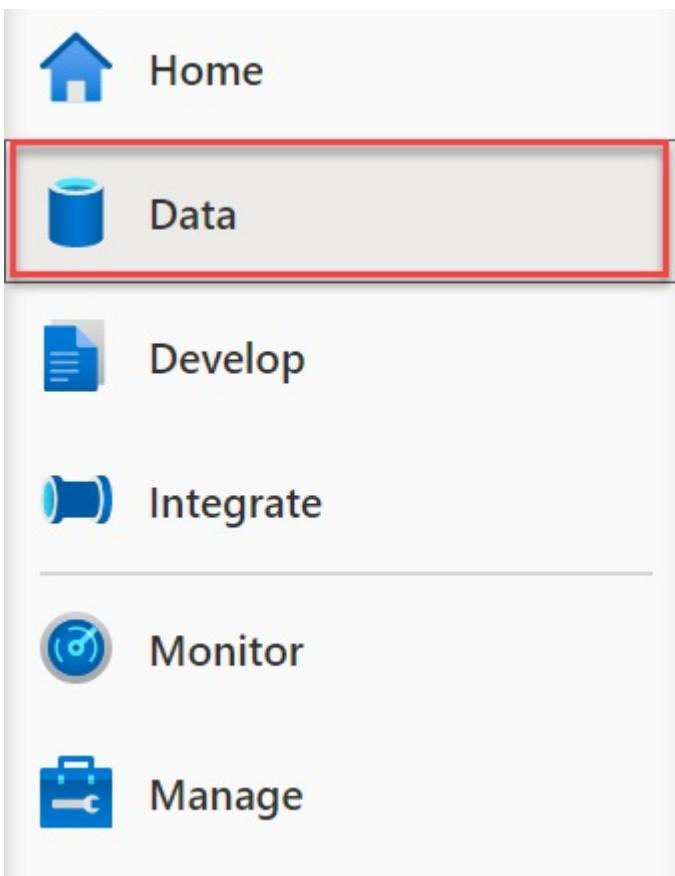
Task 2: Create campaign analytics datasets

Your organization was provided a poorly formatted CSV file containing marketing campaign data. The file was uploaded to the data lake and now it must be imported into the data warehouse.

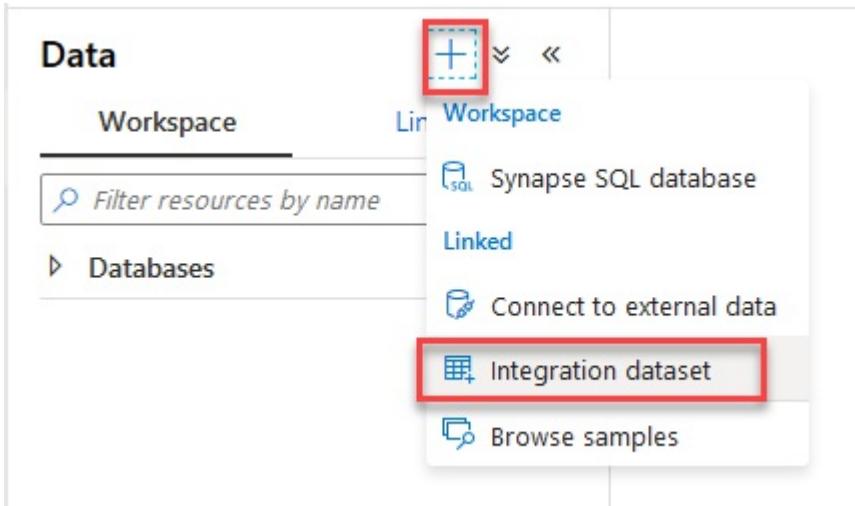
Region	Country	Product_Category	Campaign_Name	Revenue	Revenue_T	City	State	
Europe	Germany	Apparel and Footwear	Fun with Colors	\$14\	865 \$15\		960	
Far West	US	Books	EnjoyTheMoment; \$14\	992 \$15\		699	San Diego	California
Europe	Germany	Apparel and Footwear	Fall into Winter	\$5\	117 \$8\		713	
Far West	US	Books	EnjoyTheMoment; \$9\	935 \$15\		232	San Diego	California
Europe	France	Apparel and Footwear	Enjoy the Moment	\$13\	221 \$8\		584	
Far West	US	Books	EnjoyTheMoment; \$15\	119 \$17\		269	San Diego	California
Europe	UK	Lighting	Fall into Winter	\$5\	117 \$9\		305	
Far West	US	Books	EnjoyTheMoment; \$15\	740 \$7\		685	San Diego	California
South America	Mexico	Electronics	Be Unique	\$16\	240 \$16\		38	
Far West	US	Books	EnjoyTheMoment; \$14\	778 \$13\		122	San Diego	California
North & Central America	USA	DÃ©cor	Spring into Summe	\$6\	689 \$13\		88	
Far West	US	Books	EnjoyTheMoment; \$10\	296 \$7\		313	San Diego	California
South America	Mexico	Apparel and Footwear	Enjoy the Moment	\$13\	98 \$5\		663	
Far West	US	Books	EnjoyTheMoment; \$14\	605 \$18\		971	San Diego	California
South America	Brazil	DÃ©cor	Fun with Colors	\$15\	142 \$7\		147	
Far West	US	Books	EnjoyTheMoment; \$14\	328 \$15\		577	San Diego	California
South America	Mexico	Exercise	Spring into Summe	\$17\	637 \$6\		876	
Far West	US	Books	EnjoyTheMoment; \$11\	247 \$10\		339	San Diego	California
South America	Mexico	DÃ©cor	Be Unique	\$8\	284 \$9\		840	

Issues include invalid characters in the revenue currency data, and misaligned columns.

1. Navigate to the **Data** hub.



2. With the Workspace tab selected under Data, select + in the toolbar, then select **Integration dataset** to create a new dataset.



3. Create a new **Azure Data Lake Storage Gen2** integration dataset with the **DelimitedText** format type with the following characteristics:

- **Name:** Enter `asal400_campaign_analytics_source`.
- **Linked service:** Select the `asadatalakeSUFFIX` linked service.
- **File path:** Browse to the `wwi-02/campaign-analytics/campaignanalytics.csv` path.
- **First row as header:** Leave unchecked. **We are skipping the header** because there is a mismatch between the number of columns in the header and the number of columns in the data rows.
- **Import schema:** Select `From connection/store`.

Set properties

i Choose a name for your dataset. This name can be updated at any time until it is published.

Name
asal400_campaign_analytics_source

Linked service *
asadatalake212045 

File path
wwi-02 / campaign-analytics / campaignanalytics.csv  | 

First row as header

Import schema
 From connection/store From sample file None

OK **Back** **Cancel**

4. After creating the dataset, navigate to its **Connection** tab. Leave the default settings. They should match the following configuration:

- **Compression type:** Select none.
- **Column delimiter:** Select Comma (,).
- **Row delimiter:** Select Auto detect (\r,\n, or \r\n).
- **Encoding:** Select Default (UTF-8).
- **Escape character:** Select Backslash (\).
- **Quote character:** Select Double quote (").
- **First row as header:** Leave unchecked.
- **Null value:** Leave the field empty.

Connection **Schema** **Parameters**

Linked service * asadatalake212045   

File path * wwi-02 / campaign-analytics / campaignanalytics.csv  | 

Compression type none

Column delimiter Comma (,)  

Row delimiter Auto detect (\r,\n, or \r\n)  

Encoding Default(UTF-8)

Escape character Backslash (\) 

Quote character Double quote (") 

First row as header

Null value

5. Select **Preview data**.

6. Preview data displays a sample of the CSV file. You can see some of the issues shown in the screenshot at the beginning of this task. Notice that since we are not setting the first row as the header, the header columns appear as the first row. Also, notice that the city and state values seen in the earlier screenshot do not appear. This is because of the mismatch in the number of columns in the header row compared to the rest of the file. We will exclude the first row when we create the data flow in the next exercise.

Prop_0	Prop_1	Prop_2	Prop_3	Prop_4	Prop_5	Prop_6	Prop_7
Region	Country	Product_Category	Campaign_Name	Revenue	Revenue_Target	City	State
Europe	Germany	Apparel and Footwear	Fun with Colors	\$14\	865.00	\$15\	960.00
Far West	US	Books	EnjoyTheMoment; BeUnique; TailoredForYou	\$14\	992.00	\$15\	699.00
Europe	Germany	Apparel and Footwear	Fall into Winter	\$5\	117.00	\$8\	713.00
Far West	US	Books	EnjoyTheMoment; BeUnique; TailoredForYou	\$9\	935.00	\$15\	232.00
Europe	France	Apparel and Footwear	Enjoy the Moment	\$13\	221.00	\$8\	584.00
Far West	US	Books	EnjoyTheMoment; BeUnique; TailoredForYou	\$15\	119.00	\$17\	269.00
Europe	UK	Lighting	Fall into Winter	\$5\	117.00	\$9\	305.00
Far West	US	Books	EnjoyTheMoment; BeUnique; TailoredForYou	\$15\	740.00	\$7\	685.00
South America	Mexico	Electronics	Be Unique	\$16\	240.00	\$16\	038.00

7. Create a new **Azure Synapse Analytics** integration dataset with the following characteristics:

- o **Name:** Enter `asal400_wwi_campaign_analytics_asa`.
- o **Linked service:** Select the `SqlPool01` service.
- o **Table name:** Select `wwi.CampaignAnalytics`.
- o **Import schema:** Select `From connection/store`.

Set properties

i Choose a name for your dataset. This name can be updated at any time until it is published.

Name

Linked service *

▼

[Edit connection](#)

Table name



Import schema

From connection/store None

OK

[Back](#)

[Cancel](#)

Task 3: Create user profile datasets

User profile data comes from two different data sources. In lab 1, you created datasets for these sources:

asal400_ecommerce_userprofiles_source and asal400_customerprofile_cosmosdb (*complete Task 4 below if you did not complete lab 1*). The customer profile data from an e-commerce system that provides top product purchases for each visitor of the site (customer) over the past 12 months is stored within JSON files in the data lake. User profile data containing, among other things, product preferences and product reviews is stored as JSON documents in Cosmos DB.

In this task, you'll create datasets for the SQL tables that will serve as data sinks for data pipelines you'll create later in this lab.

1. Create a new **Azure Synapse Analytics** integration dataset with the following characteristics:

- o **Name:** Enter asal400_wwi_userproductreviews_asa.
- o **Linked service:** Select the SqlPool01 service.
- o **Table name:** Select wwi.UserProductReviews.
- o **Import schema:** Select From connection/store.

Set properties

i Choose a name for your dataset. This name can be updated at any time until it is published.

Name

Linked service *

 ▼

[Edit connection](#)

Table name

 ↻ Edit

Import schema

From connection/store None

OK

[Back](#)

[Cancel](#)

2. Create a new **Azure Synapse Analytics** integration dataset with the following characteristics:

- o **Name:** Enter `asal400_wwi_usertopproductpurchases_asa`.
- o **Linked service:** Select the `SqlPool01` service.
- o **Table name:** Select `wwi.UserTopProductPurchases`.
- o **Import schema:** Select `From connection/store`.

Set properties

i Choose a name for your dataset. This name can be updated at any time until it is published.

Name

Linked service *

 ▼

[Edit connection](#)

Table name

 ↻ Edit

Import schema

From connection/store None

OK

[Back](#)

[Cancel](#)

3. Select **Publish all** to save your new resources.

The screenshot shows the Azure Data Factory interface. At the top, there's a toolbar with a 'Publish all' button (highlighted with a red box), a 'Validate all' button with a checkmark, and a 'Refresh' button. Below the toolbar, the main area has a 'Develop' tab selected. A search bar says 'Filter resources by name'. Under the 'Develop' tab, there's a section for 'SQL scripts' with a count of 23. To the right, a connection named 'ASAL4' is listed with a checkmark indicating it is valid.

Task 4: OPTIONAL - Create datasets from Lab 1

If you **did not** complete Exercise 1 in lab 1, where you configure the linked service and create datasets, complete the steps below to create two additional datasets for this lab (`asal400_ecommerce_userprofiles_source` and `asal400_customerprofile_cosmosdb`).

1. Create a new **Azure Cosmos DB (SQL API)** dataset with the following characteristics:

- o **Name:** Enter `asal400_customerprofile_cosmosdb`.
- o **Linked service:** Select the Azure Cosmos DB linked service.
- o **Collection:** Select `OnlineUserProfile01`.

Set properties

i Choose a name for your dataset. This name can be updated at any time until it is published.

Name

`asal400_customerprofile_cosmosdb`

Linked service *

`asacosmosdb01`

[Edit connection](#)

Collection

`OnlineUserProfile01`

[Edit](#)



Import schema

From connection/store None

2. After creating the dataset, navigate to its **Connection** tab, then select **Preview data**.

CosmosDB Collection (SQL API)
asal400_customerprofile_cosmosdb

Connection Schema Parameters

Linked service * asacosmosdb1 Test connection Edit + New

Collection OnlineUserProfile01 Refresh Preview data

Edit

3. Preview data queries the selected Azure Cosmos DB collection and returns a sample of the documents within. The documents are stored in JSON format and include a `userId` field, `cartId`, `preferredProducts` (an array of product IDs that may be empty), and `productReviews` (an array of written product reviews that may be empty). We will use this data in lab 2.

Preview data

Linked service: asacosmosdb01

Object: OnlineUserProfile01

```
[
  {
    "userId": 9079954,
    "cartId": "406a06af-e54f-42e9-aad8-9a36f2c7f8ca",
    "preferredProducts": [],
    "productReviews": [
      {
        "productId": 3965,
        "reviewText": "It only works when I'm Bahrain.",
        "reviewDate": "2019-01-15T19:04:42.5554783+00:00"
      },
      {
        "productId": 1287,
        "reviewText": "This Harbors works so well. It imperfectly improves my baseball by a lot.",
        "reviewDate": "2017-04-23T19:54:59.273694+00:00"
      },
      {
        "productId": 169,
        "reviewText": "one of my hobbies is antique-shopping. and when i'm antique-shopping this works great.",
        "reviewDate": "2020-03-23T20:52:59.5875906+00:00"
      }
    ],
    "id": "441589f6-6754-4f75-a04a-23191dbf72de",
    "_rid": "OC8bALmbB4kBAAAAAAA==",
    "_self": "dbs/OC8bAA==/colls/OC8bALmbB4k=/docs/OC8bALmbB4kBAAAAAAA==/",
    "_etag": "\"2101cde1-0000-0200-0000-5e969f4b0000\"",
    "_attachments": "attachments/",
    "_ts": 1586929483
  },
  {
    "userId": 9079747,
    "cartId": "5c4dc5dc-a585-41ec-8149-9133caa3a73a",
    "preferredProducts": [
      4235,
      3288,
      2756
    ],
    "productReviews": [
      ...
    ]
  }
]
```

4. Select the **Schema** tab, then select **Import schema**. Synapse Analytics evaluates the JSON documents within the collection and infers the schema based on the nature of the data within. Since we are only storing one document type in this collection, you will see the inferred schema for all documents within.

Connection Schema Parameters ^

Import schema Clear

Column name	Type
userId	123 integer
cartId	abc string
preferredProducts	[] integer[]
productReviews	[] object[]
productId	123 integer
reviewText	abc string
reviewDate	abc string

5. Create a new **Azure Data Lake Storage Gen2** dataset with the **JSON** format type with the following characteristics:

- **Name:** Enter `asal400_ecommerce_userprofiles_source`.
- **Linked service:** Select the `asadatalakeXX` linked service that already exists.
- **File path:** Browse to the `wwi-02/online-user-profiles-02` path.
- **Import schema:** Select `From connection/store`.

6. Select **Publish all** to save your new resources.

Develop + ⌂ << ASAL4

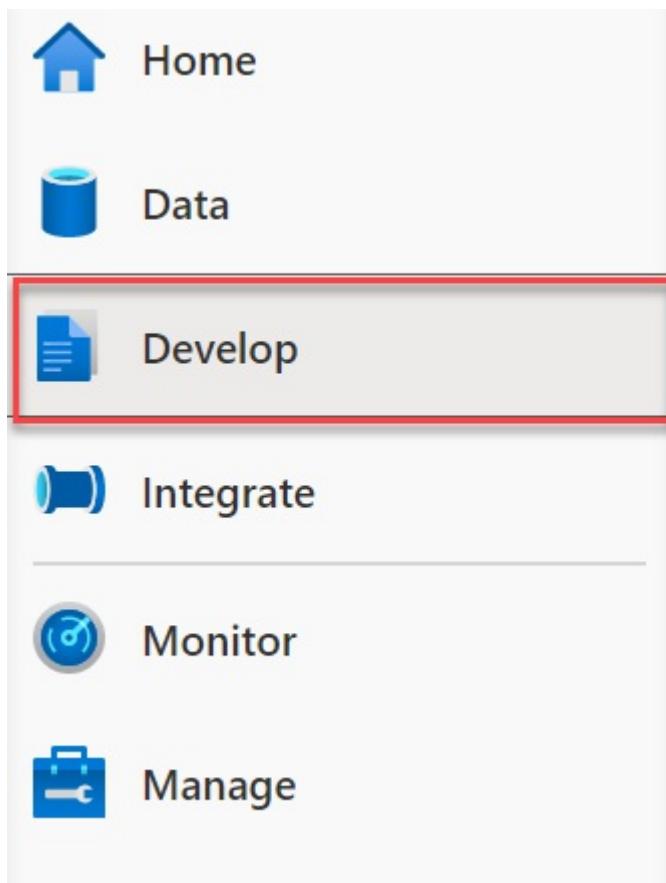
Filter resources by name

SQL scripts 23 Valid

Exercise 2: Create data pipeline to import poorly formatted CSV

Task 1: Create campaign analytics data flow

1. Navigate to the **Develop** hub.



2. Select + then **Data flow** to create a new data flow.

The screenshot shows the 'Develop' blade in the Azure Data Factory portal. On the left, there is a sidebar with the following sections:

- SQL scripts
- Notebooks
- Data flows (highlighted with a red box)
- Power BI

On the right, there is a list of items under the 'Data flows' section:

- + (highlighted with a red box)
- SQL script
- Notebook
- Data flow (highlighted with a red box)
- Spark job definition
- Power BI report
- Import

3. In the **General** settings of the **Properties** blade of the new data flow, update the **Name** to the following:
asal400_lab2_writecampaignanalyticstoasa.

Properties

General

Choose a name for your data flow.
This name can be updated at any time until it is published.

Name *

asal400_Jab2_writecampaignanalyticstoasa

Description

4. Select **Add Source** on the data flow canvas.

Validate Data flow debug

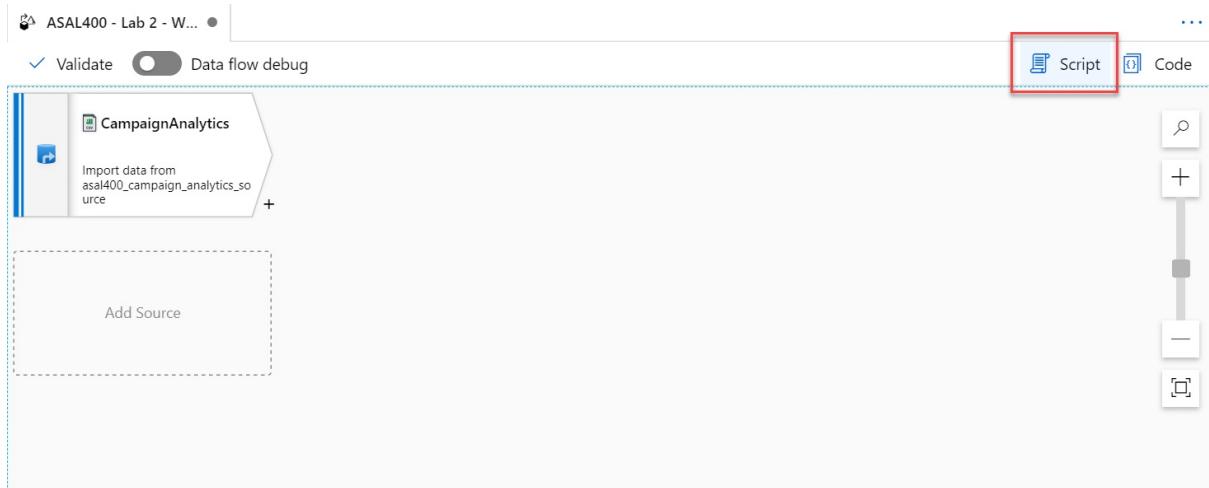


5. Under **Source settings**, configure the following:

- **Output stream name:** Enter CampaignAnalytics.
- **Source type:** Select Dataset.
- **Dataset:** Select asal400_campaign_analytics_source.
- **Options:** Select Allow schema drift and leave the other options unchecked.
- **Skip line count:** Enter 1. This allows us to skip the header row which has two fewer columns than the rest of the rows in the CSV file, truncating the last two data columns.
- **Sampling:** Select Disable.

Source settings	Source options	Projection	Optimize	Inspect	Data preview
Output stream name *	CampaignAnalytics Learn more				
Source type *	Dataset				
Dataset *	asal400_campaign_analytics_source Test connection Open New				
Options	<input checked="" type="checkbox"/> Allow schema drift <small>①</small> <input type="checkbox"/> Infer drifted column types <small>①</small> <input type="checkbox"/> Validate schema <small>①</small>				
Skip line count	1				
Sampling *	<input type="radio"/> Enable <input checked="" type="radio"/> Disable <small>①</small>				

6. When you create data flows, certain features are enabled by turning on debug, such as previewing data and importing a schema (projection). Due to the amount of time it takes to enable this option, as well as environmental constraints of the lab environment, we will bypass these features. The data source has a schema we need to set. To do this, select **Script** above the design canvas.



7. Replace the script with the following to provide the column mappings (output), then select **OK**:

```
source(output(
    {_col0_} as string,
    {_col1_} as string,
    {_col2_} as string,
    {_col3_} as string,
    {_col4_} as string,
    {_col5_} as double,
    {_col6_} as string,
    {_col7_} as double,
    {_col8_} as string,
    {_col9_} as string
),
allowSchemaDrift: true,
validateSchema: false,
skipLines: 1) ~> CampaignAnalytics
```

Your script should match the following:

Data flow name

ASAL400 - Lab 2 - Write Campaign Analytic

```
1 source(output(
2   {_col0_} as string,
3   {_col1_} as string,
4   {_col2_} as string,
5   {_col3_} as string,
6   {_col4_} as string,
7   {_col5_} as double,
8   {_col6_} as string,
9   {_col7_} as double,
10  {_col8_} as string,
11  {_col9_} as string
12 ),
13 allowSchemaDrift: true,
14 validateSchema: false,
15 skipLines: 1) ~> CampaignAnalytics
```

OK

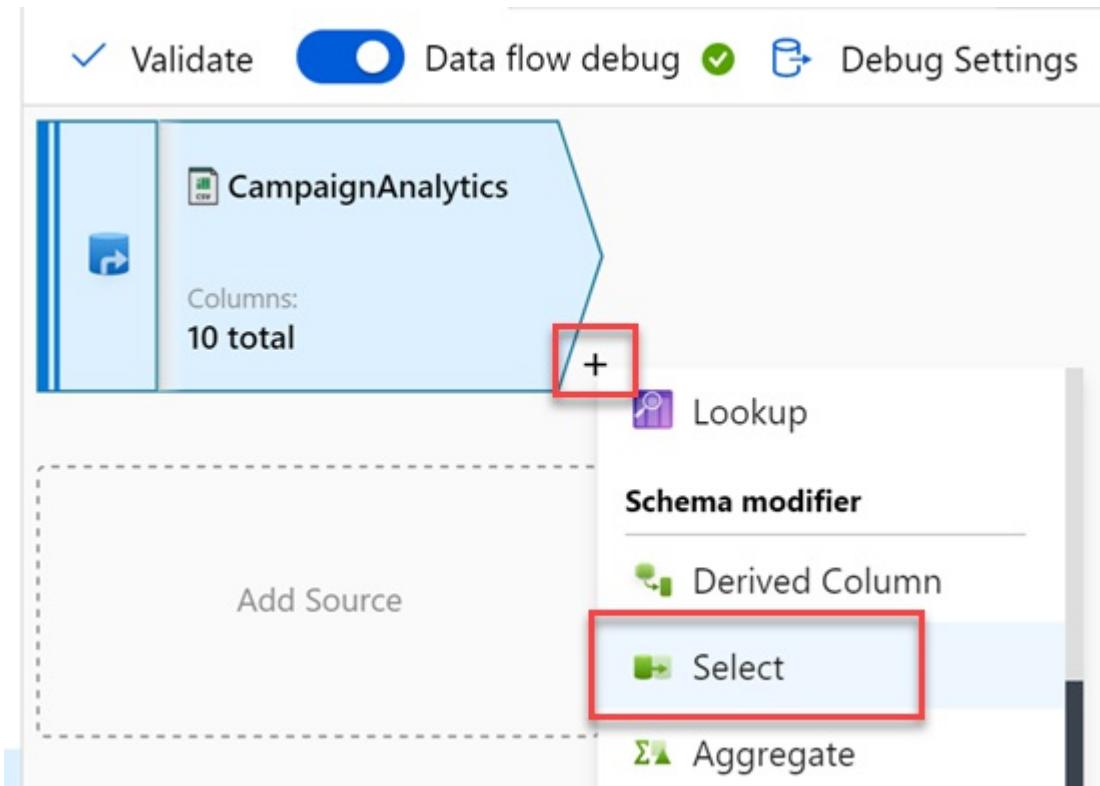
Copy as single line

Cancel

8. Select the **CampaignAnalytics** data source, then select **Projection**. The projection should display the following schema:

Source settings	Source options	Projection	Optimize	Inspect	Data preview
Define default format	Detect data type	Import projection	Reset schema		
Column name	Type		Format		
col0	abc string	▼	Specify format	▼	
col1	abc string	▼	Specify format	▼	
col2	abc string	▼	Specify format	▼	
col3	abc string	▼	Specify format	▼	
col4	abc string	▼	Specify format	▼	
col5	1.2 double	▼	Specify format	▼	
col6	abc string	▼	Specify format	▼	
col7	1.2 double	▼	Specify format	▼	
col8	abc string	▼	Specify format	▼	
col9	abc string	▼	Specify format	▼	

9. Select the + to the right of the CampaignAnalytics source, then select the **Select** schema modifier from the context menu.



10. Under **Select settings**, configure the following:

- **Output stream name:** Enter `MapCampaignAnalytics`.
- **Incoming stream:** Select `CampaignAnalytics`.
- **Options:** Check both options.
- **Input columns:** make sure **Auto mapping** is unchecked, then provide the following values in the **Name as** fields:
 - Region
 - Country
 - ProductCategory
 - CampaignName
 - RevenuePart1
 - Revenue
 - RevenueTargetPart1
 - RevenueTarget
 - City
 - State

Select settings Optimize Inspect Data preview ●

Output stream name * MapCampaignAnalytics [Learn more](#)

Incoming stream * CampaignAnalytics

Options

- Skip duplicate input columns (1)
- Skip duplicate output columns (0)

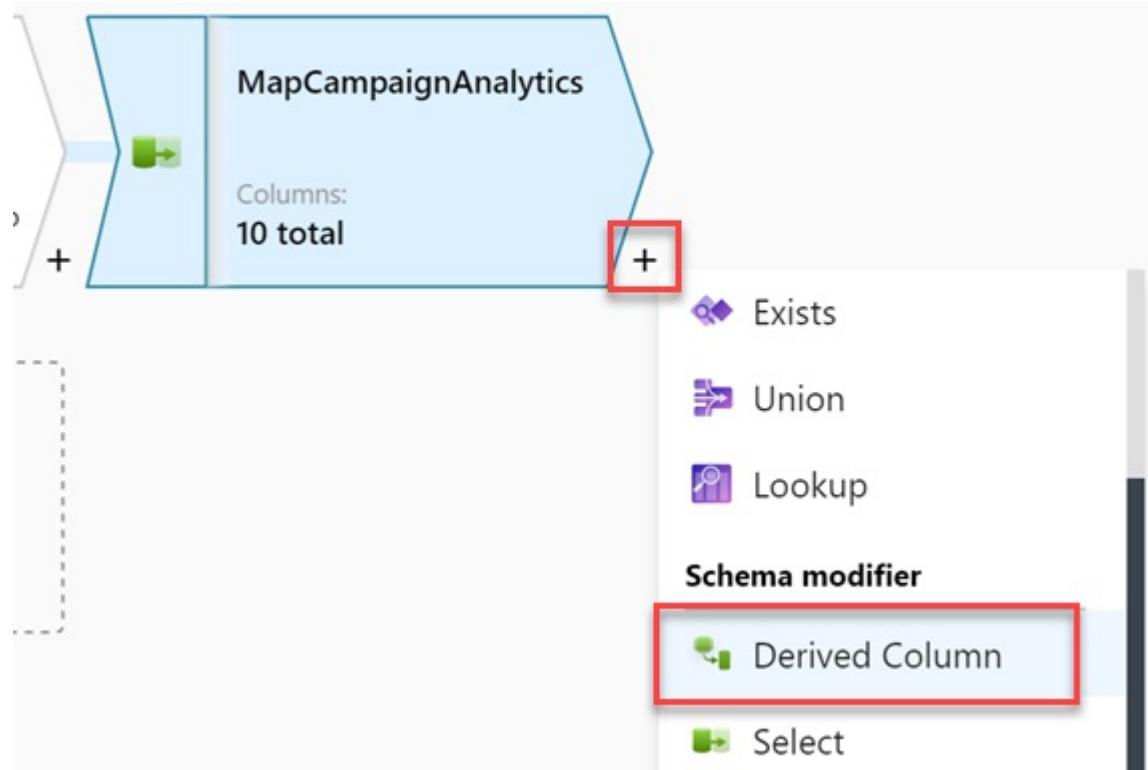
Input columns *

Auto mapping (1) ↻ Reset + Add mapping Delete

10 mappings: All in

CampaignAnalytics's column	Name as
abc_col0_	Region
abc_col1_	Country
abc_col2_	ProductCategory
abc_col3_	CampaignName
abc_col4_	RevenuePart1
1.2_col5_	Revenue
abc_col6_	RevenueTargetPart1
1.2_col7_	RevenueTarget
abc_col8_	City
abc_col9_	State

11. Select the + to the right of the MapCampaignAnalytics source, then select the **Derived Column** schema modifier from the context menu.



12. Under **Derived column's settings**, configure the following:

- **Output stream name:** Enter ConvertColumnTypesAndValues.
- **Incoming stream:** Select MapCampaignAnalytics.
- **Columns:** Provide the following information:

Column	Expression	Description
Revenue	toDecimal(replace(concat(toString(RevenuePart1), toString(Revenue)), '\\\\', ''), 10, 2, '\$##,##.##')	Concatenate the RevenuePart1 and Revenue fields, replace the invalid \ character, then convert and format the data to a decimal type.

Concatenate the
toDecimal(replace(concat(toString(RevenueTargetPart1), RevenueTargetPart1 and
RevenueTargettoString(RevenueTarget)), '\\', ''), 10, 2,
'\$###,###.##')

Derived column's settings [Optimize](#) [Inspect](#) [Data preview](#)

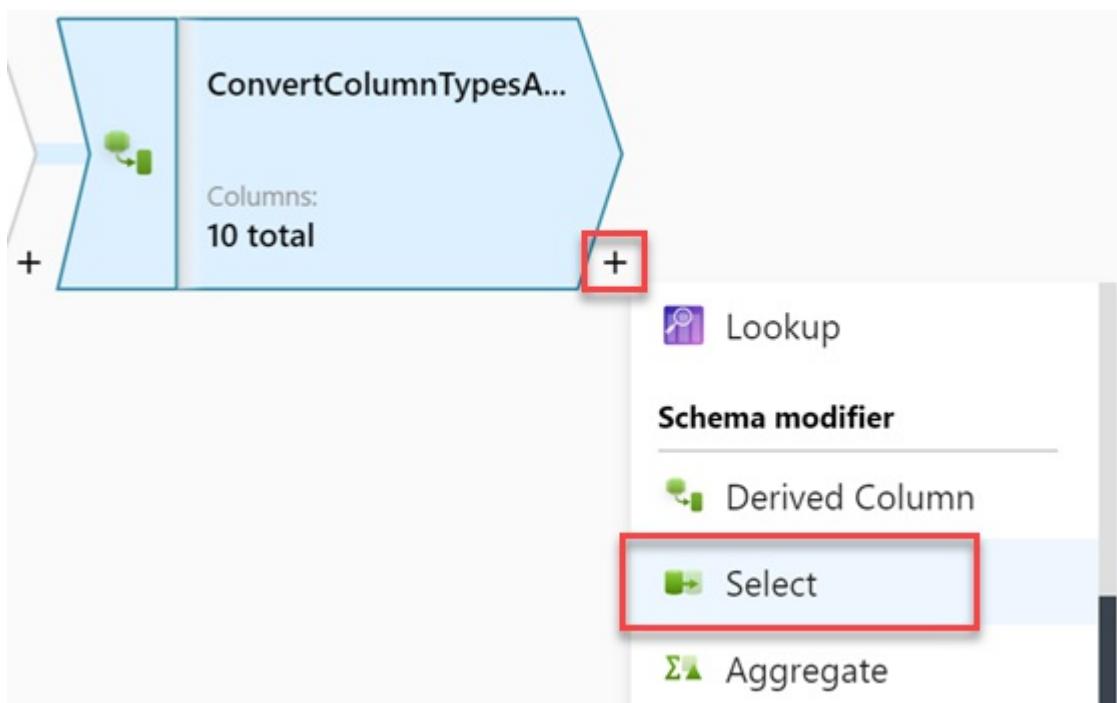
Output stream name * ConvertColumnTypeAndValues [Learn more](#)

Incoming stream * MapCampaignAnalytics

Columns * ⓘ [Add](#) [Duplicate](#) [Delete](#)

Column	Expression
Revenue	toDecimal(replace(concat(toString(RevenuePart1), t... e ^x
RevenueTarget	toDecimal(replace(concat(toString(RevenueTargetP... e ^x

13. Select the + to the right of the ConvertColumnTypeAndValues step, then select the **Select** schema modifier from the context menu.



14. Under **Select settings**, configure the following:

- **Output stream name:** Enter SelectCampaignAnalyticsColumns.
- **Incoming stream:** Select ConvertColumnTypeAndValues.
- **Options:** Check both options.
- **Input columns:** make sure Auto mapping is unchecked, then **Delete** RevenuePart1 and RevenueTargetPart1. We no longer need these fields.

Select settings Optimize Inspect Data preview ●

Output stream name * Learn more [🔗](#)

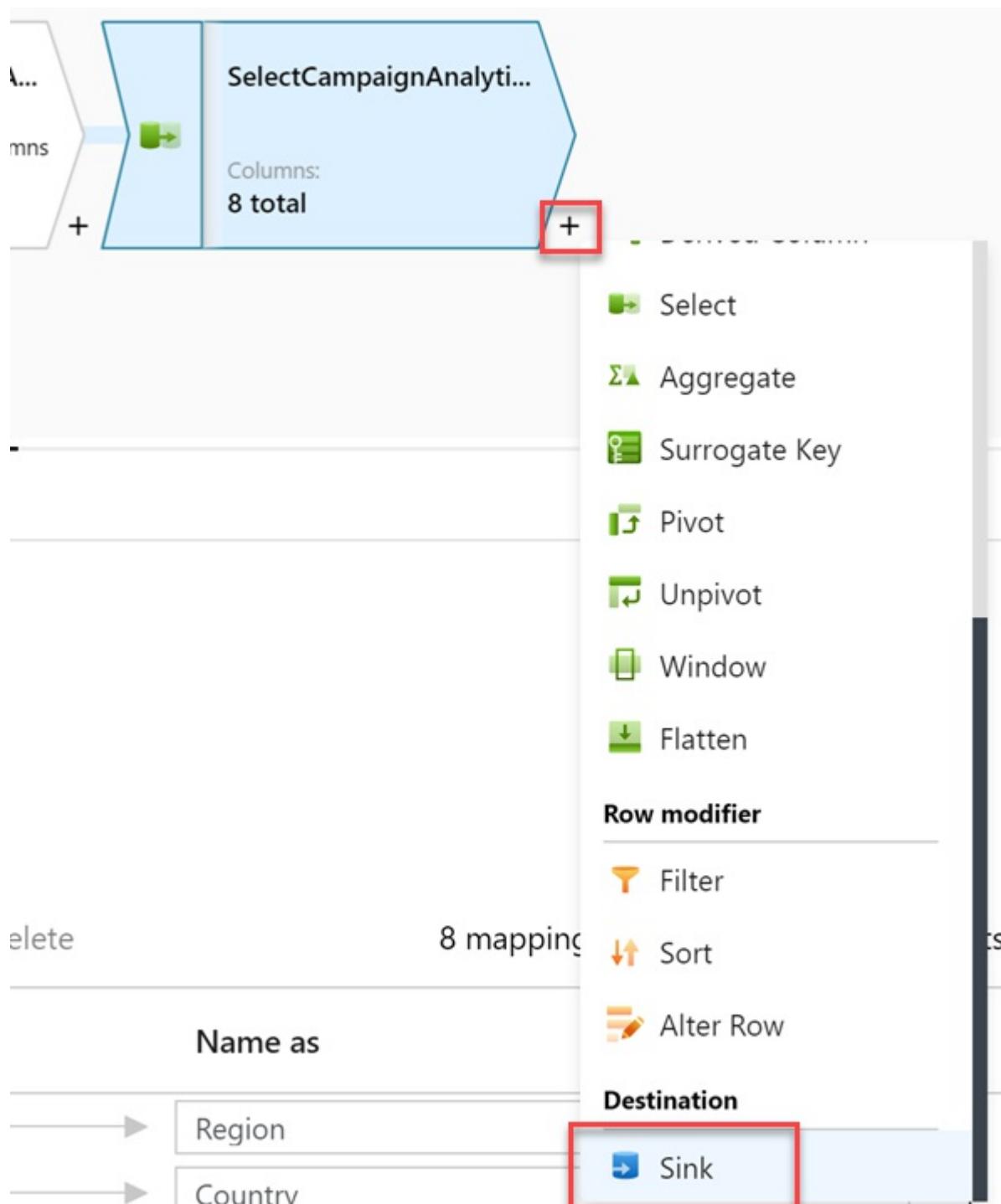
Incoming stream *

Options Skip duplicate input columns [ⓘ](#)
 Skip duplicate output columns [ⓘ](#)

Input columns * 8 mappings 2 column(s) from the inputs left unmapped

ConvertColumnTypesAndValues's column	Name as
abc Region	Region
abc Country	Country
abc ProductCategory	ProductCategory
abc CampaignName	CampaignName
e ^x Revenue	Revenue
e ^x RevenueTarget	RevenueTarget
abc City	City
abc State	State

15. Select the **+** to the right of the `SelectCampaignAnalyticsColumns` step, then select the **Sink** destination from the context menu.



16. Under **Sink**, configure the following:

- **Output stream name:** Enter CampaignAnalyticsASA.
- **Incoming stream:** Select SelectCampaignAnalyticsColumns.
- **Sink type:** Select Dataset.
- **Dataset:** Select asal400_wwi_campaign_analytics_asa, which is the CampaignAnalytics SQL table.
- **Options:** Check Allow schema drift and uncheck Validate schema.

Sink Settings Mapping Optimize Inspect Data preview ●

Output stream name * CampaignAnalyticsASA [Learn more](#)

Incoming stream * SelectCampaignAnalyticsColumns

Dataset * asal400_wwi_campaign_analytics_asa [Test connection](#)

Options

Allow schema drift [ⓘ](#)

Validate schema [ⓘ](#)

17. Select **Settings**, then configure the following:

- **Update method:** Check `Allow insert` and leave the rest unchecked.
- **Table action:** Select `Truncate table`.
- **Enable staging:** Uncheck this option. The sample CSV file is small, making the staging option unnecessary.

Sink Settings Mapping Optimize Inspect Data preview ●

i We recommend enabling staging to improve performance with Azure Synapse Analytics datasets.

Update method

Allow insert
 Allow delete
 Allow upsert
 Allow update

Table action

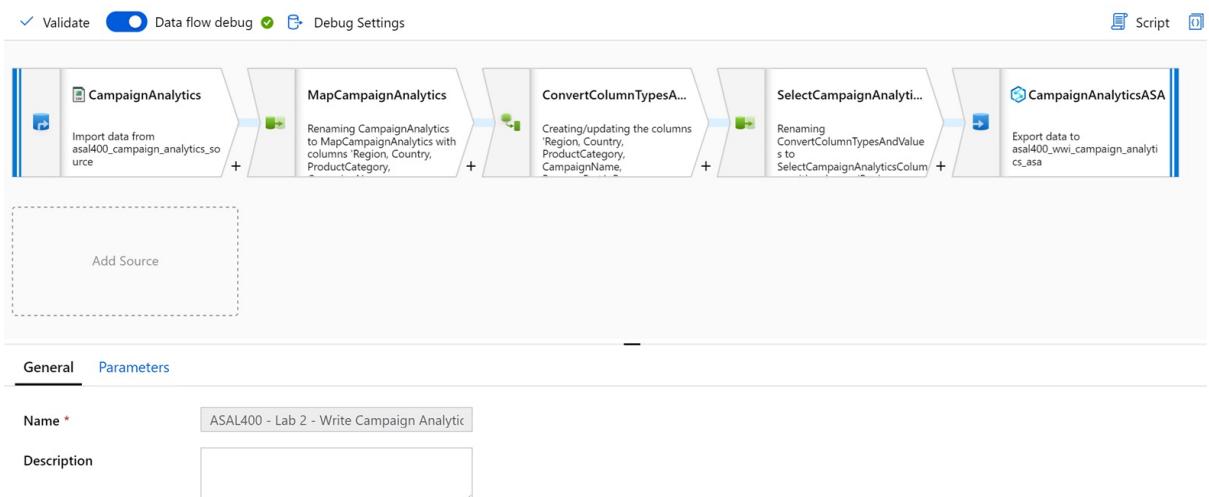
None Recreate table Truncate table

Enable staging

Batch size [ⓘ](#)

Pre SQL scripts

18. Your completed data flow should look similar to the following:



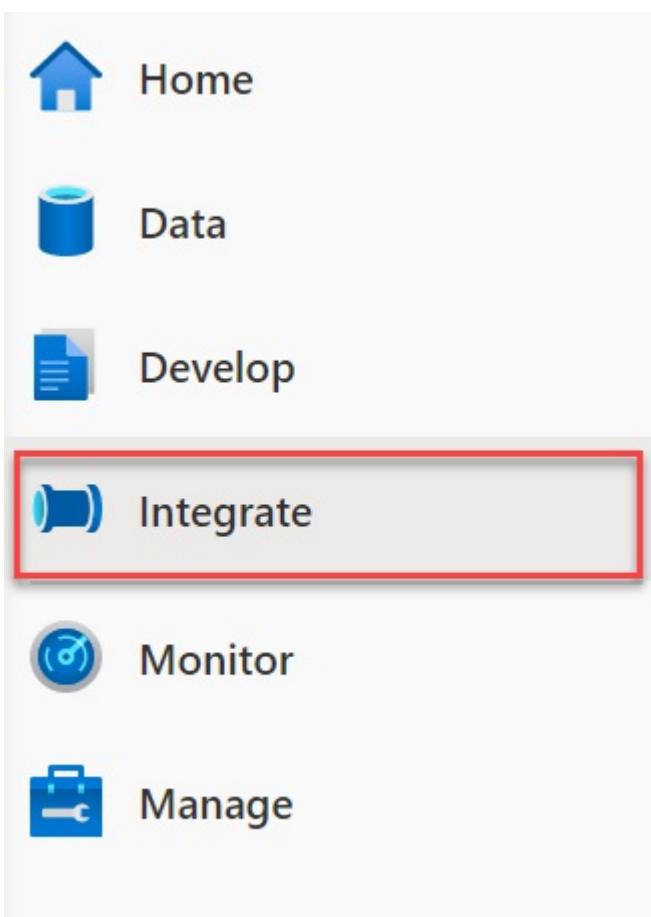
19. Select **Publish all** to save your new data flow.

The screenshot shows the 'Develop' blade in the Azure Data Factory interface. At the top, there are three buttons: 'Publish all' (highlighted with a red box), 'Validate all', and 'Refresh'. Below the buttons, the word 'Develop' is displayed with a plus sign, a downward arrow, and a double-left arrow. A search bar contains the placeholder 'Filter resources by name'. On the right, there's a sidebar with the text 'ASAL4' and a checkmark next to 'Valid'. Underneath, there's a folder named 'SQL scripts' with a count of '23'. A horizontal line separates this from the 'Task 2' section.

Task 2: Create campaign analytics data pipeline

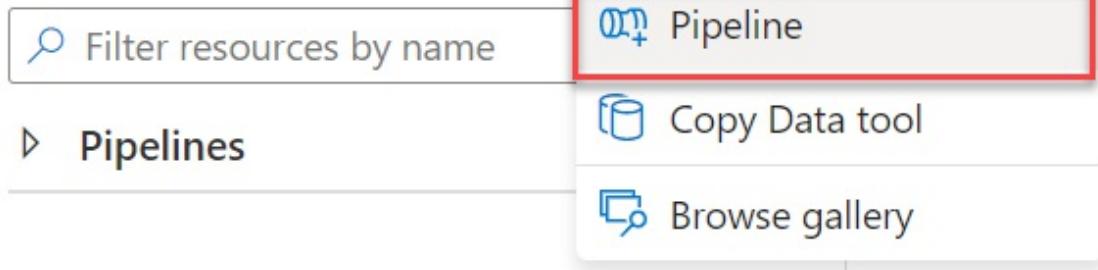
In order to run the new data flow, you need to create a new pipeline and add a data flow activity to it.

1. Navigate to the **Integrate** hub.



2. Select + then **Pipeline** to create a new pipeline.

Integrate



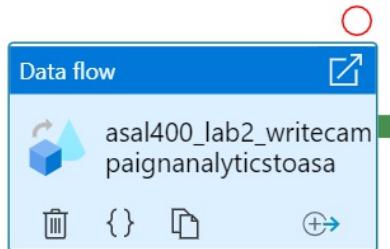
3. In the **General** section of the **Properties** blade for the new pipeline, enter the following **Name**: ASAL400 - Lab 2 - Write Campaign Analytics to ASA.
4. Expand **Move & transform** within the Activities list, then drag the **Data flow** activity onto the pipeline canvas.

The screenshot shows the 'ASAL400 - Lab 2 - W...' pipeline activities list. At the top, there's a search bar labeled 'Search activities'. Below it, there are sections for 'Synapse', 'Move & transform', 'Azure Data Explorer', 'Azure Function', and 'Batch Service'. The 'Move & transform' section is expanded, showing the 'Data flow' activity. A red arrow points from this 'Data flow' activity in the list to the pipeline canvas below. The pipeline canvas is currently empty.

5. In the Adding data flow blade, select **Use existing data flow**, then select the `asal400_lab2_writecampaignanalyticstoasa` existing data flow you created in the previous task.

The screenshot shows the 'Adding data flow' blade. It has two radio button options: 'Use existing data flow' (selected) and 'Create new data flow'. Below these options is a dropdown menu labeled 'Existing data flow *' containing the value 'asal400_lab2_writecampaignanalyticstoasa'.

6. Select **Finish**.
7. Select the mapping data flow activity on the canvas. Select the **Settings** tab, then ensure `AutoResolveIntegrationRuntime` is selected for **Run on (Azure IR)**. Choose the `Compute Optimized` **Compute type** and select `32 (+ 16 cores)` for the **Core count**.



General **Settings** Parameters User properties

Data flow * ▼ Open New

Run on (Azure IR) * (i) ▼

Compute type * ▼
Add dynamic content [Alt+P]

Core count * ▼

Logging level * (i) Verbose Basic None

▷ Sink properties
▷ staging (i)

8. Select **Publish all** to save your new pipeline.

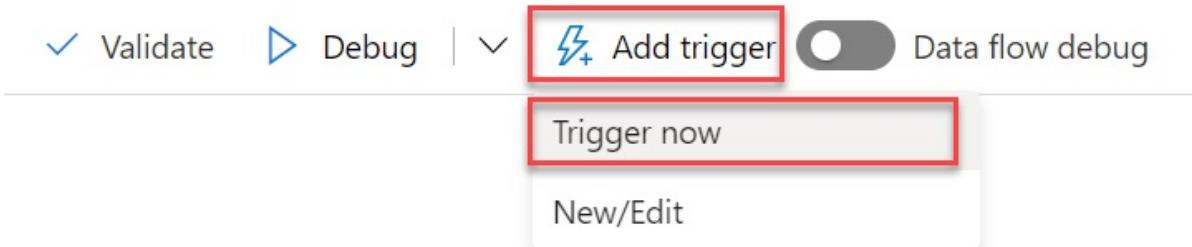
Publish all 1 Validate all Refresh

Develop ASAL4 Valid

Filter resources by name SQL scripts 23

Task 3: Run the campaign analytics data pipeline

1. Select **Add trigger**, and then select **Trigger now** in the toolbar at the top of the pipeline canvas.



2. In the Pipeline run blade, select **OK** to start the pipeline run.

Pipeline run

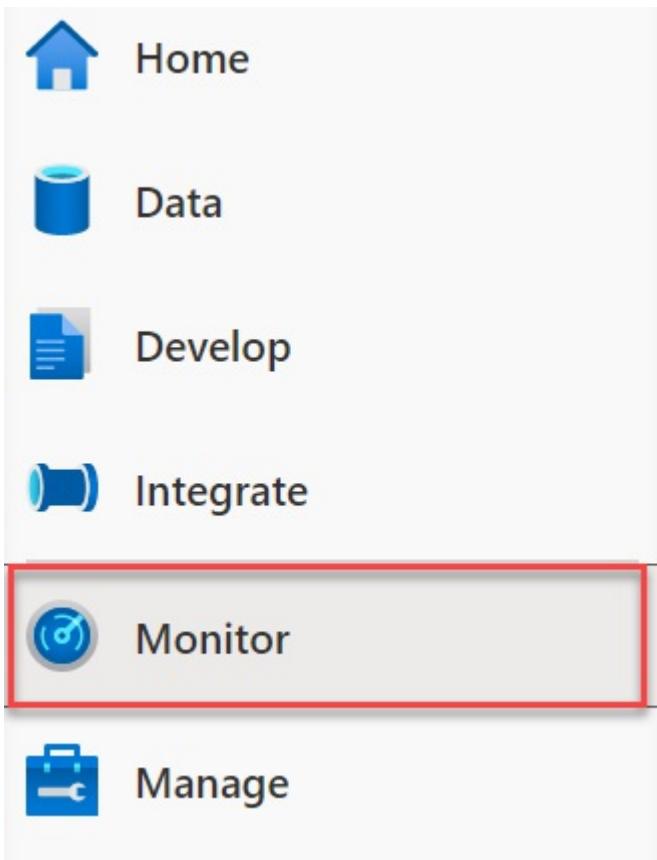
⚠ Trigger pipeline now using last published configuration.

Parameters

NAME	TYPE	VALUE
No records found		

OK **Cancel**

3. Navigate to the **Monitor** hub.



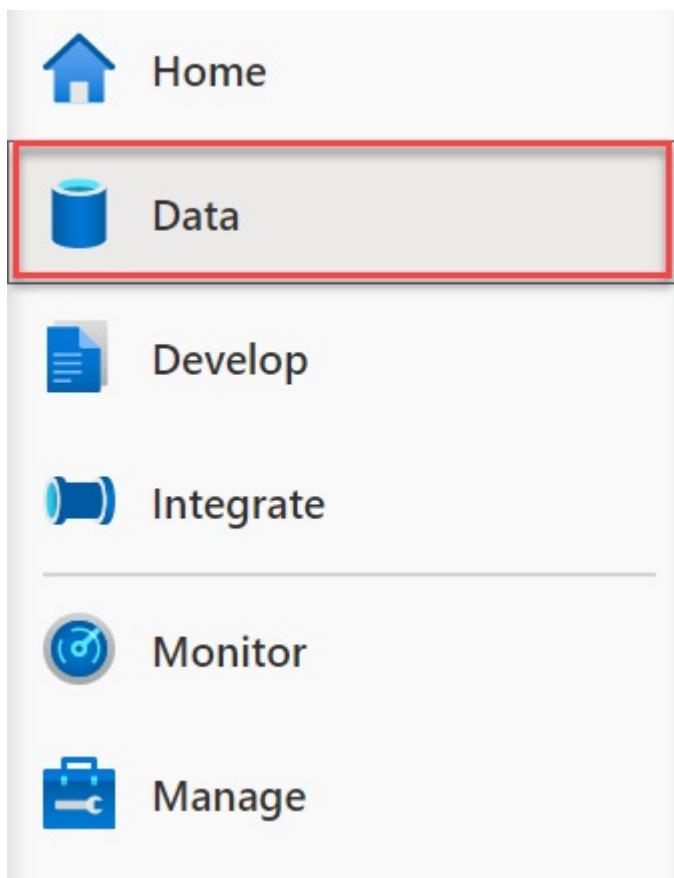
4. Wait for the pipeline run to successfully complete. You may need to refresh the view.

5. **Important:** if the pipeline run fails with Internal Server Error:Failed to submit job on job cluster. Integration Runtime or takes longer than **5 minutes** to complete, you are likely experiencing capacity-related issues. If one of these cases is true, skip ahead to **Task 4b (fallback)** to see a successful outcome.

Task 4: View campaign analytics table contents

Now that the pipeline run is complete, let's take a look at the SQL table to verify the data successfully copied.

1. Navigate to the **Data** hub.



2. Expand the `SqlPool01` database underneath the **Workspace** section, then expand **Tables**.
3. Right-click the `wwi.CampaignAnalytics` table, then select the **Select TOP 100 rows** menu item under the New SQL script context menu. You may need to refresh to see the new tables.

4. The properly transformed data should appear in the query results.

```

1  SELECT TOP (1000) [Region]
2  ,[Country]
3  ,[ProductCategory]
4  ,[CampaignName]
5  ,[Revenue]
6  ,[RevenueTarget]
7  ,[city]
8  ,[State]
9  | FROM [wwi].[CampaignAnalytics]

```

REGION	COUNTRY	PRODUCTCATEGORY	CAMPAIGNNAME	REVENUE	REVENUETARGET	CITY	STATE
South America	Brazil	Décor	Tailored for You	9229.00	16528.00	NULL	NULL
Europe	Italy	Electronics	Enjoy the Moment	6815.00	14606.00	NULL	NULL
Asia Pacific	India	Lighting	Get Sporty	920.00	13475.00	NULL	NULL
North & Central America	USA	Apparel and Footwe...	Enjoy the Moment	13554.00	19215.00	NULL	NULL
South East	US	Team Sports	Enjoy the Moment	9702.00	7876.00	Miami	Florida
Far West	US	Books	EnjoyTheMoment...	9963.00	18377.00	San Diego	California
South America	Brazil	Décor	Enjoy the Moment	13945.00	13624.00	NULL	NULL
Europe	France	Furniture	Get Sporty	13124.00	14955.00	NULL	NULL
... 10 more rows ...							

5. Update the query to the following and Run:

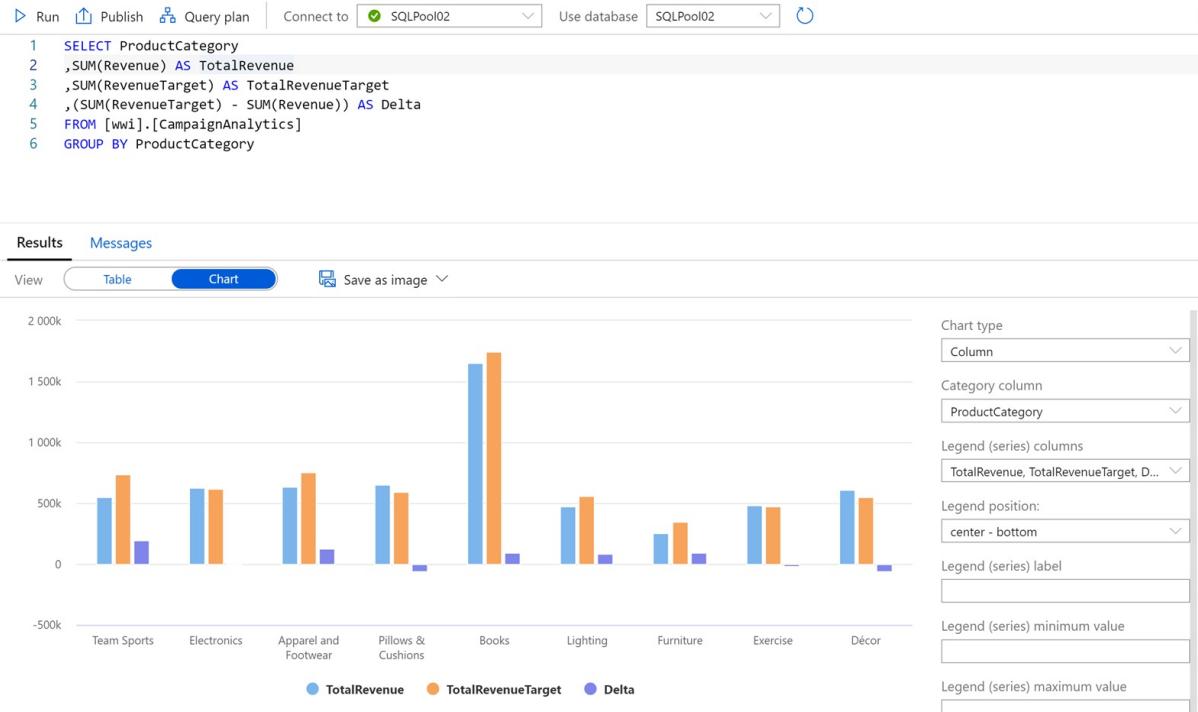
```

SELECT ProductCategory
, SUM(Revenue) AS TotalRevenue
, SUM(RevenueTarget) AS TotalRevenueTarget
, (SUM(RevenueTarget) - SUM(Revenue)) AS Delta
FROM [wwi].[CampaignAnalytics]
GROUP BY ProductCategory

```

6. In the query results, select the **Chart** view. Configure the columns as defined:

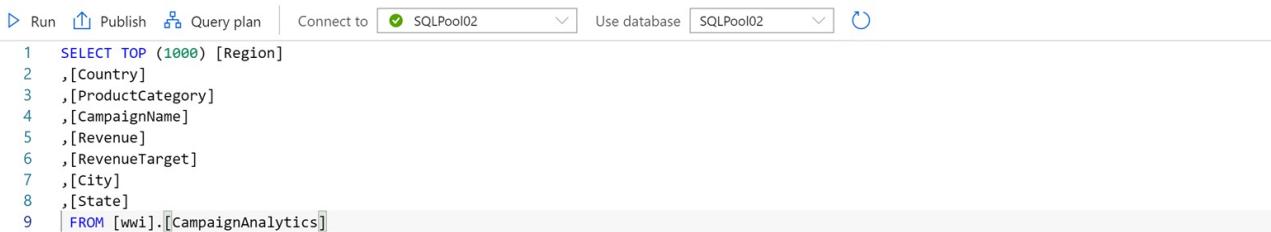
- **Chart type:** Select Column.
- **Category column:** Select ProductCategory.
- **Legend (series) columns:** Select TotalRevenue, TotalRevenueTarget, and Delta.



Task 4b (fallback): View campaign analytics table contents

Read this task if the pipeline run failed due to capacity-related issues.

The pipeline truncates the `wwi.CampaignAnalytics` table and inserts the cleaned up campaign analytics data from the improperly formatted CSV. When we query the table to view its contents, it looks like the following:



The screenshot shows a SQL query results page with a table view. The table displays campaign analytics data with columns: REGION, COUNTRY, PRODUCTCATEGORY, CAMPAIGNNAME, REVENUE, REVENUETARGET, CITY, and STATE. The data includes entries for South America, Europe, Asia Pacific, North & Central America, South East, Far West, and various countries like Brazil, Italy, India, USA, etc., with their respective campaign names, revenues, targets, cities, and states.

REGION	COUNTRY	PRODUCTCATEGORY	CAMPAIGNNAME	REVENUE	REVENUETARGET	CITY	STATE
South America	Brazil	Décor	Tailored for You	9229.00	16528.00	NULL	NULL
Europe	Italy	Electronics	Enjoy the Moment	6815.00	14606.00	NULL	NULL
Asia Pacific	India	Lighting	Get Sporty	920.00	13475.00	NULL	NULL
North & Central America	USA	Apparel and Footwe...	Enjoy the Moment	13554.00	19215.00	NULL	NULL
South East	US	Team Sports	Enjoy the Moment	9702.00	7876.00	Miami	Florida
Far West	US	Books	EnjoyTheMoment...	9963.00	18377.00	San Diego	California
South America	Brazil	Décor	Enjoy the Moment	13945.00	13624.00	NULL	NULL
Europe	France	Furniture	Get Sporty	13124.00	14955.00	NULL	NULL

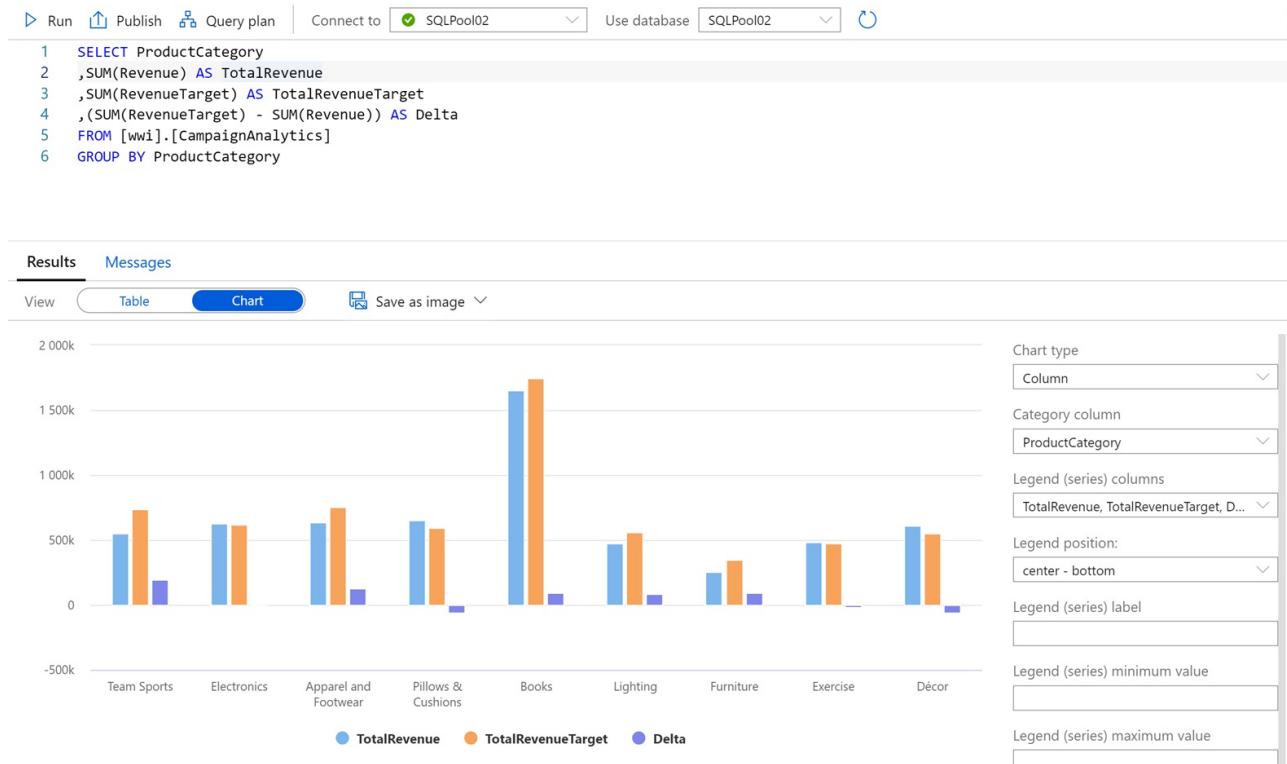
If we wish to view the total revenue compared to the target revenue of each product category, we can update the query as follows:

```

SELECT ProductCategory
, SUM(Revenue) AS TotalRevenue
, SUM(RevenueTarget) AS TotalRevenueTarget
, (SUM(RevenueTarget) - SUM(Revenue)) AS Delta
FROM [wwi].[CampaignAnalytics]
GROUP BY ProductCategory

```

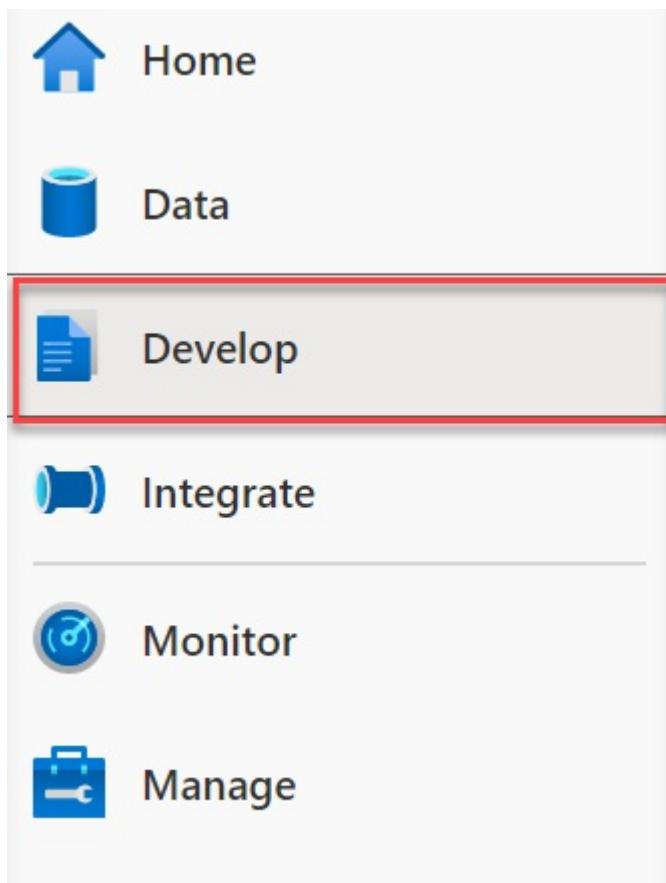
The query results output includes the standard Table view, as well as a Chart view. If we switch to the column chart view and set the category column to ProductCategory, we see the following:



Exercise 3: Create data pipeline to join disparate data sources

Task 1: Create user profile data flow

1. Navigate to the **Develop** hub.



2. Select + then **Data flow** to create a new data flow.

The screenshot shows the 'Develop' workspace. On the left is a sidebar with the following options:

- SQL scripts
- Notebooks
- Data flows (highlighted with a red box)
- Power BI

On the right is a main area with a '+' button (highlighted with a red box) and a list of resource types:

- SQL script
- Notebook
- Data flow (highlighted with a red box)
- Spark job definition
- Power BI report
- Import

3. In the **General** section of the **Profiles** pane of the new data flow, update the **Name** to the following:
asal400_lab2_writeuserprofiledatatoasa.
4. Select **Add Source** on the data flow canvas.

Validate Data flow debug



5. Under **Source settings**, configure the following:

- **Output stream name:** Enter EcommerceUserProfiles.
- **Source type:** Select Dataset.
- **Dataset:** Select asal400_ecommerce_userprofiles_source.

The screenshot shows the 'Source settings' tab selected in a pipeline configuration interface. The tab bar includes 'Source settings' (selected), 'Source options', 'Projection', 'Optimize', 'Inspect', and 'Data preview'. The 'Source settings' section contains the following fields:

- Output stream name ***: EcommerceUserProfiles [Learn more](#)
- Source type ***: Dataset [Test connection](#)
- Dataset ***: asal400_ecommerce_userprofiles_so... [Test connection](#)
- Options**:
 - Allow schema drift [i](#)
 - Infer drifted column types [i](#)
 - Validate schema [i](#)
- Sampling ***: Enable Disable [i](#)

6. Select the **Source options** tab, then configure the following:

- **Wildcard paths:** Enter online-user-profiles-02/*.json.
- **JSON Settings:** Expand this section, then select the **Single document** setting. This denotes that each file contains a single JSON document.

Source settings Source options **Projection** Optimize Inspect Data preview

Wildcard paths + Delete

Partition root path

List of files

Column to store file name

After completion * No action Delete source files Move

Start time (UTC) End time (UTC)

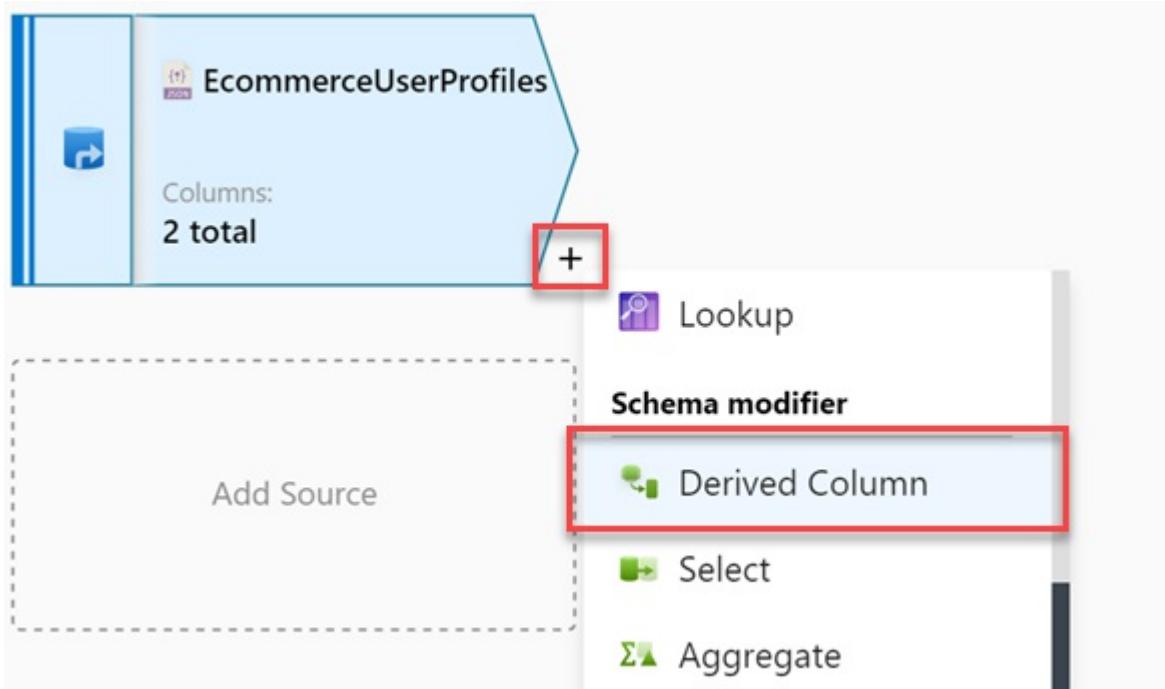
Filter by last modified

▲ JSON settings

Single document ①

Unquoted column names ①

7. Select the + to the right of the EcommerceUserProfiles source, then select the **Derived Column** schema modifier from the context menu.



8. Under **Derived column's settings**, configure the following:

- **Output stream name:** Enter `userId`.
- **Incoming stream:** Select `EcommerceUserProfiles`.
- **Columns:** Provide the following information:

Column Expression	Description
<code>visitorId toInteger(visitorId)</code>	Converts the <code>visitorId</code> column from a string to an integer.

Derived column's settings

Optimize Inspect Data preview ●

Output stream name * Learn more [?](#)

Incoming stream *

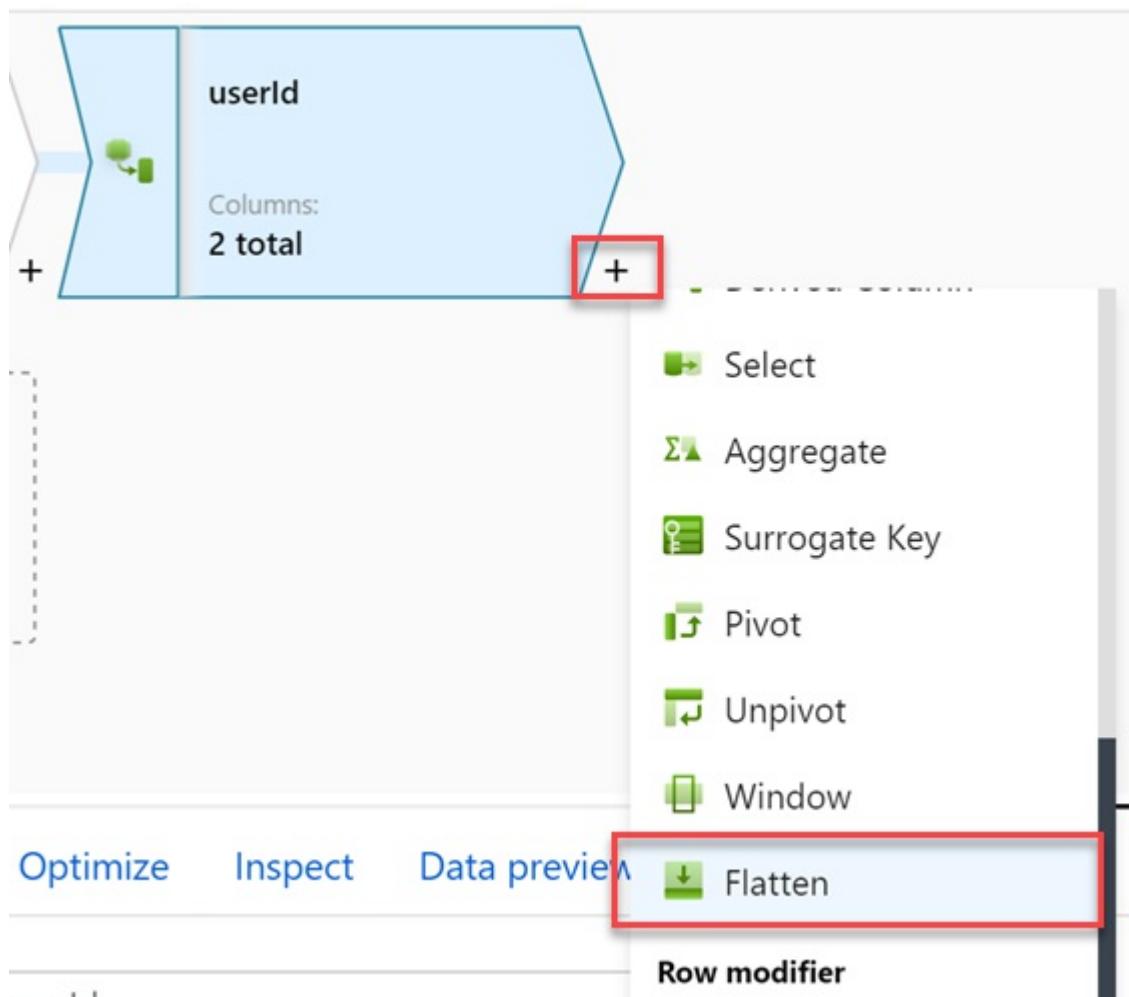
Columns * [①](#)

[+](#) Add [Duplicate](#) [Delete](#)

Column	Expression
<input type="checkbox"/> visitorId	<code>tolInteger(visitorId)</code> 123

[+](#) [D](#) [B](#)

9. Select the **+** to the right of the `userId` step, then select the **Flatten** schema modifier from the context menu.



10. Under **Flatten settings**, configure the following:

- **Output stream name:** Enter `UserTopProducts`.
- **Incoming stream:** Select `userId`.
- **Unroll by:** Select `[] topProductPurchases`.
- **Input columns:** Provide the following information:

userId's column	Name as
<code>visitorId</code>	<code>visitorId</code>
<code>topProductPurchases.productId</code>	<code>productId</code>
<code>topProductPurchases.itemsPurchasedLast12Months</code>	<code>itemsPurchasedLast12Months</code>

Flatten settings [Optimize](#) [Inspect](#) [Data preview](#) ●

Output stream name * ? Help [Learn more](#)

Incoming stream *

Unroll by *

Unroll root

Options Skip duplicate input columns Skip duplicate output columns

Input columns * [Reset](#) [Add mapping](#) [Delete](#) 3 mappings: All input:

userId's column	Name as
abc visitorId	visitorId
abc topProductPurchases.productId	productId
abc topProductPurchases.itemsPurchasedLast...	itemsPurchasedLast12Months

These settings provide a flattened view of the data source with one or more rows per `visitorId`, similar to when you explored the data within the Spark notebook in lab 1. Using data preview requires you to enable Debug mode, which we are not enabling for this lab. *The following screenshot is for illustration only:*

Flatten settings [Optimize](#) [Inspect](#) [Data preview](#) ●

Number of rows + **INSERT** 100 * **UPDATE** 0 ✘ **DELETE** 0 *+ **UPSERT** 0 🔎 **LOOKUP** 0

↻ Refresh Typecast 📈 Modify 🗺 Map drifted 📈 Statistics ✖ Remove

visitorId	productId	itemsPurchasedLast12Months
9611082	61	62
9611082	3003	20
9611082	2140	39
9611082	2918	62
9611082	4482	85
9611082	140	73
9611082	3892	33

- Select the **+** to the right of the `UserTopProducts` step, then select the **Derived Column** schema modifier from the context menu.

The screenshot shows the Data Flow interface with a step named "UserTopProducts". The step has three columns: "visitorId", "productId", and "itemsPurchasedLast12Months". A context menu is open at the bottom right of the step, with a red box highlighting the "Derived Column" option under the "Schema modifier" section.

UserTopProducts
Columns: 3 total

+ **Derived Column**

Lookup Select

- Under **Derived column's settings**, configure the following:

- Output stream name:** Enter `DeriveProductColumns`.

- **Incoming stream:** Select UserTopProducts.

- **Columns:** Provide the following information:

Column	Expression	Description
productId	toInteger(productId)	Converts the productId column from a string to an integer.
itemsPurchasedLast12Months	toInteger(itemsPurchasedLast12Months)	Converts the itemsPurchasedLast12Months column from a string to an integer.

Derived column's settings [Optimize](#) [Inspect](#) [Data preview](#) ●

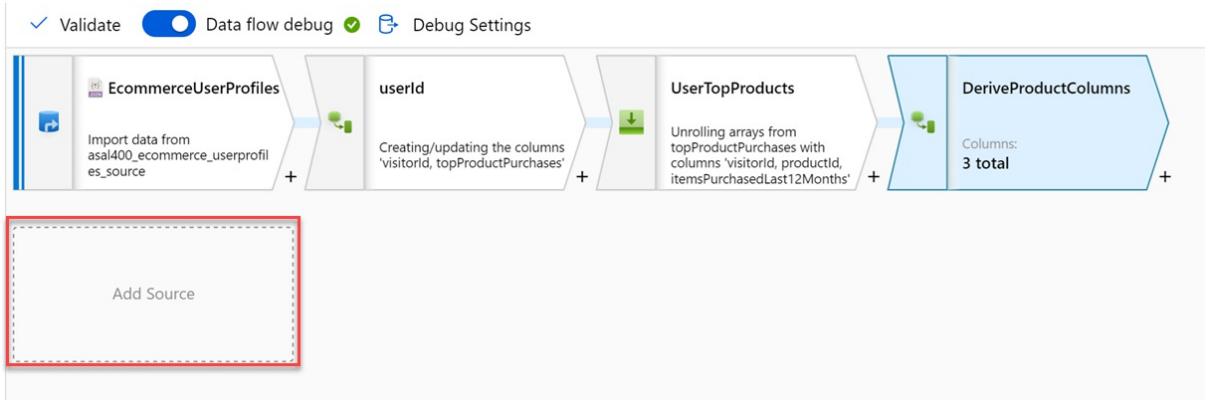
Output stream name * DeriveProductColumns [Learn more](#)

Incoming stream * UserTopProducts

Columns * ⓘ [Add](#) [Duplicate](#) [Delete](#)

Column	Expression
productId	toInteger(productId)
itemsPurchasedLast12Months	toInteger(itemsPurchasedLast12Months)

13. Select **Add Source** on the data flow canvas beneath the EcommerceUserProfiles source.



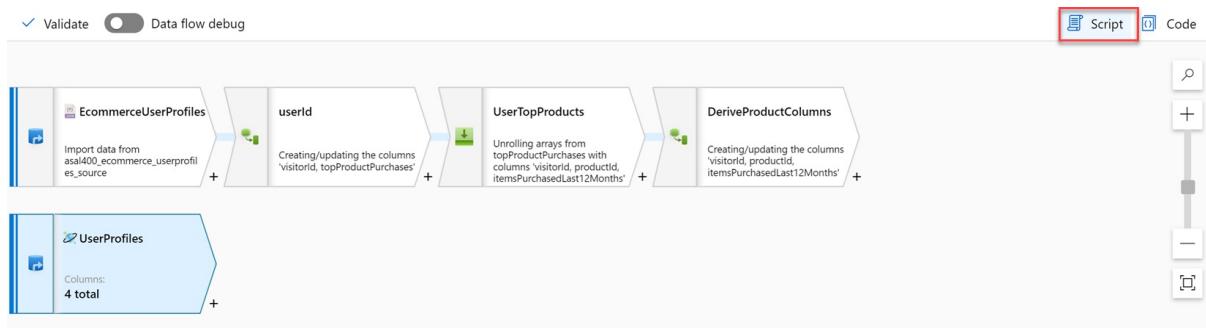
14. Under **Source settings**, configure the following:

- **Output stream name:** Enter UserProfiles.
- **Source type:** Select Dataset.
- **Dataset:** Select asal400_customerprofile_cosmosdb.

Source settings Source options Projection Optimize Inspect Data preview

Output stream name *	<input type="text" value="UserProfiles"/>	Learn more
Source type *	<input type="text" value="Dataset"/>	
Dataset *	<input type="text" value="asal400_customerprofile_cosmosdb"/>	Test connection
Options	<input checked="" type="checkbox"/> Allow schema drift <small> ⓘ </small> <input type="checkbox"/> Infer drifted column types <small> ⓘ </small> <input type="checkbox"/> Validate schema <small> ⓘ </small>	
Sampling *	<input type="radio"/> Enable <input checked="" type="radio"/> Disable <small> ⓘ </small>	

15. Since we are not using the data flow debugger, we need to enter the data flow's Script view to update the source projection.
Select **Script** in the toolbar above the canvas.



16. Locate the **UserProfiles** source in the script and replace its script block with the following to set `preferredProducts` as an `integer[]` array and ensure the data types within the `productReviews` array are correctly defined:

```
source(output(
    cartId as string,
    preferredProducts as integer[],
    productReviews as (productId as integer, reviewDate as string, reviewText as string)[],
    userId as integer
),
allowSchemaDrift: true,
validateSchema: false,
format: 'document') ~> UserProfiles
```

```

1 source(output(
2     |     visitorId as string,
3     |     topProductPurchases as (productId as string, itemsPurchasedLast12Months as string)[]
4     |),
5     allowSchemaDrift: true,
6     validateSchema: false,
7     singleDocument: true,
8     wildcardPaths:['online-user-profiles-02/*.json']) ~> EcommerceUserProfiles
9 source(output(
10    |     cartId as string,
11    |     preferredProducts as integer[],
12    |     productReviews as (productId as integer, reviewDate as string, reviewText as string)[],
13    |     userId as integer
14    |),
15    allowSchemaDrift: true,
16    validateSchema: false,
17    format: 'document') ~> UserProfiles
18 EcommerceUserProfiles derive(visitorId = toInteger(visitorId)) ~> userId
19 userId foldDown(unroll(topProductPurchases),
20     mapColumn(
21         |     visitorId,
22         |     productId = topProductPurchases.productId,
23         |     itemsPurchasedLast12Months = topProductPurchases.itemsPurchasedLast12Months
24     ),
25     skipDuplicateMapInputs: false,
26     skipDuplicateMapOutputs: false) ~> UserTopProducts
27 UserTopProducts derive(productId = toInteger(productId),
28     itemsPurchasedLast12Months = toInteger(itemsPurchasedLast12Months)) ~> DeriveProductColumns

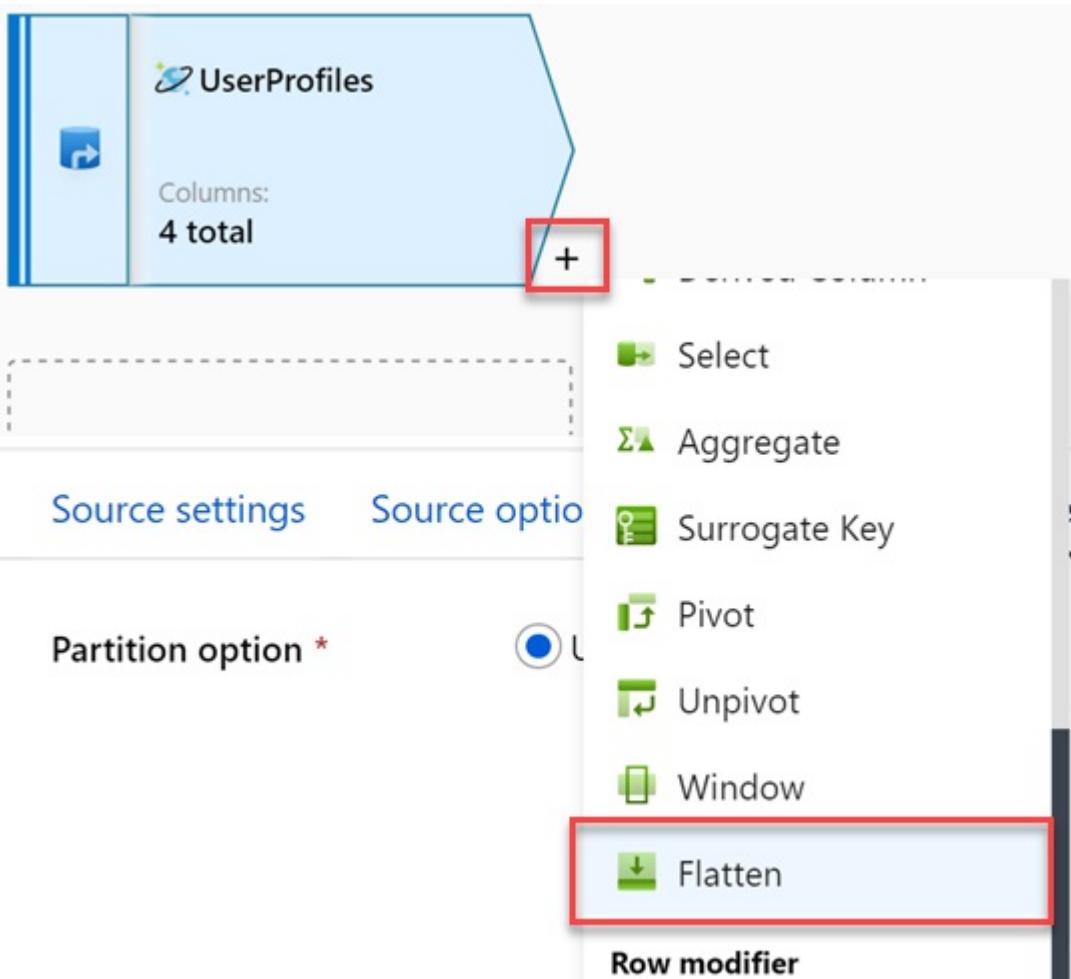
```

OK **Copy as single line** **Cancel**

17. Select **OK** to apply the script changes. The data source has now been updated with the new schema. The following screenshot shows what the source data looks like if you are able to view it with the data preview option. Using data preview requires you to enable Debug mode, which we are not enabling for this lab. *The following screenshot is for illustration only:*

Source settings		Source options		Projection		Optimize		Inspect		Data preview	
Number of rows	+ INSERT 100	*	UPDATE 0	x	DELETE 0	*	UPsert 0	q	LOOKUP 0		
↻ Refresh	[]	Typecast	▼	Modify	▼	Map drifted	▼	Statistics	x	Remove	
↑↓	cartId abc		preferredProducts []			123 preferredProducts[1]: 4444		userId 123			
+	406a06af-e54f-42e9-aad8-9a36f2c7f8ca		[...]			123 preferredProducts[2]: 330		9079954			
+	5c4dc5dc-a585-41ec-8149-9133caa3a73a		[...]					9079747			
+	d79f4b3a-6c66-497a-bf59-eca12dd4f16a		[...]					9079334			
+	acde5617-af9f-4ad9-98ed-e03c793e7bd0		[...]					9079190			
+	7f2742a6-4653-4905-80ad-2d64dae04253		[...]					9078634			
+	2d3c75f6-24ce-4307-9059-902736896c61		[...]					9078467			
+	44f4c3f9-d761-4d10-bbfb-84cd45c3a461		[...]					9078115			

18. Select the **+** to the right of the **UserProfiles** source, then select the **Flatten** schema modifier from the context menu.



19. Under **Flatten settings**, configure the following:

- **Output stream name:** Enter `UserPreferredProducts`.
- **Incoming stream:** Select `UserProfiles`.
- **Unroll by:** Select `[] preferredProducts`.
- **Input columns:** Provide the following information. Be sure to **delete** `cartId` and `[] productReviews`:

UserProfiles's column Name as

<code>userId</code>	<code>userId</code>
<input type="checkbox"/> <code>preferredProducts</code>	<code>preferredProductId</code>

Flatten settings [Optimize](#) [Inspect](#) [Data preview](#) [Help](#) [Learn more](#)

Output stream name *	<input type="text" value="UserPreferredProducts"/>	? Help Learn more												
Incoming stream *	<input type="text" value="UserProfiles"/>	▼												
Unroll by *	<input type="text" value="[] preferredProducts"/>	○												
Unroll root	<input type="text"/>	○												
Options	<input type="checkbox"/> Skip duplicate input columns ○													
	<input type="checkbox"/> Skip duplicate output columns ○													
Input columns *	Reset Add mapping Delete	2 mappings: 2 column(s) from the inputs left												
<table border="1"> <thead> <tr> <th>UserProfiles's column</th> <th>⋮</th> <th>Name as</th> <th>⋮</th> </tr> </thead> <tbody> <tr> <td><input type="checkbox"/> abc userId</td> <td>→</td> <td><input type="text" value="userId"/></td> <td>+ Delete</td> </tr> <tr> <td><input type="checkbox"/> [] preferredProducts</td> <td>→</td> <td><input type="text" value="preferredProductId"/></td> <td>+ Delete</td> </tr> </tbody> </table>			UserProfiles's column	⋮	Name as	⋮	<input type="checkbox"/> abc userId	→	<input type="text" value="userId"/>	+ Delete	<input type="checkbox"/> [] preferredProducts	→	<input type="text" value="preferredProductId"/>	+ Delete
UserProfiles's column	⋮	Name as	⋮											
<input type="checkbox"/> abc userId	→	<input type="text" value="userId"/>	+ Delete											
<input type="checkbox"/> [] preferredProducts	→	<input type="text" value="preferredProductId"/>	+ Delete											

These settings provide a flattened view of the data source with one or more rows per `userId`. Using data preview requires you to enable Debug mode, which we are not enabling for this lab. *The following screenshot is for illustration only:*

Flatten settings Optimize Inspect Data preview ●

Number of rows + INSERT 100 * UPDATE 0 X DELETE 0 * UPsert 0

Refresh Typecast Modify Map drifted Statistics Remove

↑↓	userId	preferredProductId
+	9079747	4235
+	9079747	3288
+	9079747	2756
+	9079334	4074
+	9079334	272
+	9079334	2164
+	9079334	414

20. Now it is time to join the two data sources. Select the + to the right of the **DeriveProductColumns** step, then select the **Join** option from the context menu.



21. Under **Join settings**, configure the following:

- **Output stream name:** Enter **JoinTopProductsWithPreferredProducts**.
- **Left stream:** Select **DeriveProductColumns**.
- **Right stream:** Select **UserPreferredProducts**.
- **Join type:** Select **Full outer**.
- **Join conditions:** Provide the following information:

Left: DeriveProductColumns's column Right: UserPreferredProducts's column

visitorId userId

Join settings Optimize Inspect Data preview ●

Output stream name * Learn more [?](#)

Left stream *

Right stream *

Join type * Full outer Inner Left outer Right outer Custom (cross)

Join conditions * **Left: DeriveProductColumns's column** **Right: UserPreferredProducts's column** =

22. Select **Optimize** and configure the following:

- **Broadcast:** Select **Fixed**.
- **Broadcast options:** Check **Left: 'DeriveProductColumns'**.

- **Partition option:** Select Set partitioning.
- **Partition type:** Select Hash.
- **Number of partitions:** Enter 30.
- **Column:** Select productId.

Join settings Optimize **Inspect** Data preview

Broadcast Auto ⓘ Fixed ⓘ Off ⓘ

Broadcast options *

Left: 'DeriveProductColumns' ⓘ

Right: 'UserPreferredProducts' ⓘ

Partition option *

Use current partitioning Single partition Set Partitioning

Partition type *

Round Robin Hash Dynamic Range Fixed Range Key

Number of partitions * 30

Column values to hash on * **Column**

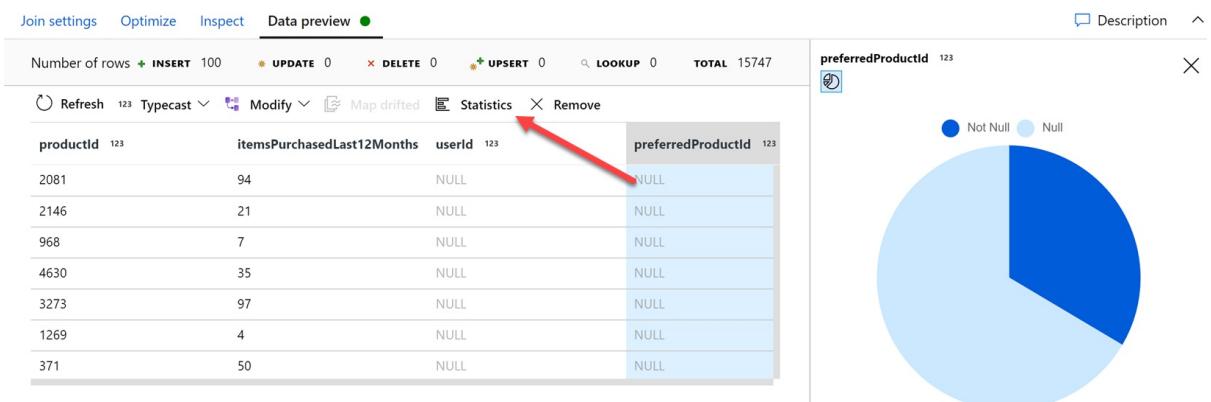
123 productId +

23. Select the **Inspect** tab to see the join mapping, including the column feed source and whether the column is used in a join.

Join settings Optimize **Inspect** Data preview

Number of columns Left: DeriveProductColumns 3		Right: UserPreferredProducts 2	
Order ↑↓	Column ↑↓	Type ↑↓	Fed by ↑↓
1	visitorId	123 integer	UserTopProducts ✓
2	productId	123 integer	DeriveProductColumns
3	itemsPurchasedLast12Months	123 integer	DeriveProductColumns
4	userId	123 integer	UserPreferredProducts ✓
5	preferredProductId	123 integer	UserPreferredProducts

For illustrative purposes of data preview only: Since we are not turning on data flow debugging, do not perform this step. In this small sample of data, likely the userId and preferredProductId columns will only show null values. If you want to get a sense of how many records contain values for these fields, select a column, such as preferredProductId, then select **Statistics** in the toolbar above. This displays a chart for the column showing the ratio of values.



24. Select the + to the right of the JoinTopProductsWithPreferredProducts step, then select the **Derived Column** schema modifier from the context menu.



25. Under **Derived column's settings**, configure the following:

- **Output stream name:** Enter `DerivedColumnsForMerge`.
- **Incoming stream:** Select `JoinTopProductsWithPreferredProducts`.
- **Columns:** Provide the following information (*type in* the first two column names):

Column	Expression	Description
<code>isTopProduct</code>	<code>toBoolean(iif(isNull(productId), 'false', 'true'))</code>	Returns <code>true</code> if <code>productId</code> is not null. Recall that <code>productId</code> is fed by the e-commerce top user products data lineage.
<code>isPreferredProduct</code>	<code>toBoolean(iif(isNull(preferredProductId), 'false', 'true'))</code>	Returns <code>true</code> if <code>preferredProductId</code> is not null. Recall that <code>preferredProductId</code> is fed by the Azure Cosmos DB user profile data lineage.
<code>productId</code>	<code>iif(isNull(productId), preferredProductId, productId)</code>	Sets the <code>productId</code> output to either the <code>preferredProductId</code> or <code>productId</code> value, depending on whether <code>productId</code> is null.
<code>userId</code>	<code>iif(isNull(userId), visitorId, userId)</code>	Sets the <code>userId</code> output to either the <code>visitorId</code> or <code>userId</code> value, depending on whether <code>userId</code> is null.

Derived column's settings [Optimize](#) [Inspect](#) [Data preview](#) ●

Output stream name *	<input type="text" value="DerivedColumnsForMerge"/>	Learn more
Incoming stream *	<input type="button" value="JoinTopProductsWithPreferredProdu..."/>	
Columns * ⓘ	<input type="button" value="Add"/> <input type="button" value="Duplicate"/> <input type="button" value="Delete"/>	
Column	Expression	
<input type="checkbox"/> <code>isTopProduct</code>	<code>toBoolean(iif(isNull(productId), 'false', 'true'))</code>	<input type="button" value="+"/> <input type="button" value="Edit"/> <input type="button" value="Delete"/>
<input type="checkbox"/> <code>isPreferredProduct</code>	<code>toBoolean(iif(isNull(preferredProductId), 'false', 'true'))</code>	<input type="button" value="+"/> <input type="button" value="Edit"/> <input type="button" value="Delete"/>
<input type="checkbox"/> <code>productId</code>	<code>iif(isNull(productId), preferredProductId, productId)</code>	<input type="button" value="+"/> <input type="button" value="Edit"/> <input type="button" value="Delete"/>
<input type="checkbox"/> <code>userId</code>	<code>iif(isNull(userId), visitorId, userId)</code>	<input type="button" value="+"/> <input type="button" value="Edit"/> <input type="button" value="Delete"/>

The derived column settings provide the following result:

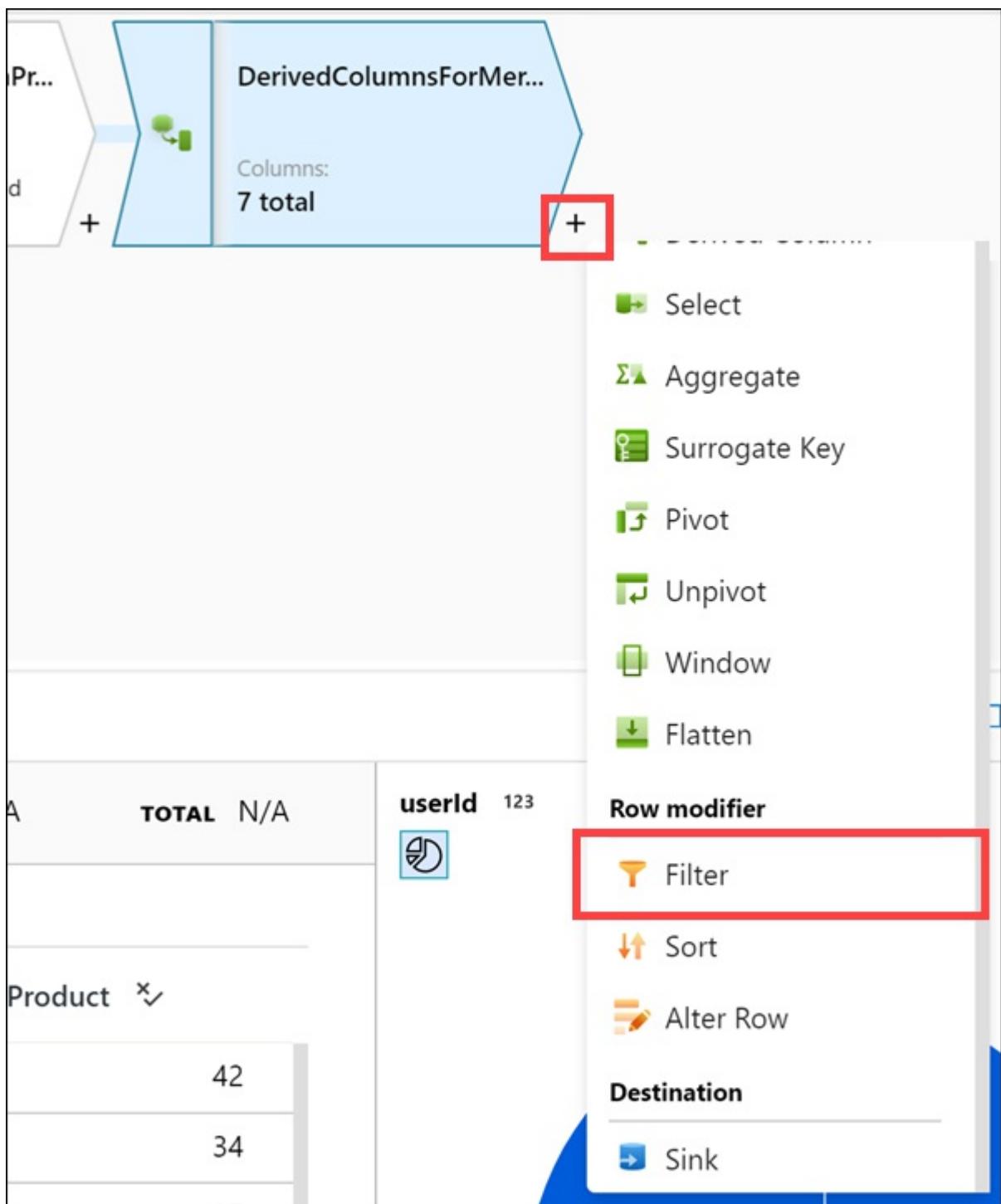
Derived column's settings Optimize Inspect Data preview ● Description

Number of rows + INSERT 100 * UPDATE 0 × DELETE 0 ⚡ UPsert 0 🔎 LOOKUP 0 TOTAL 15361

⟳ Refresh Typecast ⚓ Modify ⚓ Map drifted ⚓ Statistics X Remove

↑	userId	123	preferredProductId	123	visitorId	123	productId	123	itemsPurchasedLast12Months	123	isTopProduct	✗	isPreferredProduct	✗
+	9591082		NULL		9591082		372		9		✓		✗	
+	9591085		NULL		9591085		1731		87		✓		✗	
+	9591088		NULL		9591088		4820		72		✓		✗	
+	9591093		NULL		9591093		4101		35		✓		✗	
+	9591095		NULL		9591095		4861		96		✓		✗	
+	9591103		NULL		9591103		90		11		✓		✗	
+	9591105		NULL		9591105		1146		35		✓		✗	

26. Select the + to the right of the DerivedColumnsForMerge step, then select the Filter destination from the context menu.

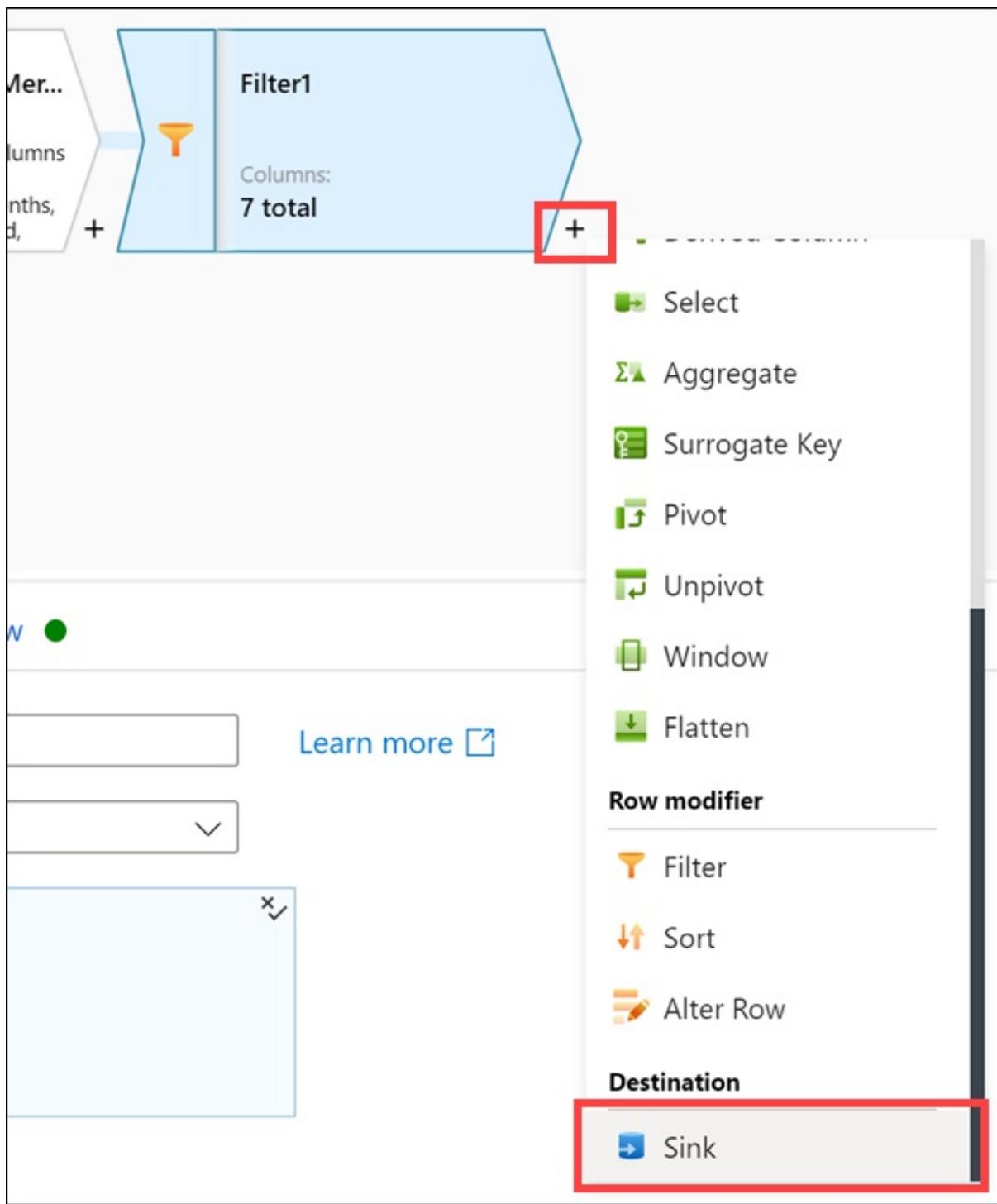


We are adding the Filter step to remove any records where the `ProductId` is null. The data sets have a small percentage of invalid records, and null `ProductId` values will cause errors when loading into the `UserTopProductPurchases` SQL pool table.

27. Set the **Filter on** expression to `!isNull(productId)`.

The screenshot shows the 'Filter settings' dialog box. At the top, there are tabs for 'Filter settings' (which is selected), 'Optimize', 'Inspect', and 'Data preview'. Below the tabs, there are three main input fields: 'Output stream name' set to 'Filter1', 'Incoming stream' set to 'DerivedColumnsForMerge', and 'Filter on' containing the expression '!isNull(productId)'. A red box highlights the 'Filter on' input field.

28. Select the + to the right of the `Filter1` step, then select the **Sink** destination from the context menu.



29. Under **Sink**, configure the following:

- **Output stream name:** Enter `UserTopProductPurchasesASA`.
- **Incoming stream:** Select `Filter1`.
- **Sink type:** Select `Dataset`.
- **Dataset:** Select `asal400_wwi_usertopproductpurchases_asa`, which is the `UserTopProductPurchases` SQL table.
- **Options:** Check `Allow schema drift` and uncheck `Validate schema`.

Sink Settings Mapping Optimize Inspect Data preview ●

Output stream name *	UserTopProductPurchasesASA	Learn more
Incoming stream *	Filter1	
Sink type *	Dataset	
Dataset *	asal400_wwi_usertopproductpurchas... Test connection	
Options	<input checked="" type="checkbox"/> Allow schema drift ⓘ <input type="checkbox"/> Validate schema ⓘ	

30. Select **Settings**, then configure the following:

- **Update method:** Check Allow insert and leave the rest unchecked.
- **Table action:** Select Truncate table.
- **Enable staging:** Check this option. Since we are importing a lot of data, we want to enable staging to improve performance.

Sink **Settings** Mapping Optimize Inspect Data preview ●

Update method	<input checked="" type="checkbox"/> Allow insert <input type="checkbox"/> Allow delete <input type="checkbox"/> Allow upsert <input type="checkbox"/> Allow update
Table action	<input type="radio"/> None <input type="radio"/> Recreate table <input checked="" type="radio"/> Truncate table
Enable staging	<input checked="" type="checkbox"/>
Batch size	<input type="text"/> ⓘ
Pre SQL scripts	<input type="text"/>

31. Select **Mapping**, then configure the following:

- **Auto mapping:** Uncheck this option.
- **Columns:** Provide the following information:

Input columns	Output columns
userId	UserId
productId	ProductId
itemsPurchasedLast12Months	ItemsPurchasedLast12Months
isTopProduct	IsTopProduct
isPreferredProduct	IsPreferredProduct

Sink Settings Mapping Optimize Inspect Data preview Description ^

Skip duplicate input columns ⓘ

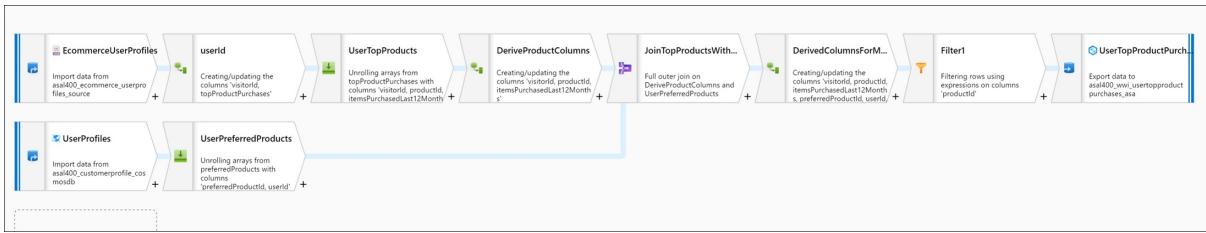
Skip duplicate output columns ⓘ

Auto mapping ⓘ

5 mappings: All outputs mapped

Input columns	Output columns
123 UserId	123 UserId
123 ProductId	123 ProductId
123 ItemsPurchasedLast12Months	123 ItemsPurchasedLast12Months
IsTopProduct	IsTopProduct
IsPreferredProduct	IsPreferredProduct

32. Your completed data flow should look similar to the following:



33. Select **Publish all** to save your new data flow.

Validate all

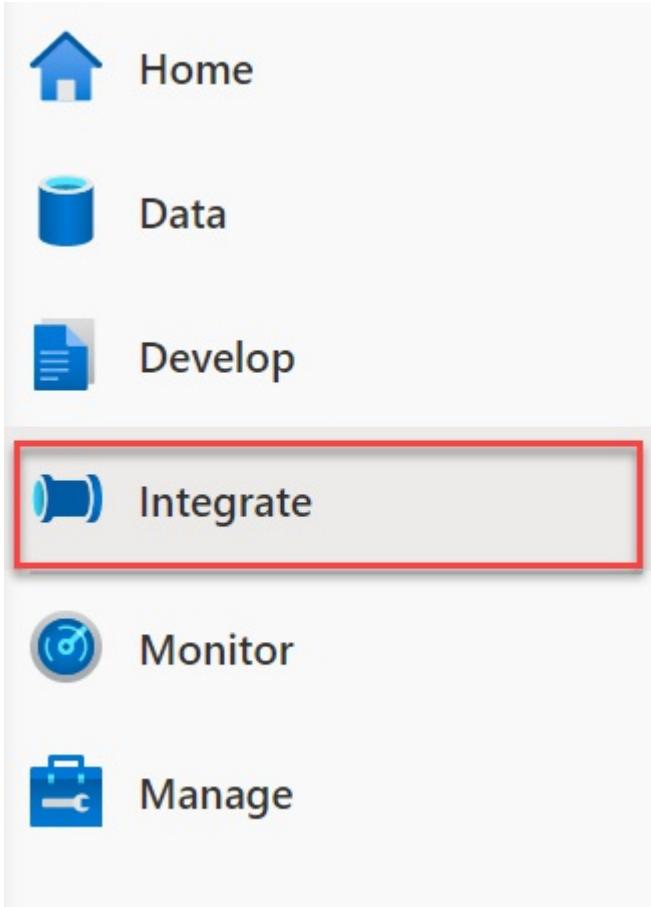
Develop + ⌂ << ASAL4

SQL scripts 23

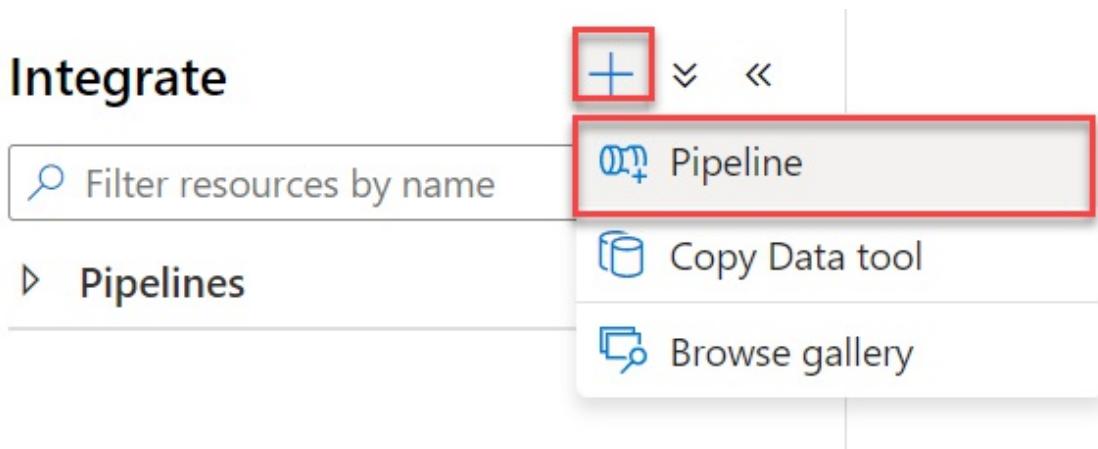
Task 2: Create user profile data pipeline

In order to run the new data flow, you need to create a new pipeline and add a data flow activity to it.

1. Navigate to the **Integrate** hub.



2. Select + then **Pipeline** to create a new pipeline.



3. In the **General** section of the **Properties** blade for the new pipeline, enter the following **Name**: ASAL400 - Lab 2 - Write User Profile Data to ASA.
4. Expand **Move & transform** within the Activities list, then drag the **Data flow** activity onto the pipeline canvas.

The screenshot shows the 'Activities' pane in the Azure Data Factory interface. At the top, there are buttons for 'Validate' (with a checkmark), 'Debug' (with a play icon), and 'Add trigger'. Below these are sections for 'Synapse', 'Move & transform', and 'Azure Functions'. Under 'Move & transform', the 'Data flow' activity is highlighted with a red arrow pointing towards it. The 'Data flow' activity is also highlighted with a dashed blue border. Other items in the list include 'Copy data', 'Azure Data Explorer', 'Azure Function', and 'Batch Service'.

5. In the Adding data flow blade, select **Use existing data flow**, then select the `asal400_lab2_writeuserprofiledataosa` existing data flow you created in the previous task.

The screenshot shows the 'Adding data flow' blade. It has two radio button options: 'Use existing data flow' (selected) and 'Create new data flow'. Below this is a dropdown menu labeled 'Existing data flow *' containing the option 'asal400_lab2_writeuserprofiledataosa'.

6. Select **Finish**.
7. Select the mapping data flow activity on the canvas. Select the **Settings** tab, then ensure `AutoResolveIntegrationRuntime` is selected for **Run on (Azure IR)**. Choose the `Compute Optimized Compute type` and select `32 (+ 16 cores)` for the **Core count**.
8. Expand **staging** and configure the following:
 - o **Staging linked service:** Select the `asadatalakeSUFFIX` linked service.
 - o **Staging storage folder:** Enter `staging/userprofiles`. The `userprofiles` folder will be automatically created for you during the first pipeline run.

9. Select **Publish all** to save your new pipeline.

Important: if your earlier pipeline run failed due to experiencing capacity-related issues and you were required to skip ahead to a fallback task, you will need to skip ahead again. The next task and the exercise that follows depend on your ability to successfully run your pipeline. If you cannot successfully run your pipeline, **skip ahead to Exercise 4b (fallback)** to see a successful outcome.

Task 3: Trigger, monitor, and analyze the user profile data pipeline

- Select **Add trigger** and select **Trigger now** in the toolbar at the top of the pipeline canvas.

✓ Validate ▶ Debug ⚡ Add trigger Data flow debug ✓

Trigger now

New/Edit

2. In the Pipeline run blade, select **OK** to start the pipeline run.

Pipeline run

⚠ Trigger pipeline now using last published configuration.

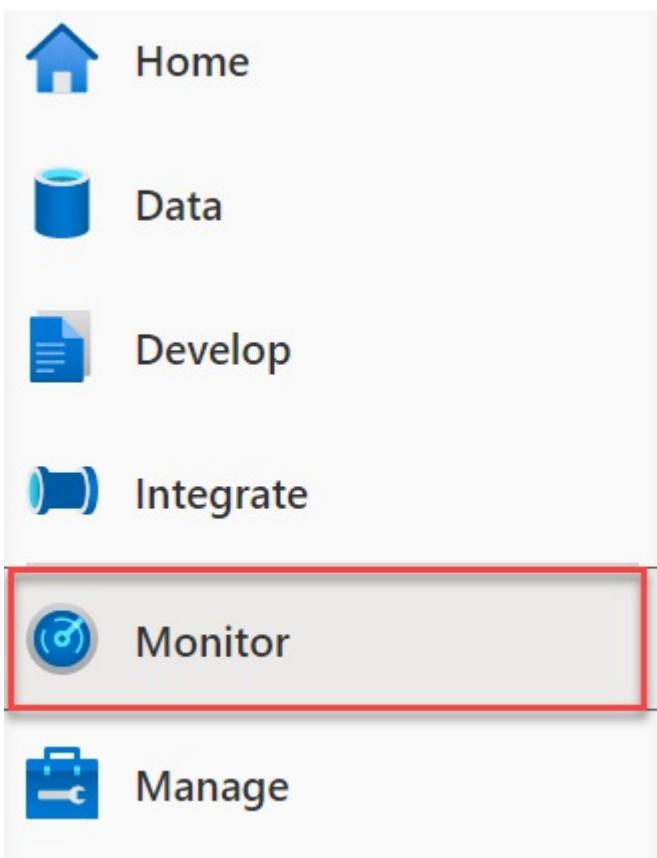
Parameters

NAME	TYPE	VALUE
No records found		

OK

Cancel

3. Navigate to the **Monitor** hub.



4. Wait for the pipeline run to successfully complete. You may need to refresh the view.

The screenshot shows the 'Pipeline runs' section of the Azure Data Factory interface. A single pipeline run is listed under 'Showing 1 - 7 items'. The run details are as follows:

- Pipeline Name:** ASAL400 - Lab 2 - Write User ...
- Run Start:** 8/14/20, 1:21:33 AM
- Duration:** 00:04:53
- Triggered By:** Manual trigger
- Status:** Succeeded (indicated by a green checkmark)
- Parameters:** (empty)

A red box highlights the 'Refresh' button at the top right of the table header, and another red box highlights the 'Succeeded' status in the table.

5. Select the name of the pipeline to view the pipeline's activity runs. Hover over the data flow activity name in the **Activity runs** list, then select the **Data flow details** icon.

The screenshot shows the 'Activity runs' page for the 'ASAL400 - Lab 2 - Write User Profile Data to ASA' pipeline. It displays one activity run:

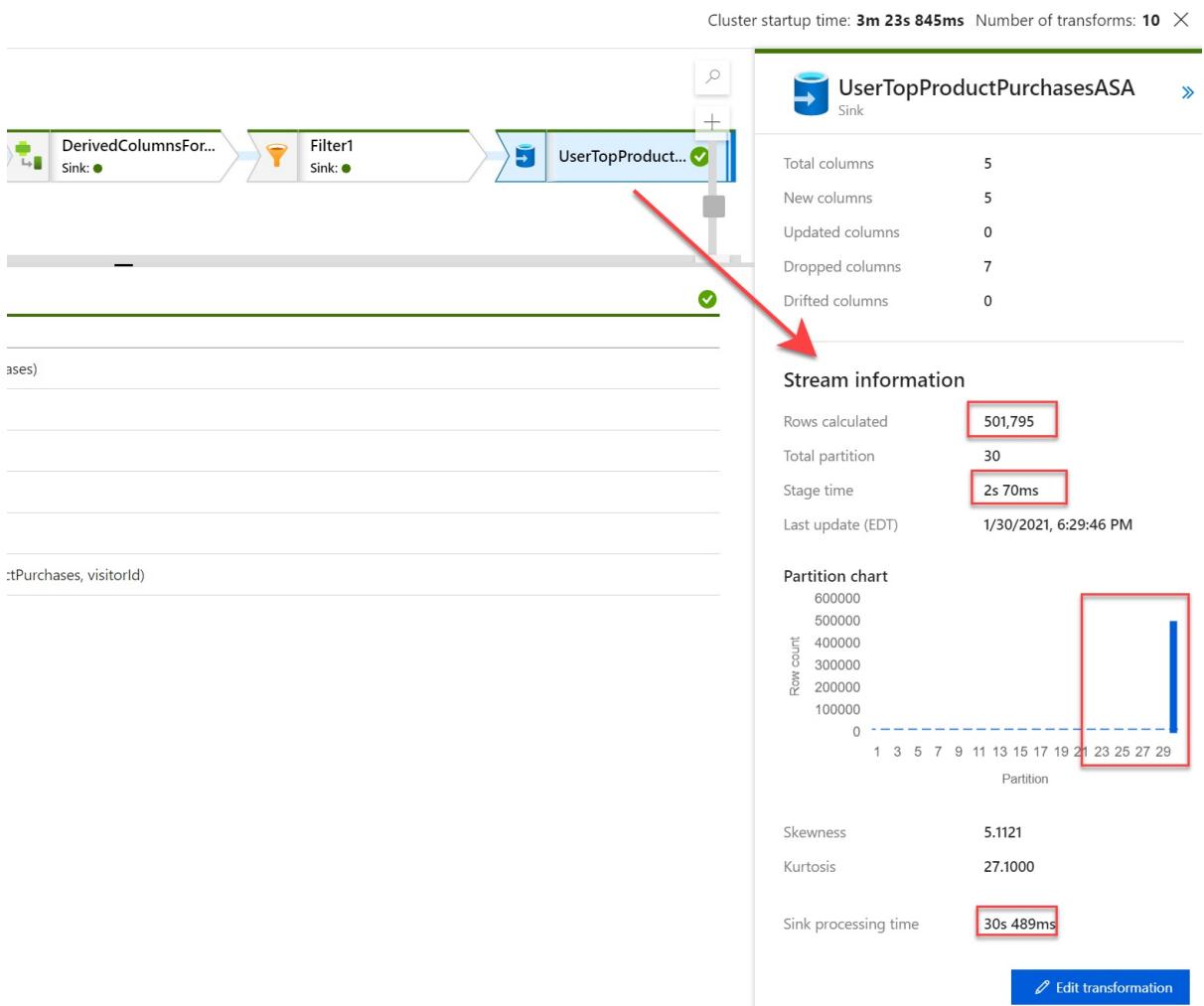
- Pipeline run ID:** 3e13d221-4aaa-449d-9f76-3d1e05036ee5
- Activity Name:** asal400_lab2_... (with a red box around the 'Data flow details' icon)
- Activity Type:** ExecuteDataFlow
- Run Start:** 8/14/20, 1:21:35 AM
- Duration:** 00:04:51
- Status:** Succeeded (indicated by a green checkmark)
- Integration Runtime:** AutoResolveIntegrationRuntime (West US 2)

A red box highlights the 'Data flow details' icon in the table row. To the right, a preview window titled 'Mapping Data Flow' shows the data flow steps: 'asal400_lab2_writeuserprofiledataosa'.

6. The data flow details displays the data flow steps and processing details. In our example, processing time took around 30 seconds to process and output around 510k rows. You can see which activities took longest to complete. The cluster startup time contributed almost three and a half minutes to the total pipeline run.



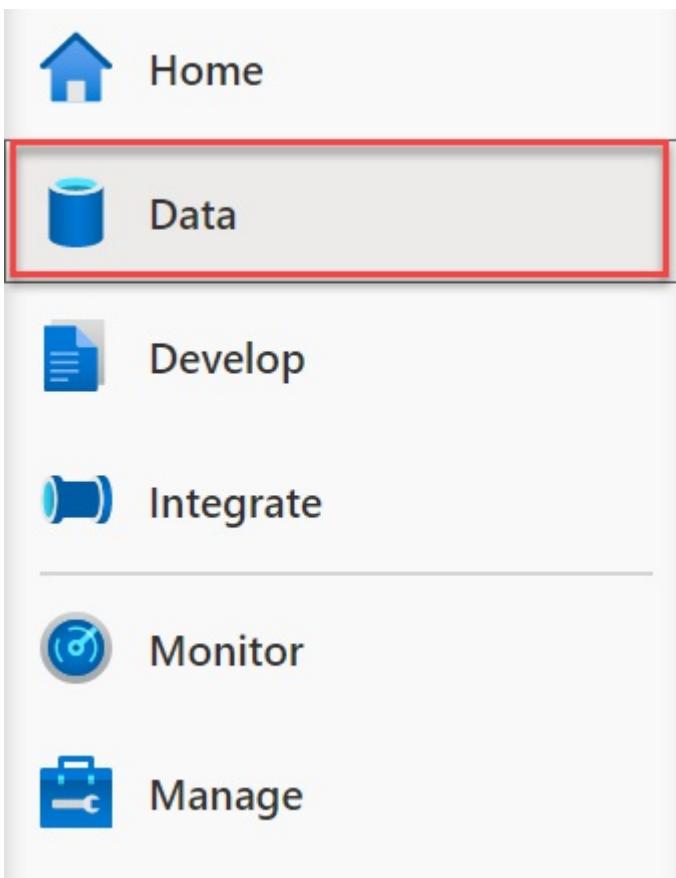
7. Select the **UserTopProductPurchasesASA** sink to view its details. We can see that 501,795 rows were calculated with a total of 30 partitions. It took around 3 seconds to stage the data in ADLS Gen2 prior to writing the data to the SQL table. The total sink processing time in our case was around 30 seconds. It is also apparent that we have a hot partition that is significantly larger than the others. If we need to squeeze extra performance out of this pipeline, we can re-evaluate data partitioning to more evenly spread the partitions to better facilitate parallel data loading and filtering. We could also experiment with disabling staging to see if there's a processing time difference. Finally, the size of the SQL Pool plays a factor in how long it takes to ingest data into the sink.



Exercise 4: Create Synapse Spark notebook to find top products

Now that we have processed, joined, and imported the user profile data, let's analyze it in greater detail. In this exercise, you will execute code to find the top 5 products for each user, based on which ones are both preferred and top, and have the most purchases in past 12 months. Then, you will calculate the top 5 products overall.

1. Navigate to the **Data hub**.



2. Expand the `SqlPool01` database underneath the **Databases** section. Right-click the `ww1.UserTopProductPurchases` table, then select the **Load to DataFrame** menu item under the New notebook context menu. If you don't see the table listed, select Refresh above.

The screenshot shows the Azure Data Studio interface in the Data workspace. The left pane displays the 'Databases' section, specifically 'SQLPool01 (SQL pool)', which contains several tables: wwi.CampaignAnalytics, wwi.Date, wwi.Product, wwi.Sale, wwi.SaleSmall, wwi.UserProductReviews, and wwi.UserTopProductPurchases. The 'wwi.UserTopProductPurchases' table is selected and highlighted with a red box. A context menu is open over this table, listing options such as 'New SQL script', 'New notebook', 'Data flow', 'Dataset', and 'Refresh'. The 'Load to DataFrame' option is also highlighted with a red box. The right pane shows the 'Activities' section with various options like Synapse, Move & transform, and Azure Function.

3. Attach the notebook to a Spark pool.

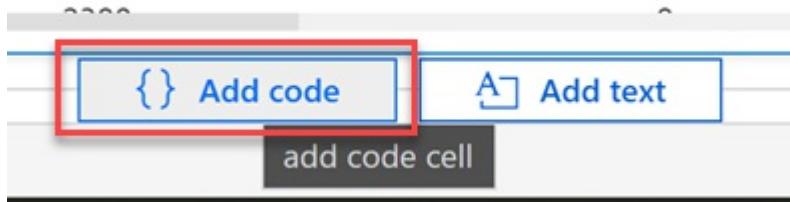
The screenshot shows the Azure Data Studio notebook toolbar. The 'Attach to' dropdown is open, displaying a list of available Spark pools: SparkPool01, SparkPool02, SparkPool03, SparkPool04, SparkPool05, SparkPool06, and 'Manage pools'. The 'Select Spark pool' dropdown is highlighted with a red box. The 'Language' dropdown is set to 'Spark (Scala)'. Below the toolbar, a code cell labeled 'Cell 1' contains Scala code: 'val df = spark.read.sqlanalytics("SQLPool01") // df.show(10)'.

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

Note: The first time you run a notebook in a Spark pool, Synapse creates a new session. This can take approximately 3-5 minutes.

Note: 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 then type **Ctrl+Enter** on your keyboard.

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



6. Enter and execute the following in the new cell to show the first 10 rows and to create a new temporary view named df:

```
df.head(10)  
  
df.createTempView("df")
```

The output should look similar to the following:

```
res3: Array[org.apache.spark.sql.Row] = Array([89792,2700,null,false,true],  
[89792,2338,null,false,true], [89792,4401,null,false,true], [89792,4423,null,false,true],  
[89792,1380,null,false,true], [6953,1296,null,false,true], [6953,1675,null,false,true],  
[20934,1395,null,false,true], [20934,891,null,false,true], [20934,657,null,false,true])
```

7. Notice that the language for this notebook is Spark Scala. We want to use Python to explore the data. To do this, we load the data into a temporary view, then we can load the view's contents into a DataFrame in a new PySpark cell. To do this, execute the following in a new cell:

```
%%pyspark  
# Calling the DataFrame df created in Scala to Python  
df = sqlContext.table("df")  
# *****  
  
topPurchases = df.select(  
    "UserId", "ProductId",  
    "ItemsPurchasedLast12Months", "IsTopProduct",  
    "IsPreferredProduct")  
  
topPurchases.show(100)
```

We set the language of the cell to PySpark with the `%%pyspark` magic. Then we loaded the `df` view into a new DataFrame. Finally, we created a new DataFrame named `topPurchases` and displayed its contents.

Cell 3

```
[9]   1 %%pyspark
      2 # Calling the dataframe df created in Scala to Python
      3 df = sqlContext.table("df")
      4 # ****
      5
      6 topPurchases = df.select(
      7     "UserId", "ProductId",
      8     "ItemsPurchasedLast12Months", "IsTopProduct",
      9     "IsPreferredProduct")
     10
     11 topPurchases.show(100)
```

Command executed in 10s 524ms by joel on 04-20-2020 15:22:20.352 -04:00

► Job execution Succeeded Spark 2 executors 8 cores

UserId	ProductId	ItemsPurchasedLast12Months	IsTopProduct	IsPreferredProduct
9065916	3020	null	false	true
9065916	2735	null	false	true
9065916	1149	null	false	true
9065916	2594	null	false	true
9065916	4591	null	false	true
9065916	3012	null	false	true
9065916	1985	null	false	true
9065916	1773	null	false	true
9065916	380	null	false	true
9068349	4383	null	false	true
9068349	681	null	false	true
9068349	852	null	false	true
9068349	4290	null	false	true
9068349	225	null	false	true
9068349	2014	null	false	true
9068349	4135	null	false	true

8. Execute the following in a new cell to create a new DataFrame to hold only top preferred products where IsTopProduct is true:

```
%%pyspark
from pyspark.sql.functions import *

topPreferredProducts = (topPurchases
    .filter( col("IsTopProduct") == True)
    .orderBy( col("ItemsPurchasedLast12Months").desc() ))

topPreferredProducts.show(100)
```



```
1  %%pyspark
2  from pyspark.sql.functions import *
3
4  topPreferredProducts = (topPurchases
5  |    .filter( col("IsTopProduct") == True)
6  |    .orderBy( col("ItemsPurchasedLast12Months").desc() ))
7
8  topPreferredProducts.show(100)
9
```

Command executed in 4s 730ms by odl_user_291899 on 01-30-2021 18:56:52.211 -05:00

> **Job execution** Succeeded Spark 2 executors 8 cores



UserId	ProductId	ItemsPurchasedLast12Months	IsTopProduct	IsPreferredProduct
120000	1486	91	true	false
120000	4985	83	true	false
120000	3371	53	true	false
120000	4433	32	true	false
120000	2961	28	true	false
120000	1724	13	true	false
120000	3299	8	true	false

9. Execute the following in a new cell to create a new temporary view by using SQL:

```
%%sql

CREATE OR REPLACE TEMPORARY VIEW top_5_products
AS
  select UserId, ProductId, ItemsPurchasedLast12Months
  from (select *,
              row_number() over (partition by UserId order by ItemsPurchasedLast12Months desc) as seqnum
        from df
      ) a
  where seqnum <= 5 and IsTopProduct == true
  order by a.UserId
```

Note that there is no output for the above query. The query uses the `df` temporary view as a source and applies a `row_number()` over method to apply a row number for the records for each user where `ItemsPurchasedLast12Months` is greatest. The `where` clause filters the results so we only retrieve up to five products where both `IsTopProduct` and `IsPreferredProduct` are set to true. This gives us the top five most purchased products for each user where those products are also identified as their favorite products, according to their user profile stored in Azure Cosmos DB.

10. Execute the following in a new cell to create and display a new DataFrame that stores the results of the `top_5_products` temporary view you created in the previous cell:

```
%%pyspark

top5Products = sqlContext.table("top_5_products")

top5Products.show(100)
```

You should see an output similar to the following, which displays the top five preferred products per user:



```
1 %%pyspark
2
3 top5Products = sqlContext.table("top_5_products")
4
5 top5Products.show(100)
```

Command executed in 6s 755ms by odl_user_291899 on 01-30-2021 19:00:43.761 -05:00

> Job execution Succeeded Spark 2 executors 8 cores



UserId	ProductId	ItemsPurchasedLast12Months
120000	1486	91
120000	4985	83
120000	3371	53
120000	4433	32
120000	2961	28

11. Execute the following in a new cell to compare the number of top preferred products to the top five preferred products per customer:

```
%%pyspark
print('before filter: ', topPreferredProducts.count(), ', after filter: ', top5Products.count())
```

The output should be similar to before filter: 7, after filter: 5.

12. Finally, let's calculate the top five products overall, based on those that are both preferred by customers and purchased the most. To do this, execute the following in a new cell:

```
%%pyspark

top5ProductsOverall = (top5Products.select("ProductId", "ItemsPurchasedLast12Months")
    .groupBy("ProductId")
    .agg( sum("ItemsPurchasedLast12Months").alias("Total") )
    .orderBy( col("Total").desc() )
    .limit(5))

top5ProductsOverall.show()
```

In this cell, we grouped the top five preferred products by product ID, summed up the total items purchased in the last 12 months, sorted that value in descending order, and returned the top five results. Your output should be similar to the following:

ProductId	Total
2107	91
4833	83
347	53
3459	32
4246	28

Exercise 4b (fallback) Monitor and analyze the user profile data pipeline and create Synapse Spark notebook to find top products

Read this exercise if you are unable to run the pipelines due to capacity-related issues.

For illustrative purposes, we have triggered the user profile pipeline that runs the data flow that processes, joins, and imports user profile data into a Synapse SQL Pool table.

The **Monitor** hub contains, among other things, pipeline runs. When the pipeline run is successful, we select the name of the pipeline to view its activity runs. Notice that the custom `AzureLargeComputeOptimizedIntegrationRuntime` IR was used. To view its details, we hover over the data flow activity name in the **Activity runs** list, then select the **Data flow details** icon.

All pipeline runs > ASAL400 - Lab 2 - Write User Profile Data to ASA - Activity runs

ASAL400 - Lab 2 - Write User Profile Data to ASA

[List](#) [Gantt](#)

[Rerun](#) [Rerun from activity](#) [Rerun from failed activity](#) [Refresh](#)

Mapping Data Flow [Edit](#) [Checkmark](#)
 asal400_lab2_writeuserprofiledatatoasa

+ - [New](#) [Edit](#) [Delete](#)

Activity runs

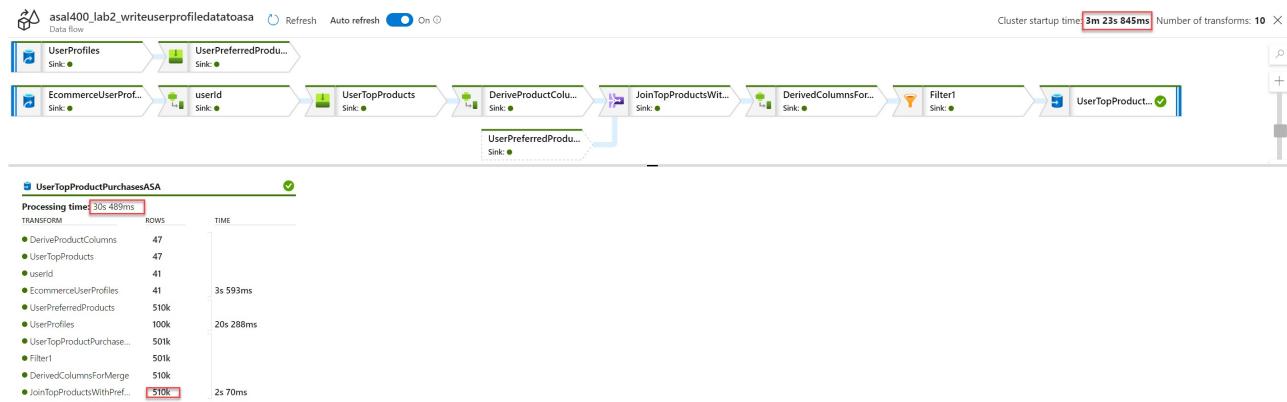
Pipeline run ID 3e13d221-4aaa-449d-9f76-3d1e05036ee5

All status [▼](#)

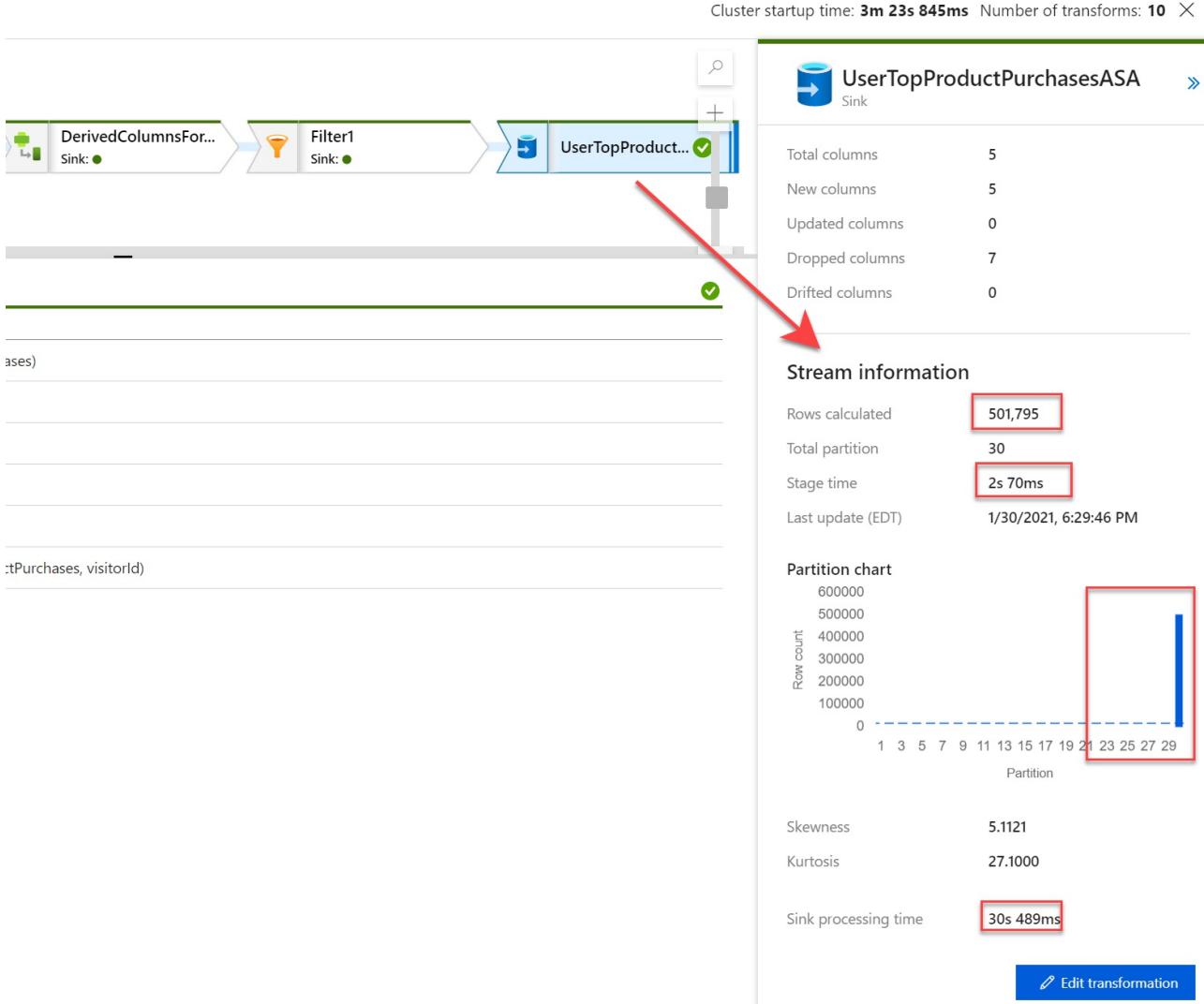
Showing 1 - 1 of 1 items

ACTIVITY NAME	ACTIVITY TYPE	RUN START ↑	DURATION	STATUS	INTEGRATION RUNTIME
asal400_lab2... Edit ExecuteDataFlow Data flow details	ExecuteDataFlow	8/14/20, 1:21:35 AM	00:04:51	Succeeded	AutoResolveIntegrationRuntime (West US 2)

The data flow details displays the data flow steps and processing details. In our example, processing time took around 30 seconds to process and output around 510k rows. You can see which activities took longest to complete. The cluster startup time contributed almost three and a half minutes to the total pipeline run.



Here we select the `UserTopProductPurchasesASA` sink to view its details. We can see that 15,308,766 rows were calculated with a total of 30 partitions. It took around seven seconds to stage the data in ADLS Gen2 prior to writing the data to the SQL table. The total sink processing time in our case was around 45 seconds. It is also apparent that we have a hot partition that is significantly larger than the others. If we need to squeeze extra performance out of this pipeline, we can re-evaluate data partitioning to more evenly spread the partitions to better facilitate parallel data loading and filtering. We could also experiment with disabling staging to see if there's a processing time difference. Finally, the size of the SQL Pool plays a factor in how long it takes to ingest data into the sink.



Now that we have processed, joined, and imported the user profile data, let's analyze it in greater detail. In the example that follows, we execute code to find the top 5 products for each user, based on which ones are both preferred and top, and have the most purchases in past 12 months. Then, we calculate the top 5 products overall.

The easiest way to create a new notebook to explore the `UserTopProductPurchases` table, which we populated with the data flow, is to navigate to the **Data** hub, expand the `SqlPool01` database underneath the **Databases** section, right-click the `wwi.UserTopProductPurchases` table, then select the **Load to DataFrame** menu item under the New notebook context menu.

The notebook's language is set to Spark (Scala) by default. The first cell is populated with code that creates a new DataFrame from the `spark.read.sqlanalytics` method, which reads from the table in the SQL Pool. We update the cell to show the first 10 records (`df.head(10)`) and to create a new temporary view named "df":

```
val df = spark.read.sqlanalytics("SQLPool01.wwi.UserTopProductPurchases")
df.head(10)

df.createTempView("df")
```

The output looks like the following:

```
res3: Array[org.apache.spark.sql.Row] = Array([89792,2700,null,false,true], [89792,2338,null,false,true],
[89792,4401,null,false,true], [89792,4423,null,false,true], [89792,1380,null,false,true],
[6953,1296,null,false,true], [6953,1675,null,false,true], [20934,1395,null,false,true],
[20934,891,null,false,true], [20934,657,null,false,true])
```

Notice that the language for this notebook is Spark Scala. We want to use Python to explore the data. To do this, we load the data into a temporary view, then we can load the view's contents into a DataFrame in a new PySpark cell.

```

%%pyspark
# Calling the DataFrame df created in Scala to Python
df = sqlContext.table("df")
# *****

topPurchases = df.select(
    "UserId", "ProductId",
    "ItemsPurchasedLast12Months", "IsTopProduct",
    "IsPreferredProduct")

topPurchases.show(100)

```

We set the language of the cell to PySpark with the `%%pyspark` magic. Then we loaded the `df` view into a new DataFrame. Finally, we created a new DataFrame named `topPurchases` and displayed its contents.

Cell 3

```

[9]   1 %%pyspark
      2 # Calling the dataframe df created in Scala to Python
      3 df = sqlContext.table("df")
      4 # *****

      5
      6 topPurchases = df.select(
      7     "UserId", "ProductId",
      8     "ItemsPurchasedLast12Months", "IsTopProduct",
      9     "IsPreferredProduct")
     10
     11 topPurchases.show(100)

```

Command executed in 10s 524ms by joel on 04-20-2020 15:22:20.352 -04:00

► **Job execution** Succeeded **Spark** 2 executors 8 cores

UserId	ProductId	ItemsPurchasedLast12Months	IsTopProduct	IsPreferredProduct
9065916	3020	null	false	true
9065916	2735	null	false	true
9065916	1149	null	false	true
9065916	2594	null	false	true
9065916	4591	null	false	true
9065916	3012	null	false	true
9065916	1985	null	false	true
9065916	1773	null	false	true
9065916	380	null	false	true
9068349	4383	null	false	true
9068349	681	null	false	true
9068349	852	null	false	true
9068349	4290	null	false	true
9068349	225	null	false	true
9068349	2014	null	false	true
9068349	4135	null	false	true

The following cell creates a new DataFrame to hold only top preferred products where `IsTopProduct` is true:

```

%%pyspark
from pyspark.sql.functions import *

topPreferredProducts = (topPurchases
    .filter( col("IsTopProduct") == True)
    .orderBy( col("ItemsPurchasedLast12Months").desc() ))

topPreferredProducts.show(100)

```

```

1 %%pyspark
2 from pyspark.sql.functions import *
3
4 topPreferredProducts = (topPurchases
5     .filter( col("IsTopProduct") == True)
6     .orderBy( col("ItemsPurchasedLast12Months").desc() ))
7
8 topPreferredProducts.show(100)
9

```

Command executed in 4s 730ms by odl_user_291899 on 01-30-2021 18:56:52.211 -05:00

> **Job execution** Succeeded **Spark** 2 executors 8 cores



UserId	ProductId	ItemsPurchasedLast12Months	IsTopProduct	IsPreferredProduct
120000	1486	91	true	false
120000	4985	83	true	false
120000	3371	53	true	false
120000	4433	32	true	false
120000	2961	28	true	false
120000	1724	13	true	false
120000	3299	8	true	false

This cell creates a new temporary view by using SQL:

```

%%sql

CREATE OR REPLACE TEMPORARY VIEW top_5_products
AS
    select UserId, ProductId, ItemsPurchasedLast12Months
    from (select *,
        row_number() over (partition by UserId order by ItemsPurchasedLast12Months desc) as seqnum
    from df
    ) a
    where seqnum <= 5 and IsTopProduct == true
    order by a.UserId

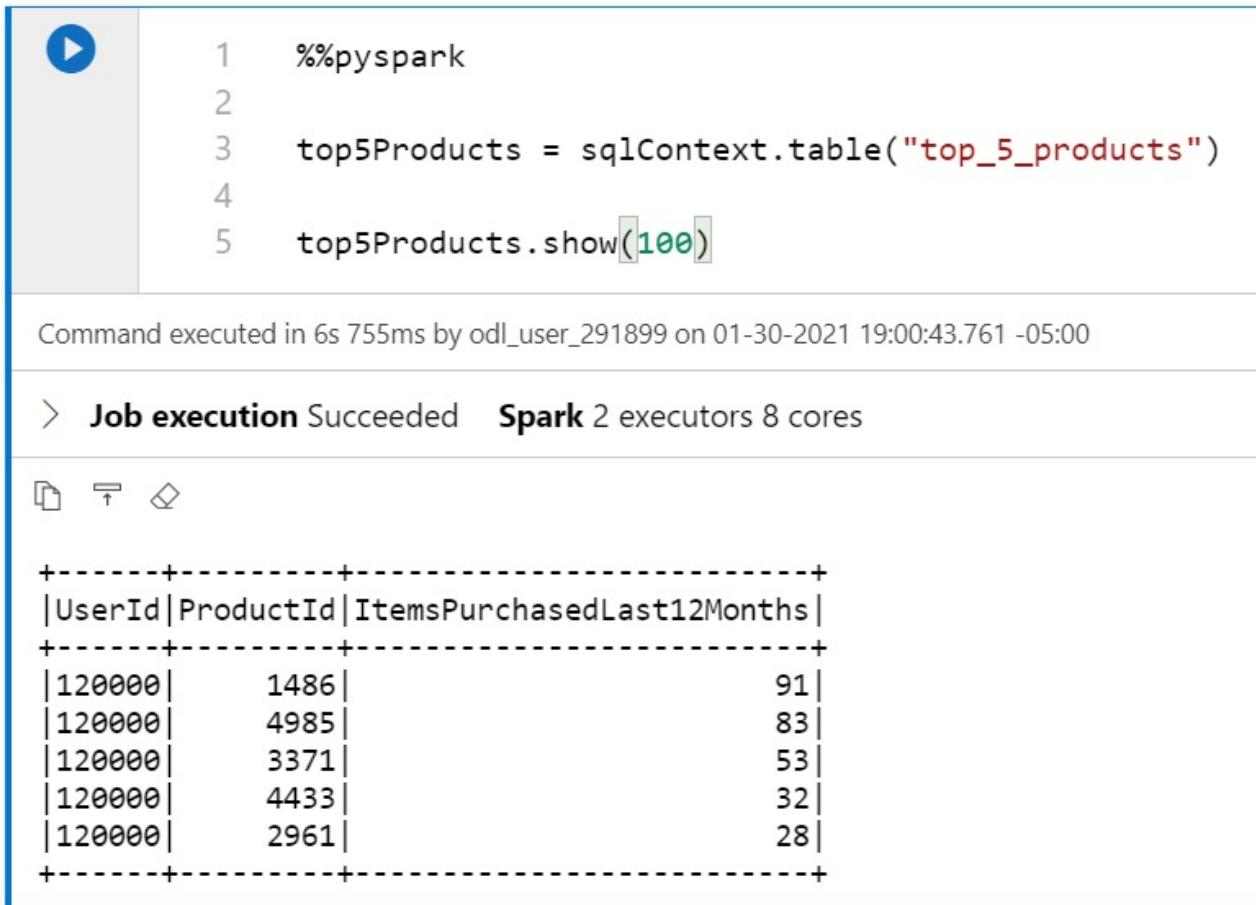
```

Note that there is no output for the above query. The query uses the df temporary view as a source and applies a `row_number()` over method to apply a row number for the records for each user where `ItemsPurchasedLast12Months` is greatest. The `where` clause filters the results so we only retrieve up to five products where both `IsTopProduct` and `IsPreferredProduct` are set to true. This gives us the top five most purchased products for each user where those products are also identified as their favorite products, according to their user profile stored in Azure Cosmos DB.

The following cell creates and displays a new DataFrame that stores the results of the `top_5_products` temporary view that was created in the previous cell:

```
%%pyspark
top5Products = sqlContext.table("top_5_products")
top5Products.show(100)
```

The output displays the top five preferred products per user:



A screenshot of a Jupyter Notebook cell. The cell contains the following PySpark code:

```
1 %%pyspark
2
3 top5Products = sqlContext.table("top_5_products")
4
5 top5Products.show(100)
```

Below the code, the output shows the command was executed in 6s 755ms by odl_user_291899 on 01-30-2021 19:00:43.761 -05:00. The job execution status is "Succeeded" with Spark 2 executors and 8 cores.

The output displays a table with three columns: UserId, ProductId, and ItemsPurchasedLast12Months. The data is as follows:

UserId	ProductId	ItemsPurchasedLast12Months
120000	1486	91
120000	4985	83
120000	3371	53
120000	4433	32
120000	2961	28

This cell compares the number of top preferred products to the top five preferred products per customer:

```
%%pyspark
print('before filter: ', topPreferredProducts.count(), ', after filter: ', top5Products.count())
```

The output is before filter: 7, after filter: 5.

Finally, this cell calculates the top five products overall, based on those that are both preferred by customers and purchased the most.

```
%%pyspark
top5ProductsOverall = (top5Products.select("ProductId", "ItemsPurchasedLast12Months")
    .groupBy("ProductId")
    .agg( sum("ItemsPurchasedLast12Months").alias("Total") )
    .orderBy( col("Total").desc() )
    .limit(5))

top5ProductsOverall.show()
```

We grouped the top five preferred products by product ID, summed up the total items purchased in the last 12 months, sorted that value in descending order, and returned the top five results. The output is:

ProductId	Total
2107	91
4833	83
347	53
3459	32
4246	28