

Data Integration Part 2

Resource naming throughout this lab

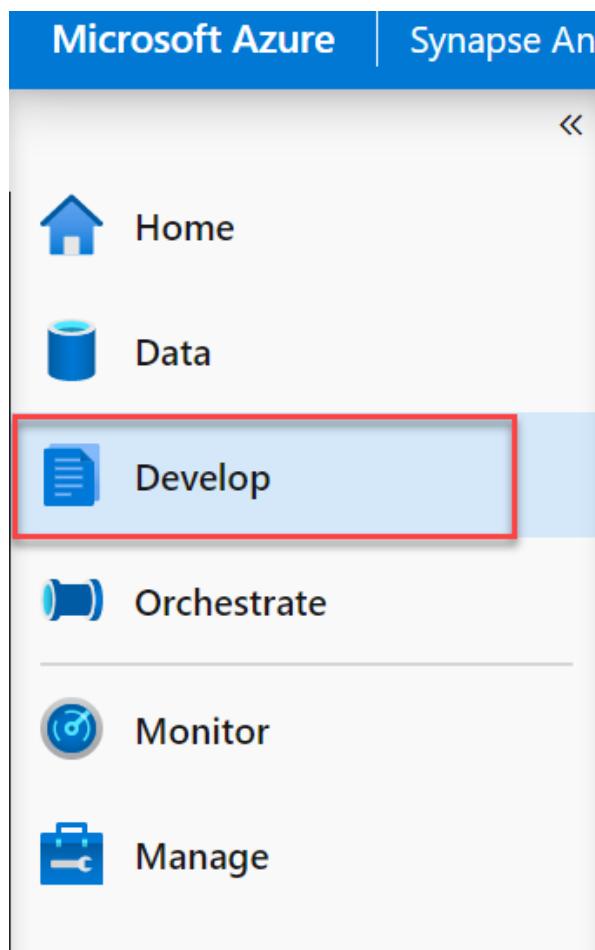
For the remainder of this guide, the following terms will be used for various ASA-related resources (make sure you replace them with actual names and values):

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	SqlPool01

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 Azure Data Studio interface. At the top, there are buttons for 'Publish all', 'Validate all', 'Refresh', and 'Discard'. Below the toolbar, the 'Develop' tab is selected. A dropdown menu is open, showing options: 'SQL script' (which is highlighted with a red box), 'Notebook', 'Data flow', 'Spark job definition', 'Power BI report', and 'Import'. On the left, a sidebar lists categories: 'SQL scripts', 'Notebooks', 'Data flows', 'Power BI' (which is expanded), and 'Data & AI Demo'.

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

The screenshot shows the Azure Data Studio toolbar. The 'Connect to' dropdown menu is highlighted with a red box. Other items in the toolbar include 'Run', 'Publish', 'Query plan', 'Use database', and a refresh button.

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.

The screenshot shows the Azure Data Studio toolbar. The 'Run' button is highlighted with a red box. Other items in the toolbar include 'Publish', 'Query plan', 'Connect to', and a refresh button.

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.

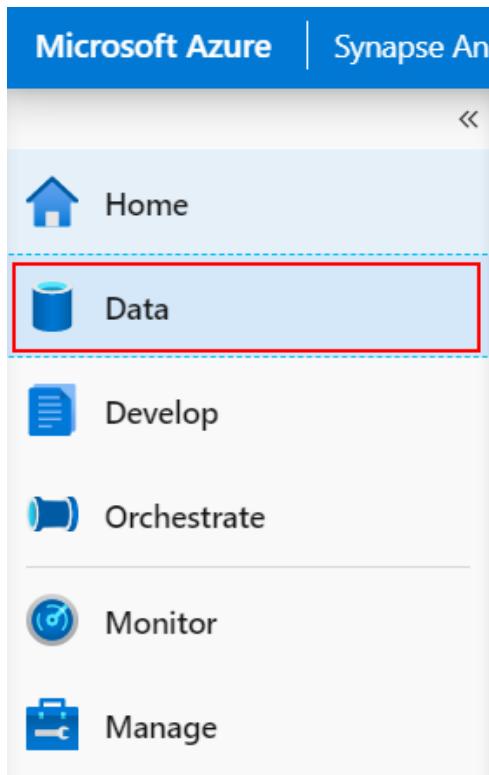
Task 2: Create campaign analytics datasets

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.

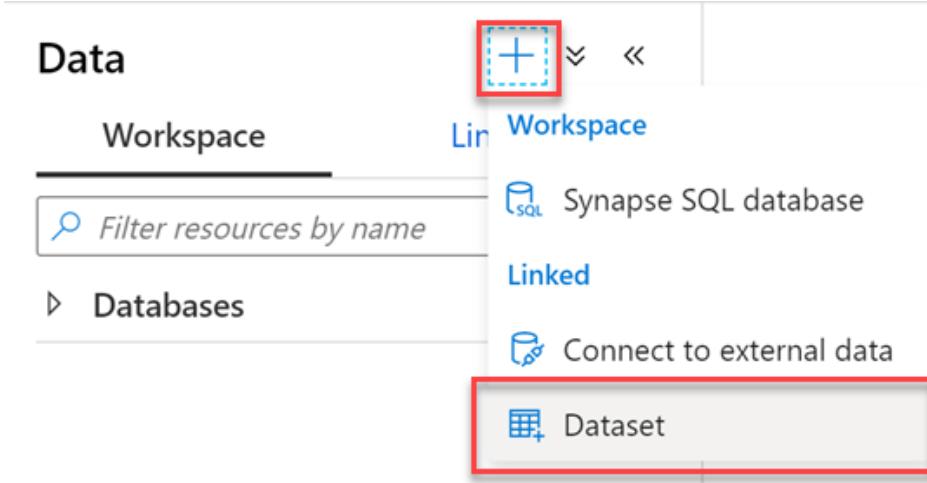
Region	Country	Product_Category	Campaign_Name	Revenue	Revenue_T	City	State	
Europe	Germany	Apparel and Footwear	Fun with Colors	\$14\	865	\$15\	960	
Far West	US	Books	EnjoyTheMoment;	\$14\	992	\$15\	699	San Diego California
Europe	Germany	Apparel and Footwear	Fall into Winter	\$5\	117	\$8\	713	
Far West	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** 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

Linked service *

 ▼ 

File path

 / / 📁 | ▾

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.

Preview data



Linked service: asadatalake01

Object: campaignanalytics.csv

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** dataset with the following characteristics:

- **Name:** Enter `asal400_wwi_campaign_analytics_asa`.
- **Linked service:** Select the `SqlPool01` service.
- **Table name:** Select `wwi.CampaignAnalytics`.
- **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

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** 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

2. Create a new **Azure Synapse Analytics** 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

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

The screenshot shows the Azure Data Factory interface. At the top, there are three buttons: 'Publish all' (with a yellow circle containing '1'), 'Validate all', and 'Refresh'. Below this is the 'Develop' tab, which has a '+' icon, a downward arrow, and a double-left arrow. A search bar with the placeholder 'Filter resources by name' is present. On the right, there's a sidebar with the name 'ASAL4' and 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:

- **Name:** Enter [asal400_customerprofile_cosmosdb](#).
- **Linked service:** Select the Azure Cosmos DB linked service.
- **Collection:** Select [OnlineUserProfile01](#).

The screenshot shows the 'Set properties' dialog for creating a dataset. It includes fields for Name (set to 'asal400_customerprofile_cosmosdb'), Linked service (set to 'asacosmosdb01'), Collection (set to 'OnlineUserProfile01'), and Import schema (set to 'From connection/store').

Set properties

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 *

Collection

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.

General Connection Schema **Parameters**

Import schema Clear

Column name	Type
userId	123 integer
cartId	abc string
preferredProducts	[] any[]
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:

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

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

Publish all (1) Validate all Refresh

Develop + << ASAL4

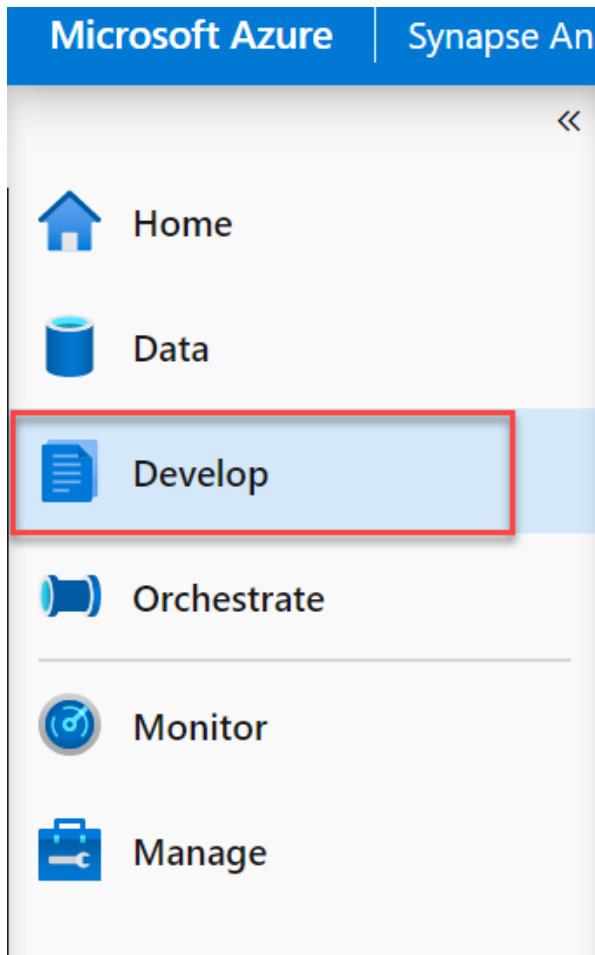
Filter resources by name Valid

SQL scripts 23

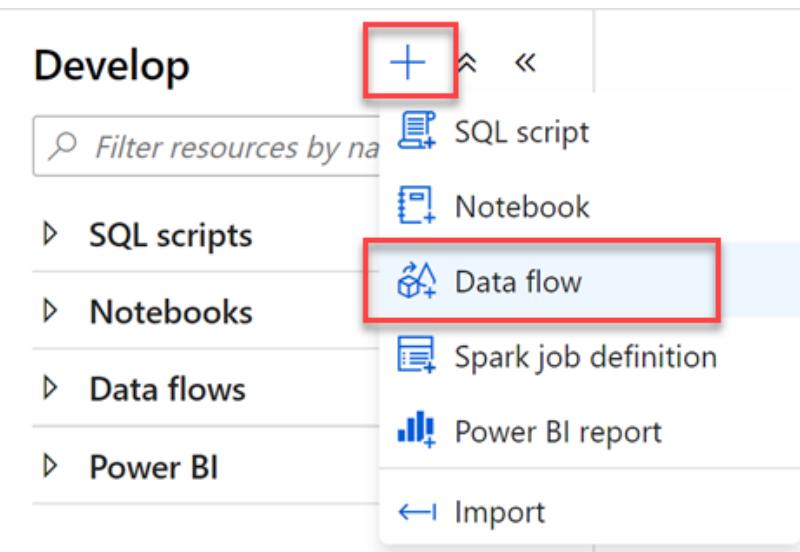
Exercise 2: Create data pipeline to import poorly formatted CSV

Task 1: Create campaign analytics data flow

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



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



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

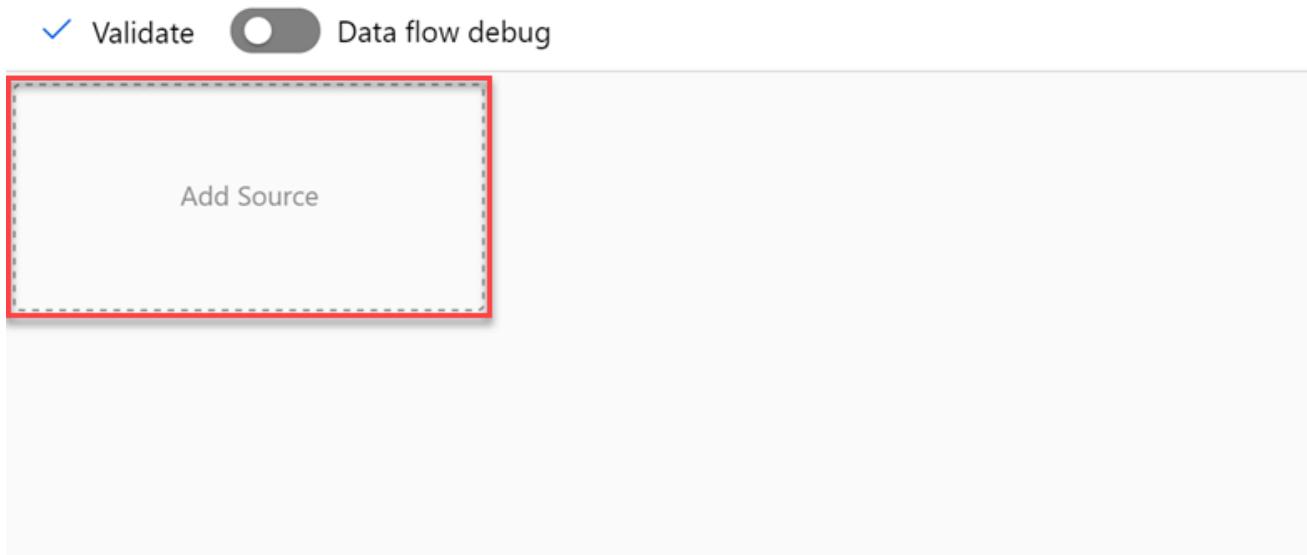
Choose a name for your data flow.
This name can be updated at any time until it is published.

Name *

asal400_lab2_writecampaignanalyticstoasa

Description

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



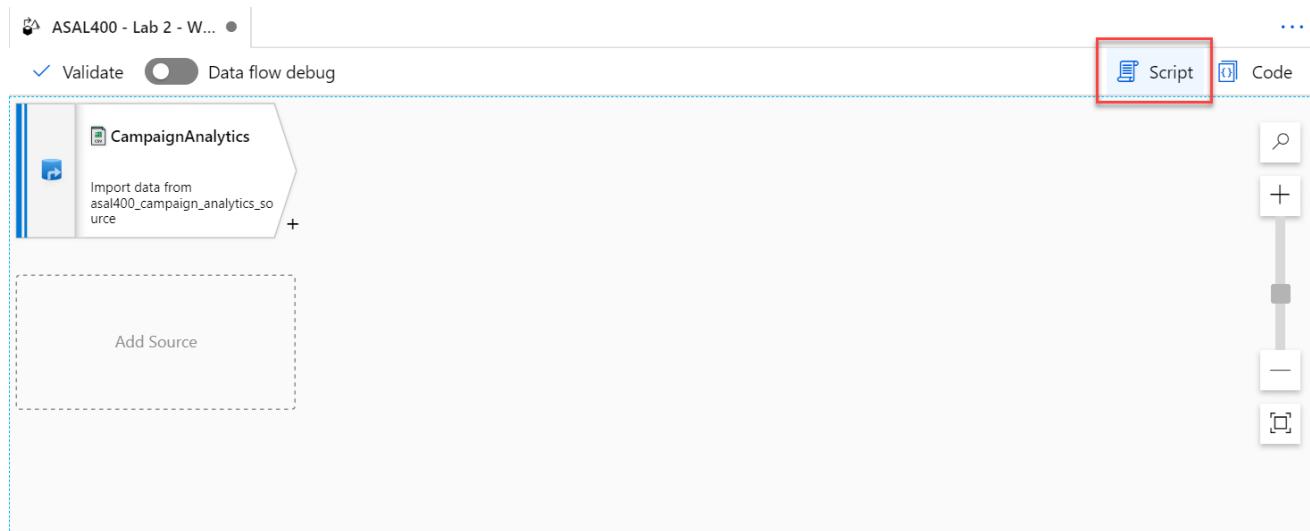
5. Under **Source settings**, configure the following:

- **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**.

The screenshot shows the 'Source settings' tab selected in the navigation bar. The configuration fields are as follows:

- Output stream name ***: CampaignAnalytics
- Source type ***: Dataset
- Dataset ***: asal400_campaign_analytics_source (with a dropdown arrow, a 'Test connection' button, an 'Open' button, and a 'New' button)
- Options**:
 - Allow schema drift
 - Infer drifted column types
 - Validate schema
- Skip line count**: 1
- Sampling ***: Enable Disable

2. 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.



3. 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

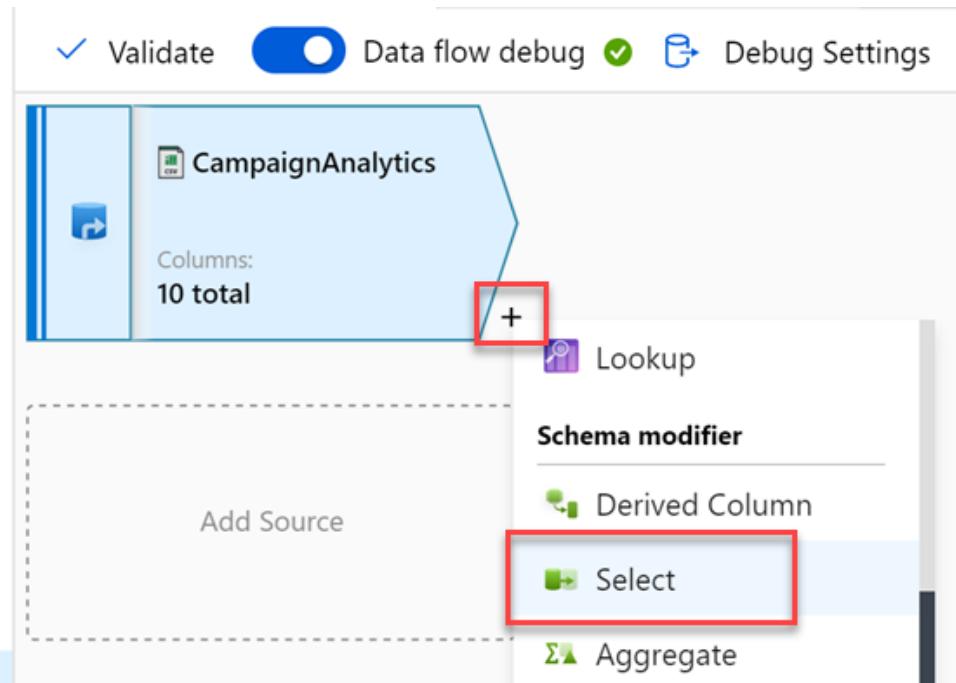
Copy as single line

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.



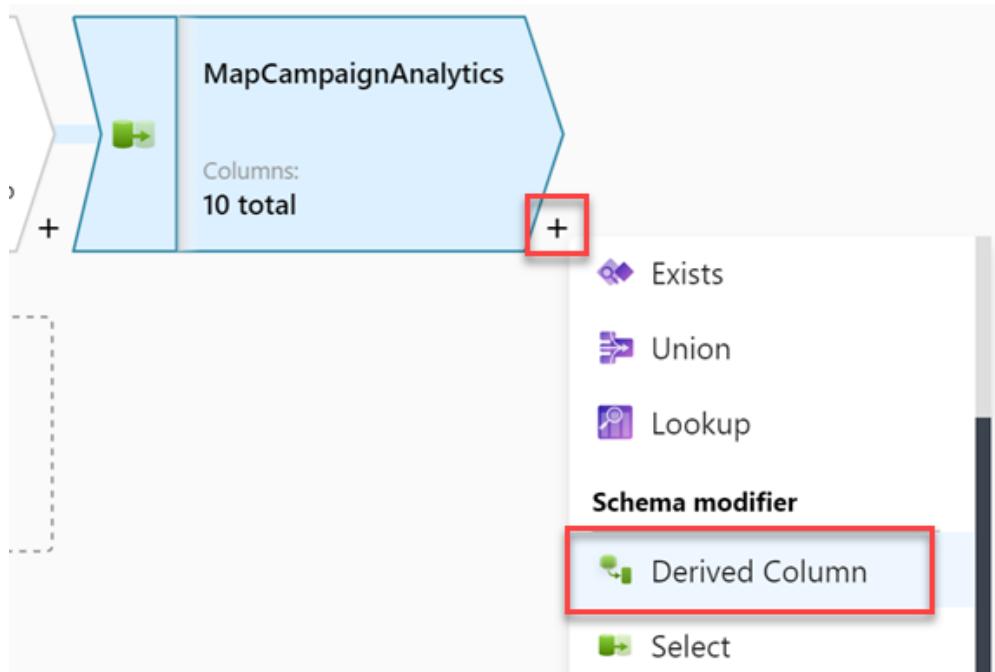
10. Under **Select settings**, configure the following:

- **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

The screenshot shows the 'Select settings' configuration page. At the top, there are tabs: 'Select settings' (selected), 'Optimize', 'Inspect', and 'Data preview'. Below the tabs, there are fields for 'Output stream name' (set to `MapCampaignAnalytics`) and 'Incoming stream' (set to `CampaignAnalytics`). Under 'Options', two checkboxes are checked: 'Skip duplicate input columns' and 'Skip duplicate output columns'. Below these, there's a 'Input columns' section with a 'Auto mapping' toggle (unchecked) and a 'Reset' button. A table below lists 10 input columns from the 'CampaignAnalytics' source and their corresponding 'Name as' fields in the 'MapCampaignAnalytics' stream.

CampaignAnalytics's column	Name as
abc_col0_	Region
abc_col1_	Country
abc_col2_	ProductCategory
abc_col3_	CampaignName
abc_col4_	RevenuePart1
12_col5_	Revenue
abc_col6_	RevenueTargetPart1
12_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:

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

Column	Expression	Description
Revenue	<code>toDecimal(replace(concat(toString(RevenuePart1), toString(Revenue)), '\\', ''), 10, 2, '\$###,###.##')</code>	Concatenate the RevenuePart1 and Revenue fields, replace the invalid \ character, then convert and format the data to a decimal type.
RevenueTarget	<code>toDecimal(replace(concat(toString(RevenueTargetPart1), toString(RevenueTarget)), '\\', ''), 10, 2, '\$###,###.##')</code>	Concatenate the RevenueTargetPart1 and RevenueTarget fields, replace the invalid \ character, then convert and format the data to a decimal type.

Derived column's settings Optimize Inspect Data preview

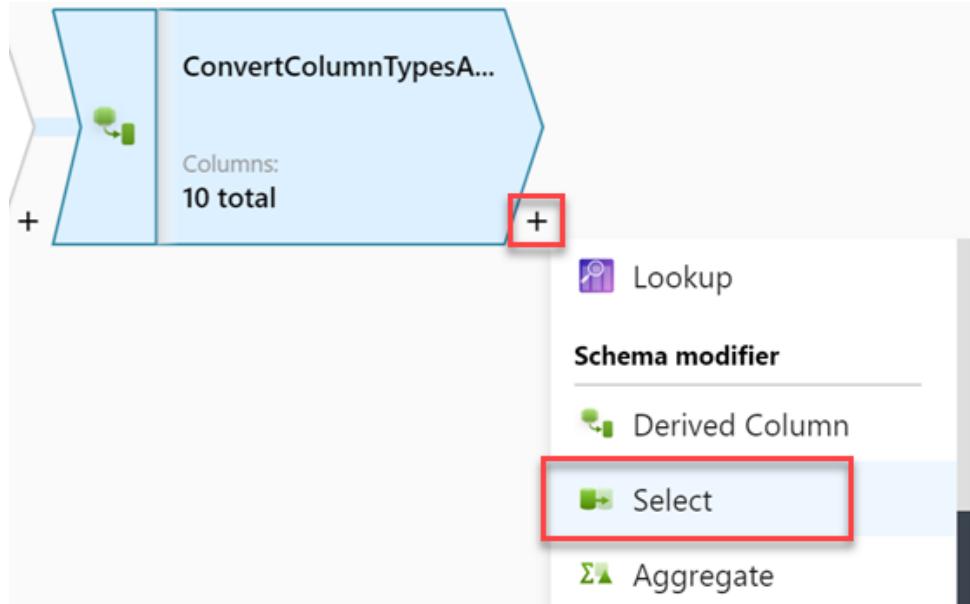
Output stream name * ConvertColumnTypeAndValues [Learn more](#)

Incoming stream * MapCampaignAnalytics

Columns * ① [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 * SelectCampaignAnalyticsColumns Learn more ↗

Incoming stream * ConvertColumnTypesAndValues

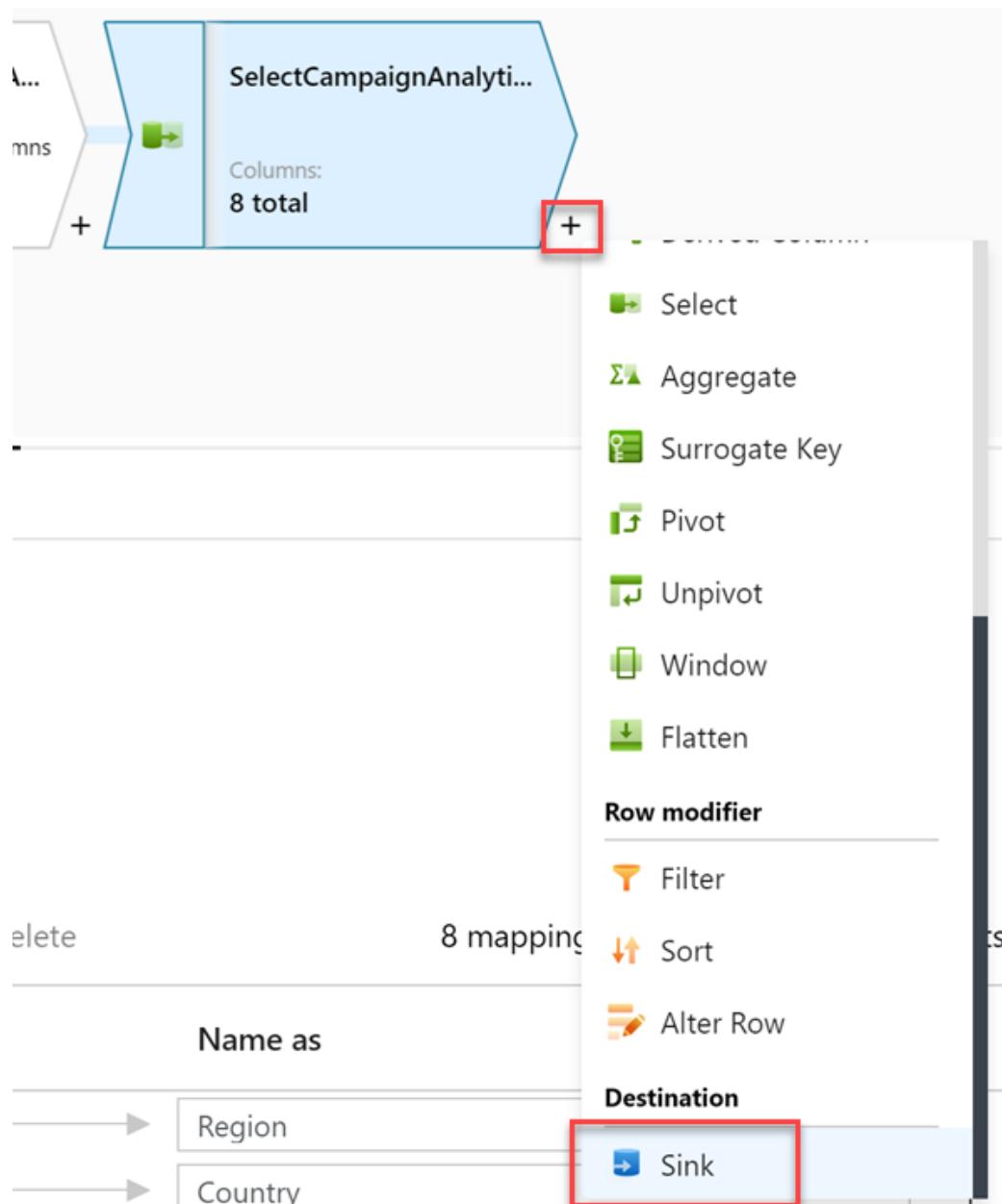
Options

- Skip duplicate input columns ⓘ
- Skip duplicate output columns ⓘ

Input columns * Auto mapping ⚡ 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 *	<input type="text" value="CampaignAnalyticsASA"/>	Learn more
Incoming stream *	<input type="text" value="SelectCampaignAnalyticsColumns"/>	
Dataset *	<input type="text" value="asal400_wwi_campaign_analytics_asa"/>	Test connection
Options	<input checked="" type="checkbox"/> Allow schema drift	
	<input type="checkbox"/> Validate schema	

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

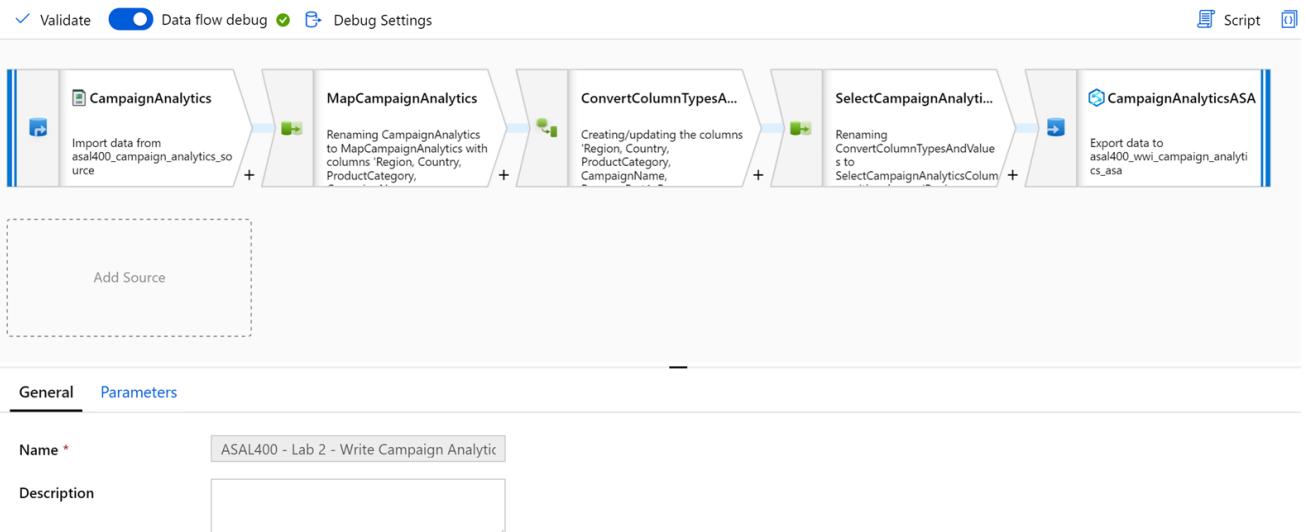
- **Update method:** Check `Allow insert` and leave the rest unchecked.
- **Table action:** Select `Truncate table`.
- **Enable staging:** Uncheck this option. The sample CSV file is small, making the staging option unnecessary.

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

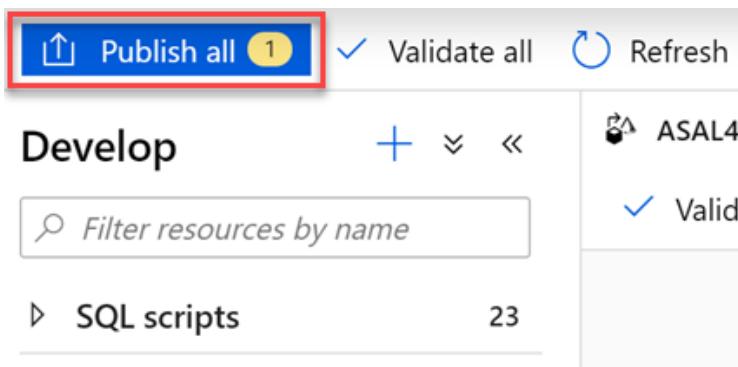
Tip: We recommend enabling staging to improve performance with Azure Synapse Analytics datasets.

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 type="checkbox"/>			
Batch size	<input type="text"/>			
Pre SQL scripts	<input type="text"/>			

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



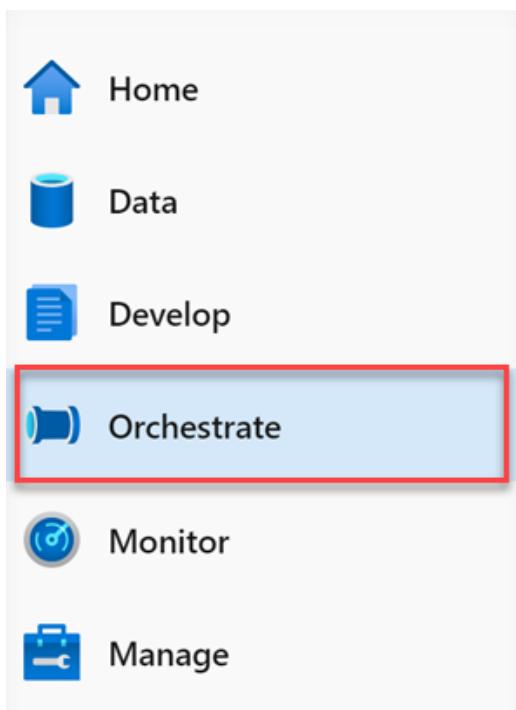
19. Select **Publish all** to save your new data flow.



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 **Orchestrate** hub.



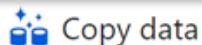
2. Select **+** then **Pipeline** to create a new pipeline.

Orchestrate



Pipeline

▶ Pipelines



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 'Activities' list in the Azure Data Factory interface. The 'Move & transform' category is expanded, showing the 'Data flow' activity. A red arrow points from the 'Data flow' activity in the list to a larger 'Data flow' icon on the pipeline canvas below. The pipeline canvas has a title bar with 'ASAL400 - Lab 2 - W...' and buttons for Validate, Debug, and Add trigger.

- Activities
- Validate
- Debug
- Add trigger

Search activities

- ▶ Synapse
- ▲ Move & transform
 - Copy data
 - Data flow**
- ▶ Azure Data Explorer
- ▶ Azure Function
- ▶ Batch Service

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. The 'Use existing data flow' radio button is selected. The 'Existing data flow' dropdown shows the option 'asal400_lab2_writecampaignanalyticstoasa'.

Adding data flow

Use existing data flow Create new data flow

Existing data flow *

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 **64 (+ 16 cores)** for the **Core count**.

The screenshot shows the 'Settings' tab of a 'Mapping Data Flow' pipeline named 'asal400_lab2_writecampaignanalyticstoasa'. The pipeline has two sink components: 'asal400_lab2_writecam' and 'paignanalyticstoasa'. Below the pipeline name, there are four buttons: a trash can, a brace {}, a copy icon, and a plus icon.

General Settings Parameters User properties

Data flow *: asal400_lab2_writecampaignanalytic... [Open](#)

Run on (Azure IR) *: AutoResolveIntegrationRuntime

Compute type *: Compute Optimized

Core count *: 64 (+ 16 Driver cores)

PolyBase ⓘ

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

The screenshot shows the 'Develop' blade. At the top, there are three buttons: 'Publish all' (highlighted with a red box), 'Validate all', and 'Refresh'. Below this, the 'Develop' section shows a '+' icon, a downward arrow, and a double-left arrow. To the right, it says 'ASAL4' and has a checked 'Valid' checkbox. A search bar says 'Filter resources by name' and a 'SQL scripts' section shows '23' items. A sidebar on the right lists resources under 'ASAL4'.

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.

The screenshot shows the toolbar at the top of the pipeline canvas. It includes buttons for 'Debug', 'Add trigger' (highlighted with a red box), a toggle switch for 'Data flow debug', and a dropdown menu with 'Trigger now' (highlighted with a red box) and 'New/Edit' options.

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

Pipeline run

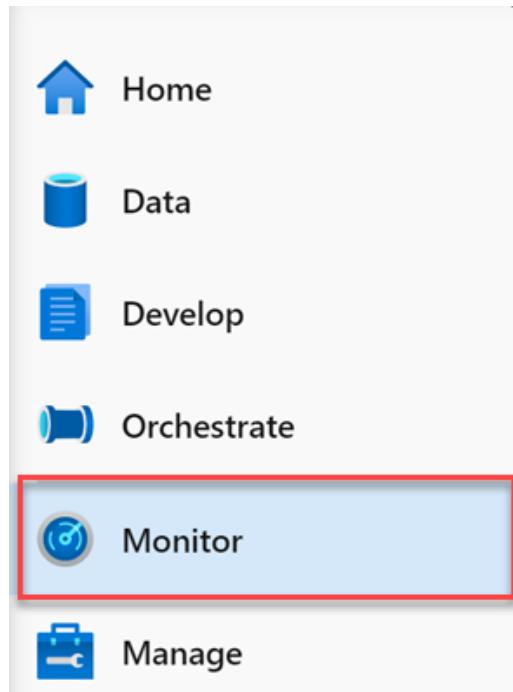
Trigger pipeline now using last published configuration.

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.

Pipeline runs

Time : Last 24 hours (8/13/20 12:14 AM - 8/14/20 12:14 AM) Time zone : Eastern Time (US & Canada) (UT...) Runs : Latest runs

All status **Refresh** Edit columns

Showing 1 - 6 items

Pipeline Name	Run Start	Duration	Triggered By	Status	Parameters
ASAL400 - Lab 2 - Write Cam...	8/14/20, 12:09:53 AM	00:04:37	Manual trigger	Succeeded	

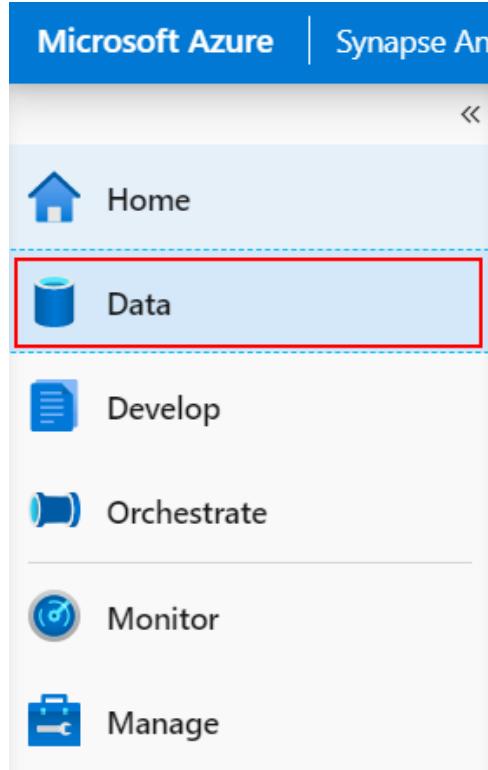
5. **Important:** if the pipeline run fails with **Internal Server Error:Failed to submit job on job cluster**. **Integration Runtime** or takes longer than **5 minutes** to complete, you are likely experiencing capacity-related issues.

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 1000 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.

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

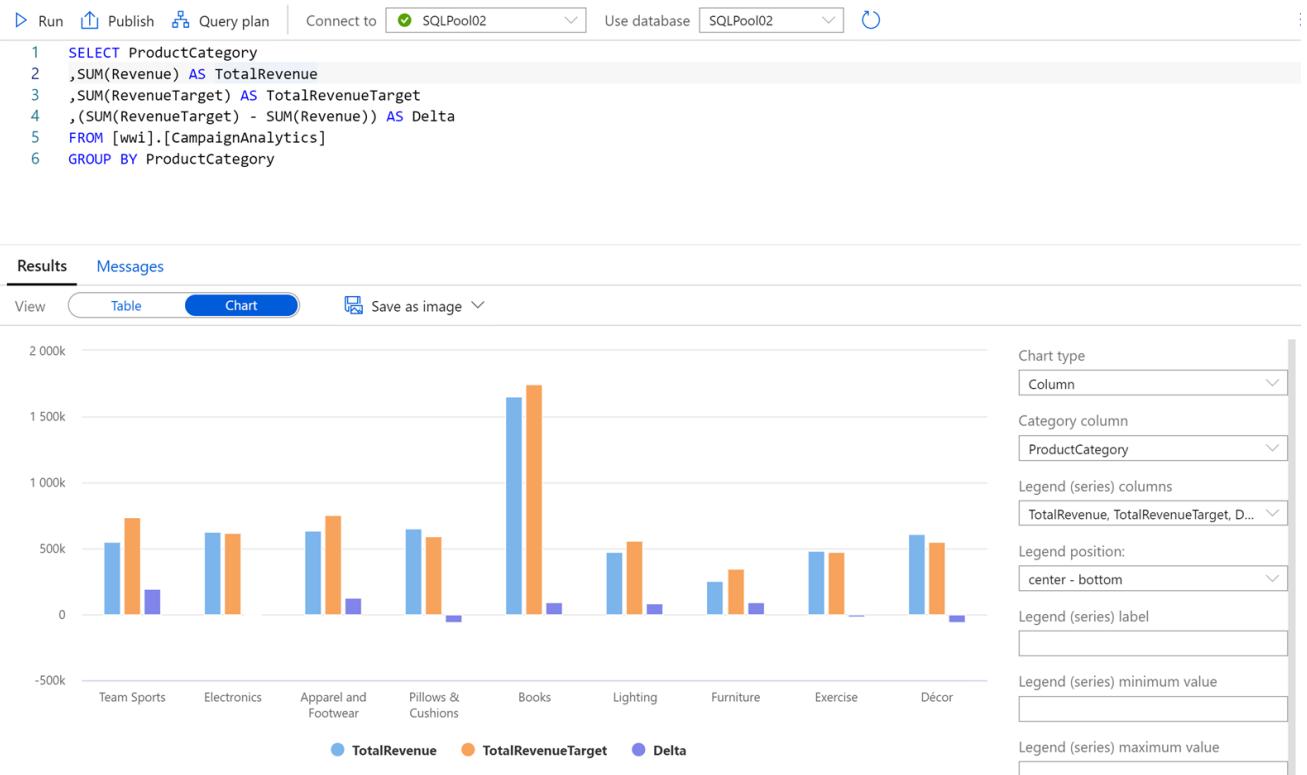
5. Update the query to the following and **Run**:

```
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:

```

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

```

Results Messages

View **Table** Chart [Export results](#)

Search

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

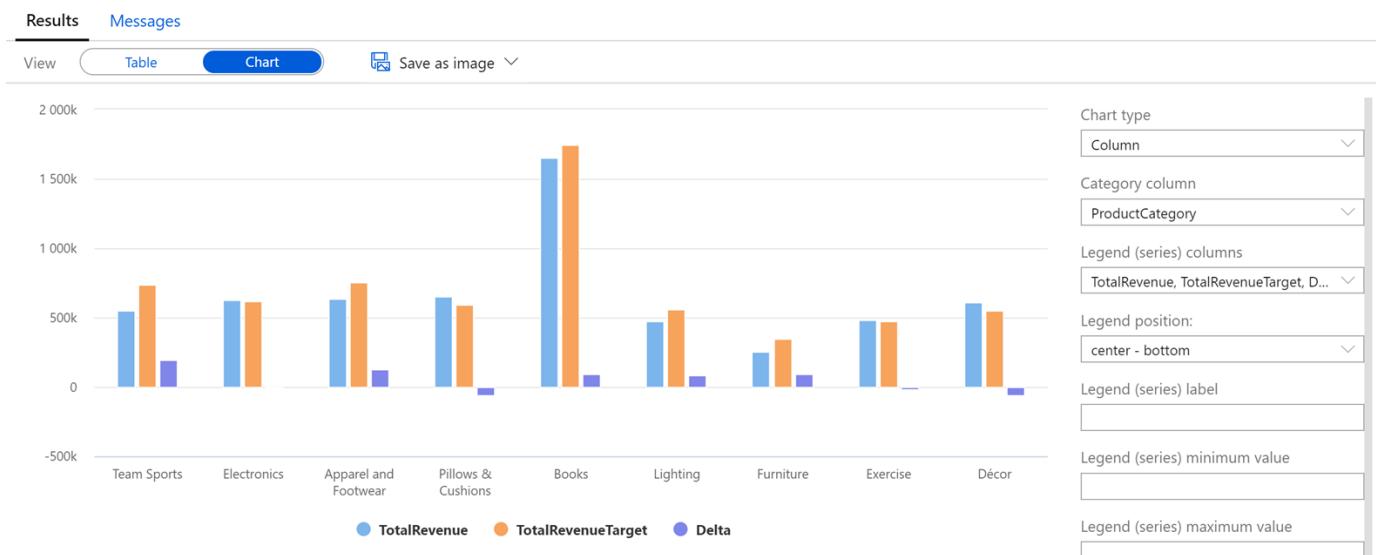
```

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:

```

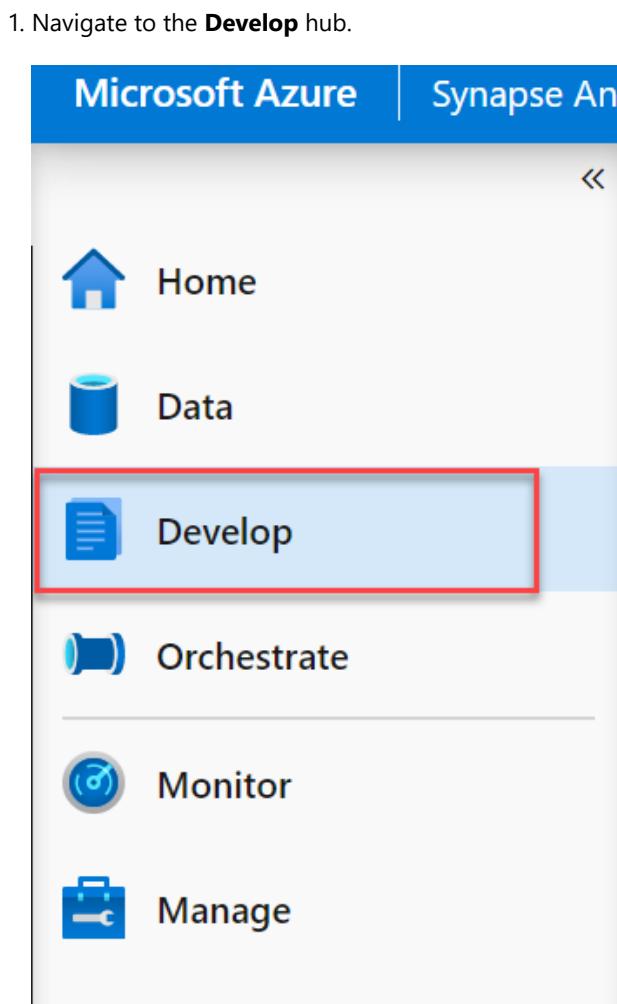
1 SELECT ProductCategory
2 ,SUM(Revenue) AS TotalRevenue
3 ,SUM(RevenueTarget) AS TotalRevenueTarget
4 ,(SUM(RevenueTarget) - SUM(Revenue)) AS Delta
5 FROM [wwi].[CampaignAnalytics]
6 GROUP BY ProductCategory

```

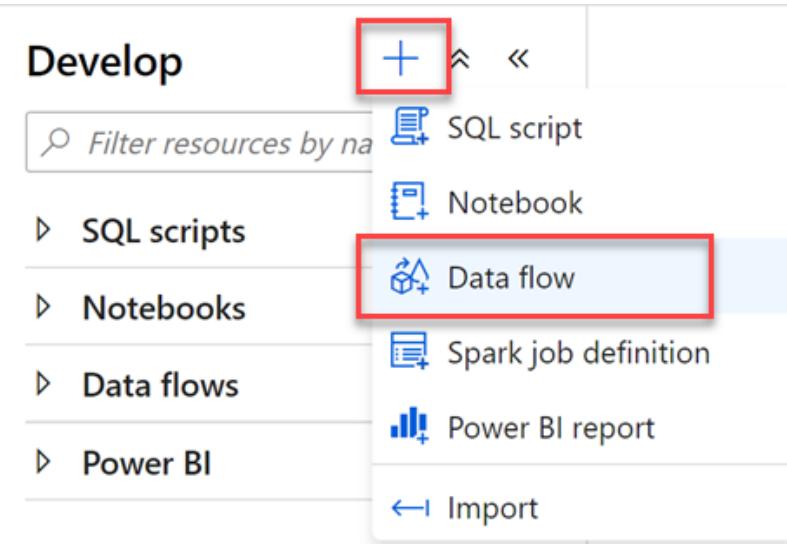


Exercise 3: Create data pipeline to join disparate data sources

Task 1: Create user profile data flow

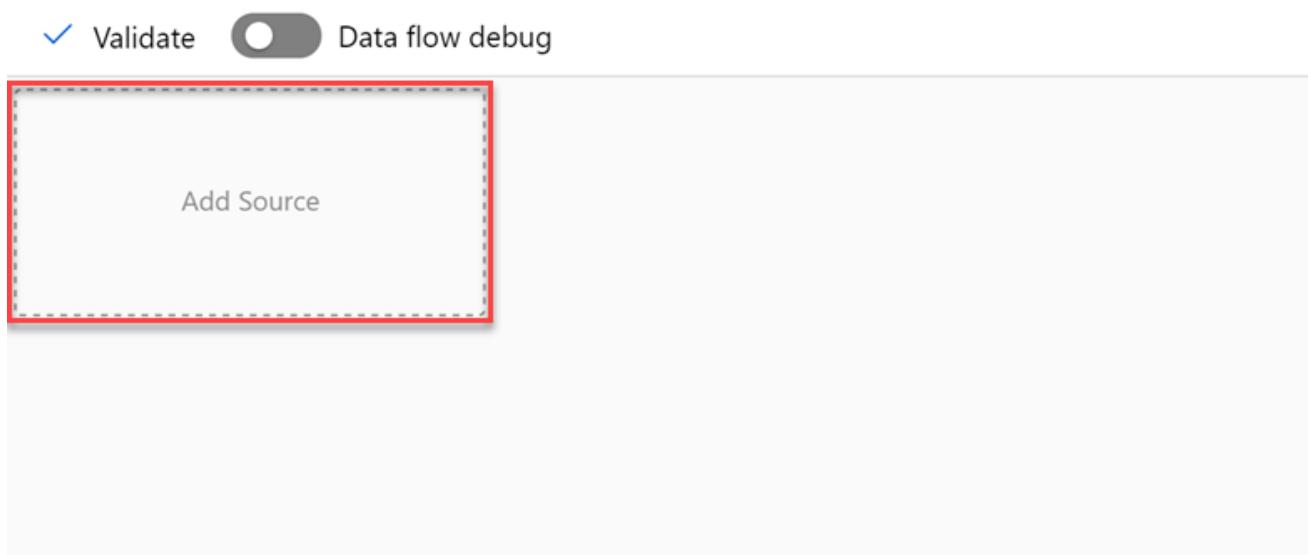


2. Select + then **Data flow** to create a new data flow.



3. In the **General** section of the **Profiles** pane of the new data flow, update the **Name** to the following:
asal400_lab2_writeuserprofiledataosa.

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



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

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

Source settings Source options Projection Optimize Inspect Data preview

Output stream name *	EcommerceUserProfiles	Learn more
Source type *	Dataset	
Dataset *	asal400_ecommerce_userprofiles_so...	Test connection
Options	<input checked="" type="checkbox"/> Allow schema drift ⓘ <input type="checkbox"/> Infer drifted column types ⓘ <input type="checkbox"/> Validate schema ⓘ	
Sampling *	<input type="radio"/> Enable <input checked="" type="radio"/> Disable <input type="radio"/>	

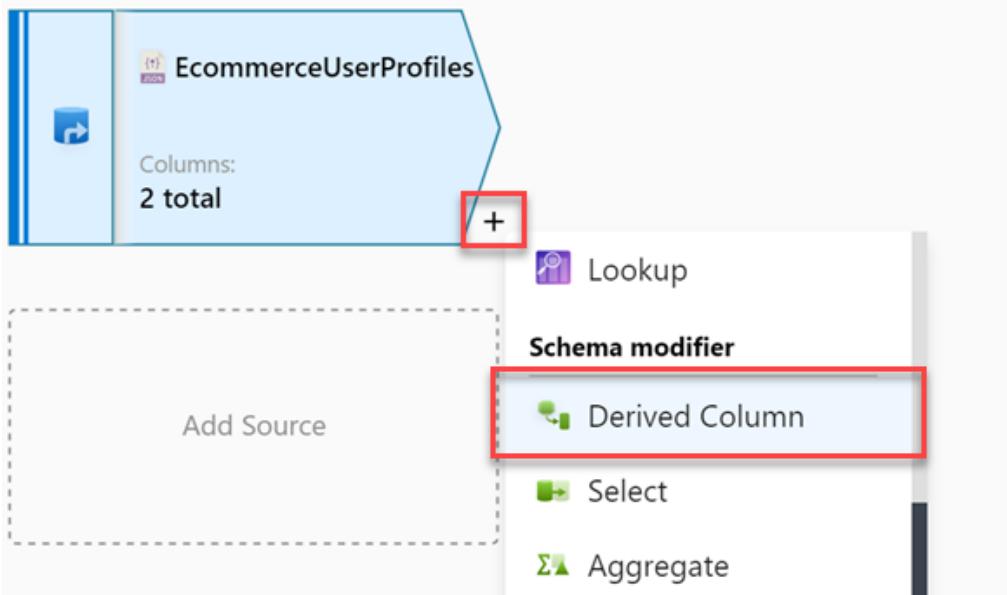
6. Select the **Source options** tab, then configure the following:

- **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	wwi-02 / <input type="text" value="online-user-profiles-02/*.json"/> ⓘ + - Delete	
Partition root path	<input type="text"/>	
List of files	<input type="checkbox"/> ⓘ	
Column to store file name	<input type="text"/> ⓘ	
After completion *	<input checked="" type="radio"/> No action <input type="radio"/> Delete source files <input type="radio"/> Move	
Filter by last modified	<input type="text"/> Start time (UTC)	<input type="text"/> End time (UTC)
▲ JSON settings		
Single document	<input checked="" type="checkbox"/> ⓘ	
Unquoted column names	<input type="checkbox"/> ⓘ	

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



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

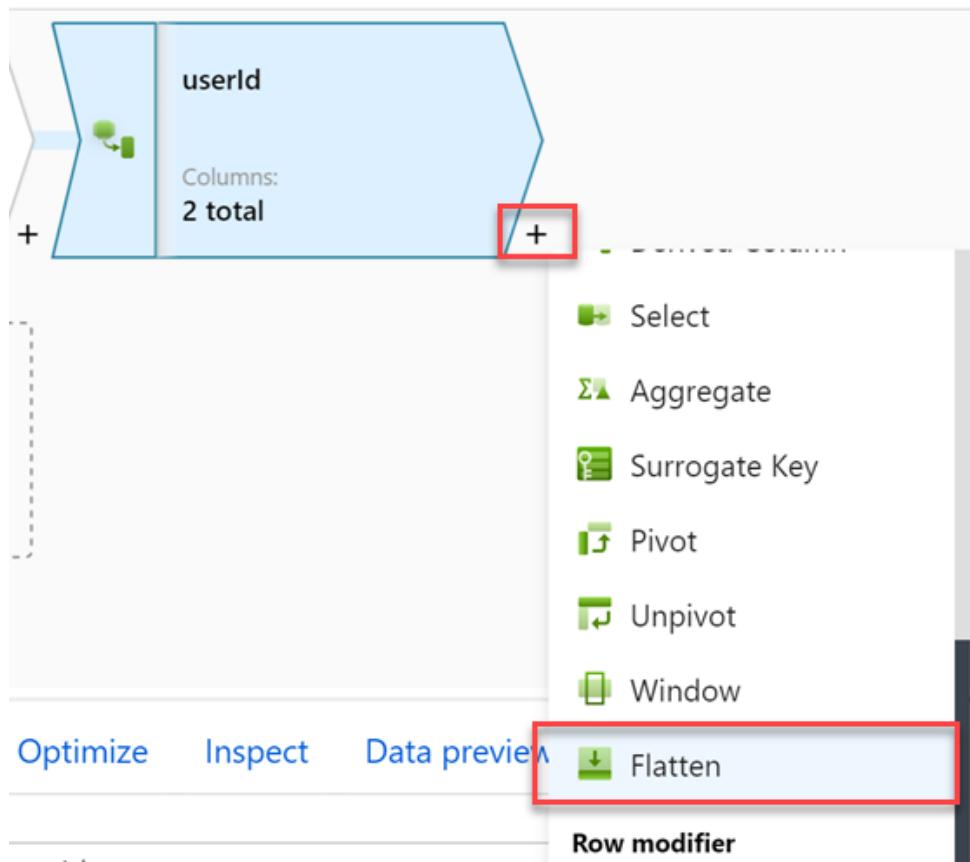
- **Output stream name:** Enter `userId`.
- **Incoming stream:** Select `EcommerceUserProfiles`.
- **Columns:** Provide the following information:

Column	Expression	Description
visitorId	<code>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 *	<input type="text" value="userId"/>	Learn more				
Incoming stream *	<input type="text" value="EcommerceUserProfiles"/>	▼				
Columns * ⓘ	Add Duplicate Delete					
	<table border="1"> <thead> <tr> <th>Column</th> <th>Expression</th> </tr> </thead> <tbody> <tr> <td><input type="text" value="visitorId"/></td> <td><input type="text" value="toInteger(visitorId)"/> 123</td> </tr> </tbody> </table>	Column	Expression	<input type="text" value="visitorId"/>	<input type="text" value="toInteger(visitorId)"/> 123	+ Duplicate Delete
Column	Expression					
<input type="text" value="visitorId"/>	<input type="text" value="toInteger(visitorId)"/> 123					

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



4. 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
visitorId	visitorId
topProductPurchases.productId	productId
topProductPurchases.itemsPurchasedLast12Months	itemsPurchasedLast12Months

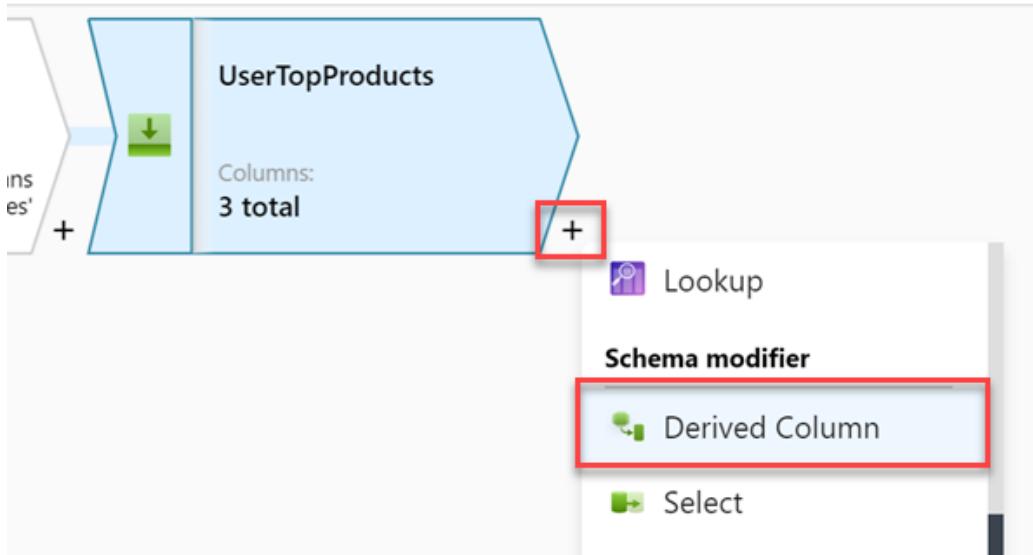
The screenshot shows the 'Flatten settings' configuration page. At the top, there are tabs: 'Flatten settings' (selected), 'Optimize', 'Inspect', and 'Data preview'. Below the tabs, there are fields for 'Output stream name' (set to 'UserTopProducts'), 'Incoming stream' (set to 'userId'), 'Unroll by' (set to '[] topProductPurchases'), and 'Unroll root' (empty). Under 'Options', there are two checkboxes: 'Skip duplicate input columns' and 'Skip duplicate output columns'. At the bottom, there is a section for 'Input columns' with a table mapping input columns to output names. The table has columns: 'userId's column', 'Name as', and 'Delete'. There are three rows: 'visitorId' mapped to 'visitorId', 'topProductPurchases.productId' mapped to 'productId', and 'topProductPurchases.itemsPurchasedLast12Months' mapped to 'itemsPurchasedLast12Months'. Each row has a red-bordered 'Delete' button.

userId's column	Name as	
visitorId	visitorId	
topProductPurchases.productId	productId	
topProductPurchases.itemsPurchasedLast12Months	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 123	productId abc	itemsPurchasedLast12Months abc
+	9611082	61	62
+	9611082	3003	20
+	9611082	2140	39
+	9611082	2918	62
+	9611082	4482	85
+	9611082	140	73
+	9611082	3892	33

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



6. 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	<code>toInteger(productId)</code>	Converts the <code>productId</code> column from a string to an integer.
itemsPurchasedLast12Months	<code>toInteger(itemsPurchasedLast12Months)</code>	Converts the <code>itemsPurchasedLast12Months</code> column from a string to an integer.

Derived column's settings Optimize Inspect Data preview

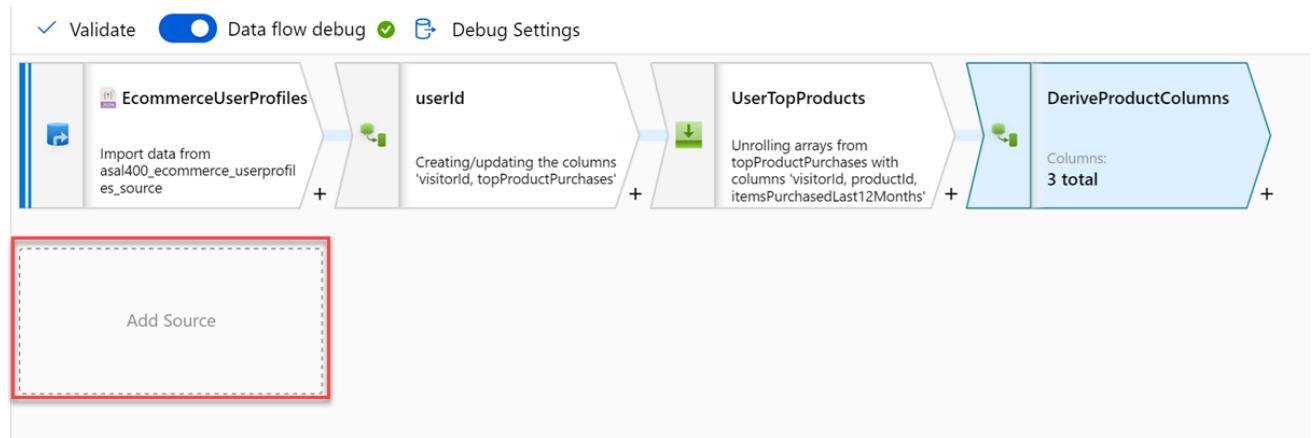
Output stream name * DeriveProductColumns Learn more

Incoming stream * UserTopProducts

Columns *

<input type="checkbox"/> Column	Expression
<input type="checkbox"/> productId	tolInteger(productId)
<input type="checkbox"/> itemsPurchasedLast12Months	tolInteger(itemsPurchasedLast12Months)

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



8. 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 * UserProfiles Learn more

Source type * Dataset

Dataset * asal400_customerprofile_cosmosdb Test connection

Options

Allow schema drift

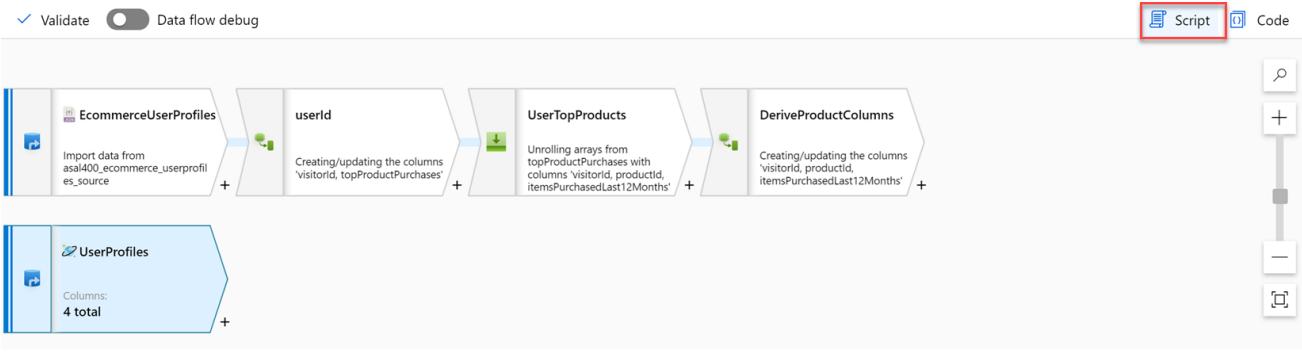
Infer drifted column types

Validate schema

Sampling *

Enable Disable

9. 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.



10. 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](#)

11. 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.*

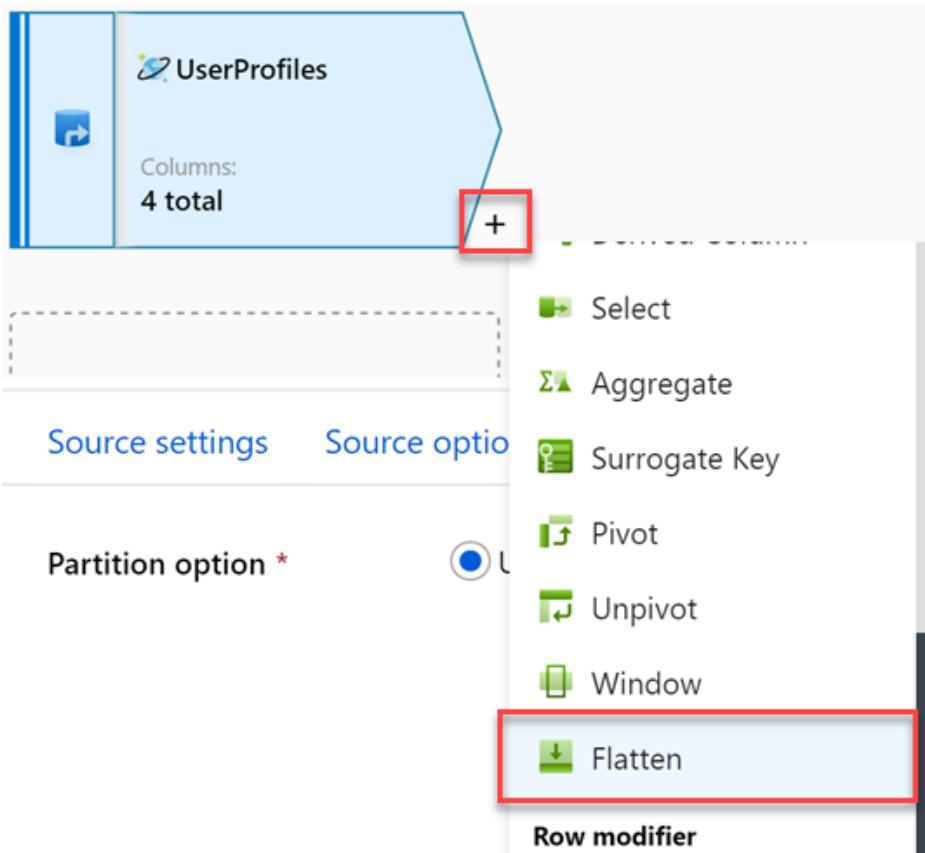
Source settings Source options Projection Optimize Inspect Data preview ●

Number of rows + INSERT 100 * UPDATE 0 X DELETE 0 *+ UPSERT 0 Q LOOKUP 0

Refresh Typecast Modify Map drifted Statistics Remove

↑↓	cartId abc	preferredProducts []	userId 123
+	406a06af-e54f-42e9-aad8-9a36f2c7f8ca	[...]	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

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



13. 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
userId	userId
[] preferredProducts	preferredProductId

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](#) 2 mappings: 2 column(s) from the inputs left ↗

UserProfiles's column	Name as
<input type="text" value="abc userId"/> ↗ ⓘ	<input type="text" value="userId"/> ↗ + ⌂
<input type="text" value="[] preferredProducts"/> ↗ ⓘ	<input type="text" value="preferredProductId"/> ↗ + ⌂

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

- 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.



- Under **Join settings**, configure the following:

- Output stream name**: Enter `JoinTopProductsWithPreferredProducts`.
- Left stream**: Select `DeriveProductColumns`.

- o **Right stream:** Select `UserPreferredProducts`.
- o **Join type:** Select `Full outer`.
- o **Join conditions:** Provide the following information:

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

<code>visitorId</code>	<code>userId</code>
------------------------	---------------------

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**

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

- o **Broadcast:** Select `Fixed`.
- o **Broadcast options:** Check `Left: 'DeriveProductColumns'`.
- o **Partition option:** Select `Set partitioning`.
- o **Partition type:** Select `Hash`.
- o **Number of partitions:** Enter `30`.
- o **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 *

Number of partitions *

Column values to hash on * **Column**

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

Number of columns		Left: DeriveProductColumns	Right: UserPreferredProducts	
Order ↑↓	Column ↑↓	Type ↑↓	Fed by ↑↓	Used in Join ↑↓
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.

The screenshot shows the Data Preview pane with a table of data and a pie chart on the right. The table has columns: productId, itemsPurchasedLast12Months, userId, and preferredProductId. Most values in the userId and preferredProductId columns are NULL. A red arrow points to the 'Statistics' button in the toolbar above the table.

Number of rows: 100 | * UPDATE 0 | X DELETE 0 | + UPSERT 0 | Q LOOKUP 0 | TOTAL 15747

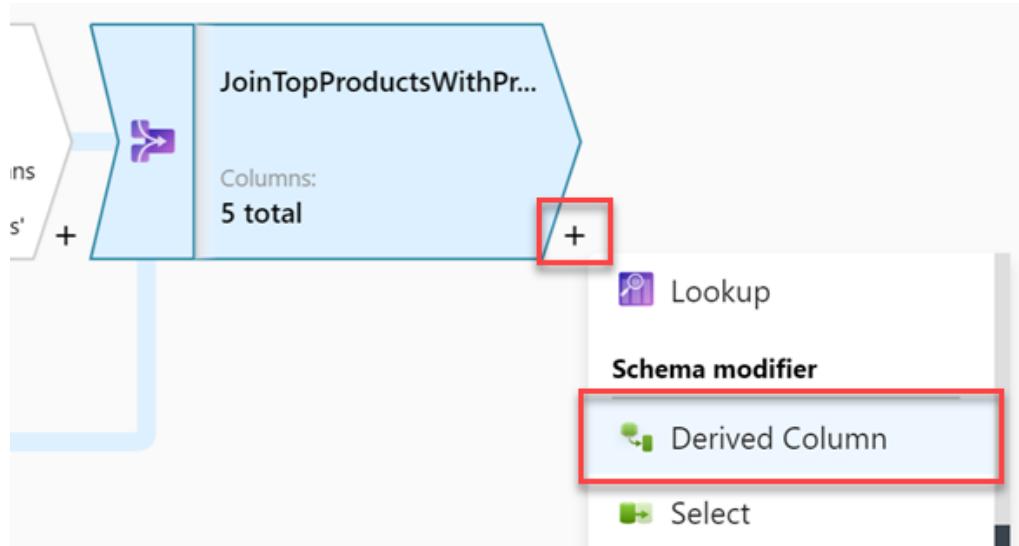
productId	itemsPurchasedLast12Months	userId	preferredProductId
2081	94	NULL	NULL
2146	21	NULL	NULL
968	7	NULL	NULL
4630	35	NULL	NULL
3273	97	NULL	NULL
1269	4	NULL	NULL
371	50	NULL	NULL

preferredProductId 123

Not Null Null

A pie chart shows that approximately 25% of the values are 'Not Null' (dark blue) and 75% are 'Null' (light blue).

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



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

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

Column	Expression	Description
--------	------------	-------------

Column	Expression	Description
isTopProduct	toBoolean(iif(isNull(productId), 'false', 'true'))	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.
isPreferredProduct	toBoolean(iif(isNull(preferredProductId), 'false', 'true'))	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.
productId	iif(isNull(productId), preferredProductId, productId)	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.
userId	iif(isNull(userId), visitorId, userId)	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 * [Learn more](#)

Incoming stream *

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

Column	Expression
isTopProduct	toBoolean(iif(isNull(productId), 'false', 'true'))
isPreferredProduct	toBoolean(iif(isNull(preferredProductId), 'false', 'true'))
productId	iif(isNull(productId), preferredProductId, productId) 123
userId	iif(isNull(userId), visitorId, userId) 123

The derived column settings provide the following result:

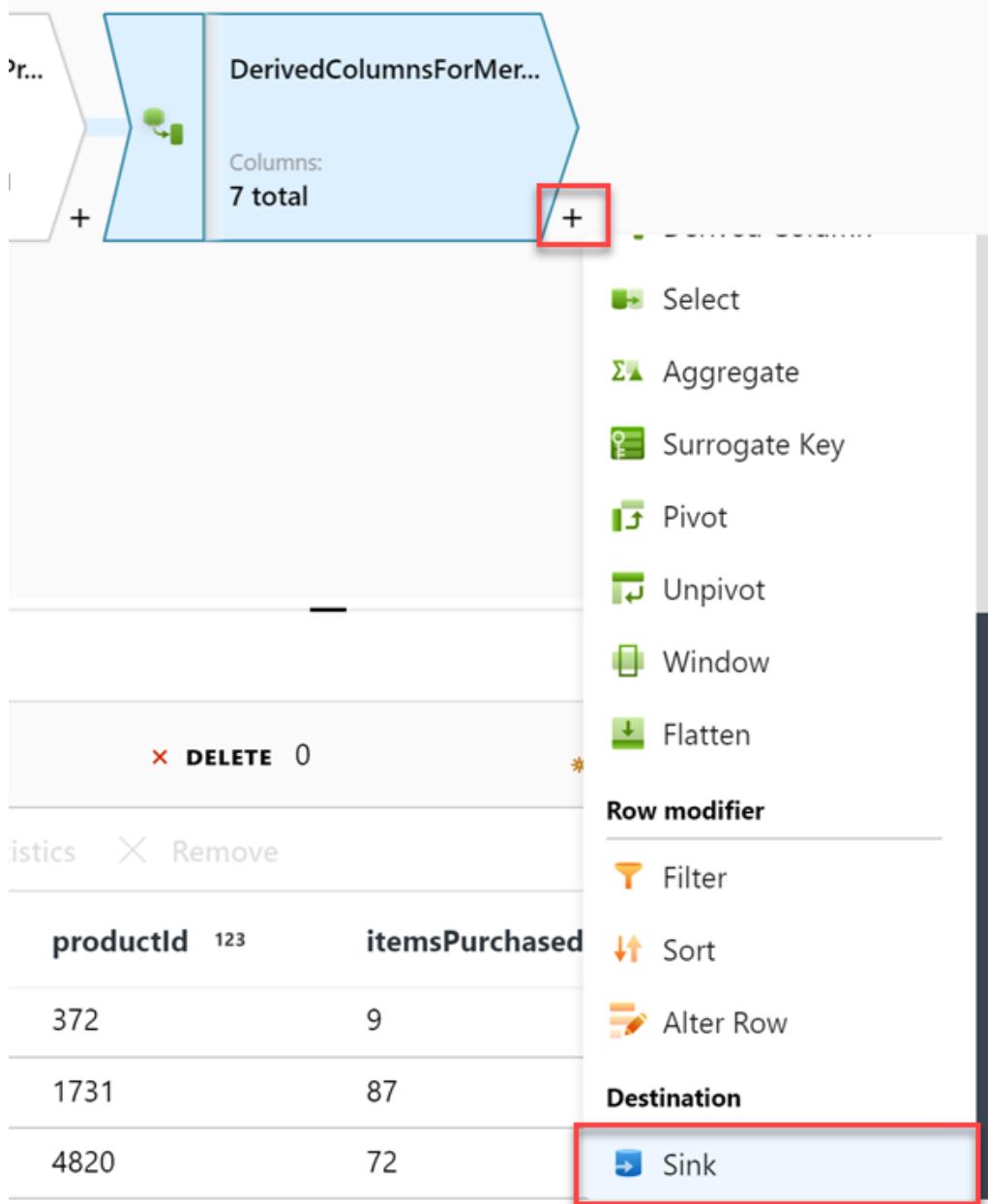
Derived column's settings [Optimize](#) [Inspect](#) [Data preview](#) ● [Description](#)

Number of rows [+ INSERT](#) 100 [* UPDATE](#) 0 [✖ DELETE](#) 0 [* UPsert](#) 0 [LOOKUP](#) 0 **TOTAL** 15361

[⟳ Refresh](#) [Typecast](#) [Modify](#) [Map drifted](#) [Statistics](#) [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	✓	✗

20. Select the + to the right of the `DerivedColumnsForMerge` step, then select the `Sink` destination from the context menu.



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

- **Output stream name:** Enter `UserTopProductPurchasesASA`.
- **Incoming stream:** Select `DerivedColumnsForMerge`.
- **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 * DerivedColumnsForMerge

Sink type * Dataset

Dataset * asal400_wwi_usertopproductpurchas... [Test connection](#)

Options

- Allow schema drift ⓘ
- Validate schema ⓘ

22. 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

- Allow insert
- Allow delete
- Allow upsert
- Allow update

Table action

- None
- Recreate table
- Truncate table

Enable staging

Batch size ⓘ

Pre SQL scripts

23. 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

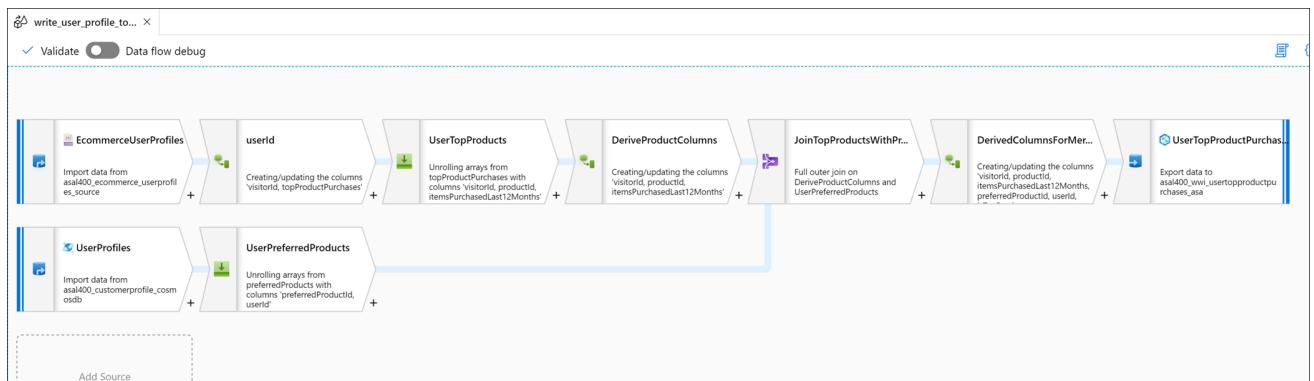
Sink Settings Mapping Optimize Inspect Data preview Description ▾

Options Skip duplicate input columns ⓘ Skip duplicate output columns ⓘ

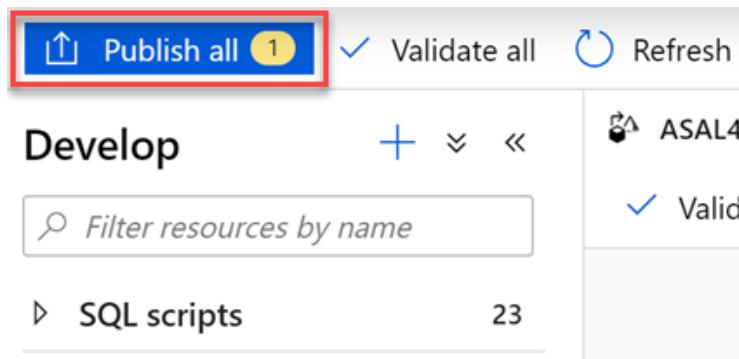
Auto mapping ⓘ 5 mappings: All outputs mapped

Input columns	Output columns
12s userId	12s Userid
12s productId	12s ProductId
12s itemsPurchasedLast12Months	12s ItemsPurchasedLast12Months
Kv isTopProduct	Kv IsTopProduct
Kv isPreferredProduct	Kv IsPreferredProduct

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



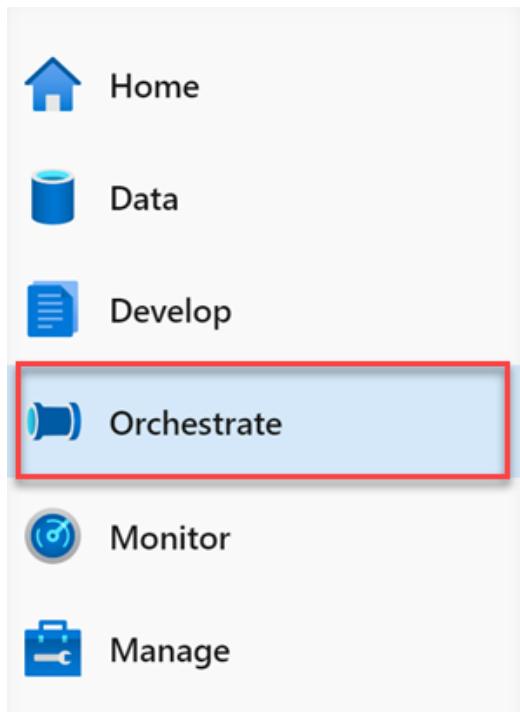
25. Select **Publish all** to save your new data flow.



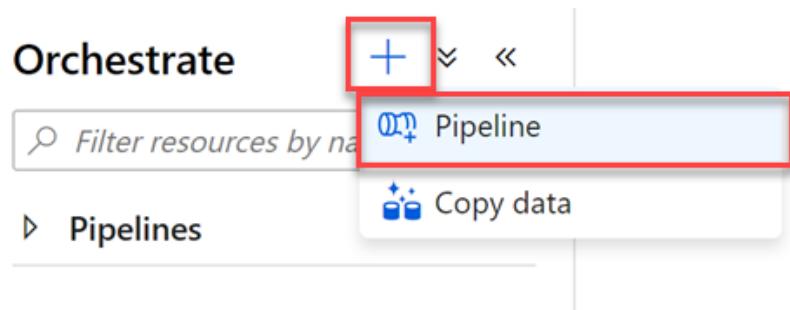
Task 2: Create user profile data pipeline

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 **Orchestrate** hub.

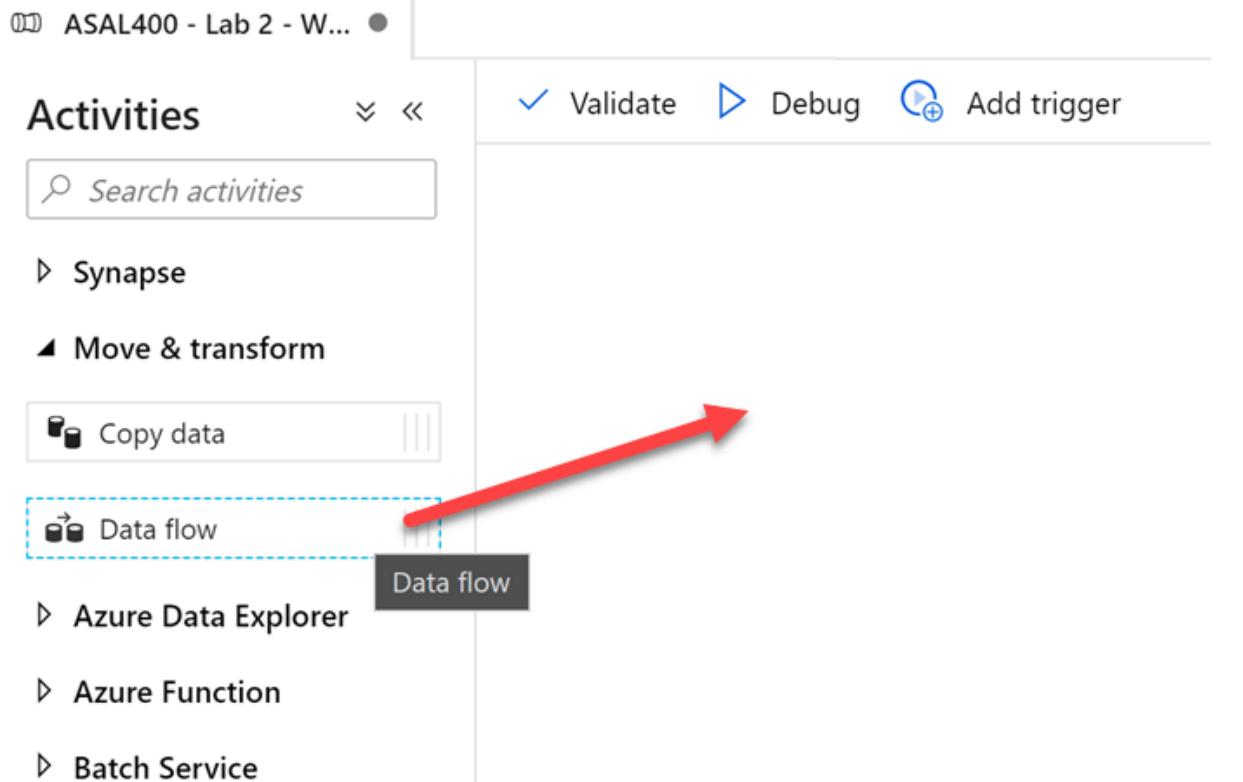


2. Select + then **Pipeline** to create a new pipeline.

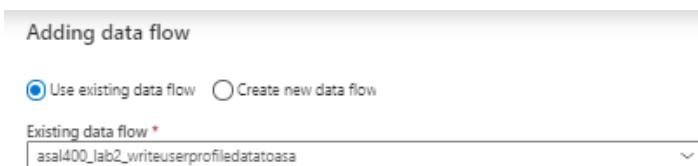


3. In the **General** section of the **Properties** blade for the new pipeline, enter the following **Name:** ASAL400 - Lab 2 - Write User Profile Data to ASA.

4. Expand **Move & transform** within the Activities list, then drag the **Data flow** activity onto the pipeline canvas.



5. In the **Adding data flow** blade, select **Use existing data flow**, then select the `asal400_lab2_writeuserprofiledataoasa` existing data flow you created in the previous task.



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 `64 (+ 16 cores)` for the **Core count**.

8. Expand **PolyBase** and configure the following:

- **Staging linked service:** Select the `asadatalakeSUFFIX` linked service.
- **Staging storage folder:** Enter `staging/userprofiles`. The `userprofiles` folder will be automatically created for you during the first pipeline run.

The screenshot shows the 'General' tab of the pipeline settings. At the top, there's a preview window titled 'Mapping Data Flow' showing a single activity named 'asal400_lab2_writeuserprofiledatatoasa'. Below the tabs, there are several configuration fields:

- Data flow ***: A dropdown set to 'asal400_lab2_writeuserprofiledatatoasa' with 'Open' and 'New' buttons.
- Run on (Azure IR) ***: A dropdown set to 'AutoResolveIntegrationRuntime'.
- Compute type ***: A dropdown set to 'Compute Optimized'.
- Core count ***: A dropdown set to '64 (+ 16 Driver cores)'.
- PolyBase**: A collapsed section with a 'Test connection' button and a 'Browse' button.
- Staging linked service**: A dropdown set to 'asadatalake212045' with a 'Test connection' button and a 'Browse' button.
- Staging storage folder**: A dropdown set to 'staging / userprofiles' with a 'Browse' button.

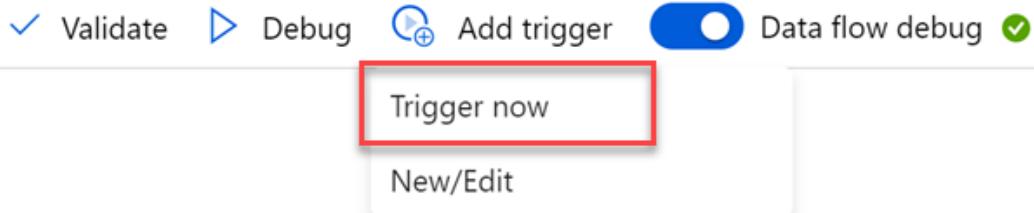
9. Select **Publish all** to save your new pipeline.

The screenshot shows the pipeline overview page. At the top, there are three buttons: 'Publish all' (highlighted with a red box), 'Validate all', and 'Refresh'. Below this, the pipeline name 'ASAL4' is shown with a status of 'Valid'. On the left, there's a sidebar with sections for 'Develop' (containing 'SQL scripts' with 23 items) and 'Filter resources by name'.

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

1. Select **Add trigger** and 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.

A screenshot of the 'Pipeline run' blade. At the top, there is a message: '⚠ Trigger pipeline now using last published configuration.' Below this is a section titled 'Parameters' with a table header: 'NAME', 'TYPE', 'VALUE'. A message 'No records found' is displayed below the table. At the bottom of the blade are two buttons: a blue 'OK' button on the left and a white 'Cancel' button on the right.

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

A screenshot of the 'Monitor' hub navigation menu. It lists several items with corresponding icons: 'Home' (house), 'Data' (cylinder), 'Develop' (document), 'Orchestrate' (gears), 'Monitor' (target), and 'Manage' (gear wrench). The 'Monitor' item is highlighted with a red rectangular border.

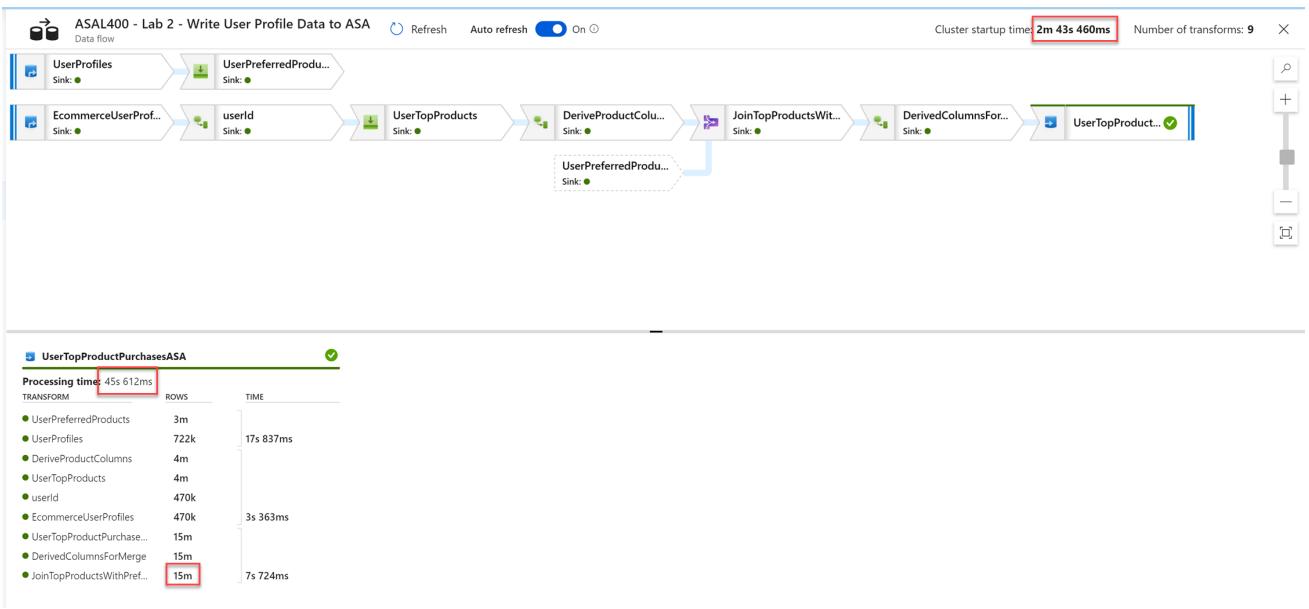
4. Wait for the pipeline run to successfully complete. You may need to refresh the view.

The screenshot shows the 'Pipeline runs' section of the Azure Data Factory interface. On the left, there's a sidebar with icons for Orchestration, Pipeline runs, Trigger runs, Integration runtimes, Activities, Apache Spark applications, and SQL requests. The main area is titled 'Pipeline runs' and shows a table of runs. A single row is visible for 'ASAL400 - Lab 2 - Write User ...', which started at 8/14/20, 1:21:33 AM and completed at 00:04:53. The status is 'Succeeded' (indicated by a green checkmark icon). A red box highlights the 'Refresh' button at the top right of the table header.

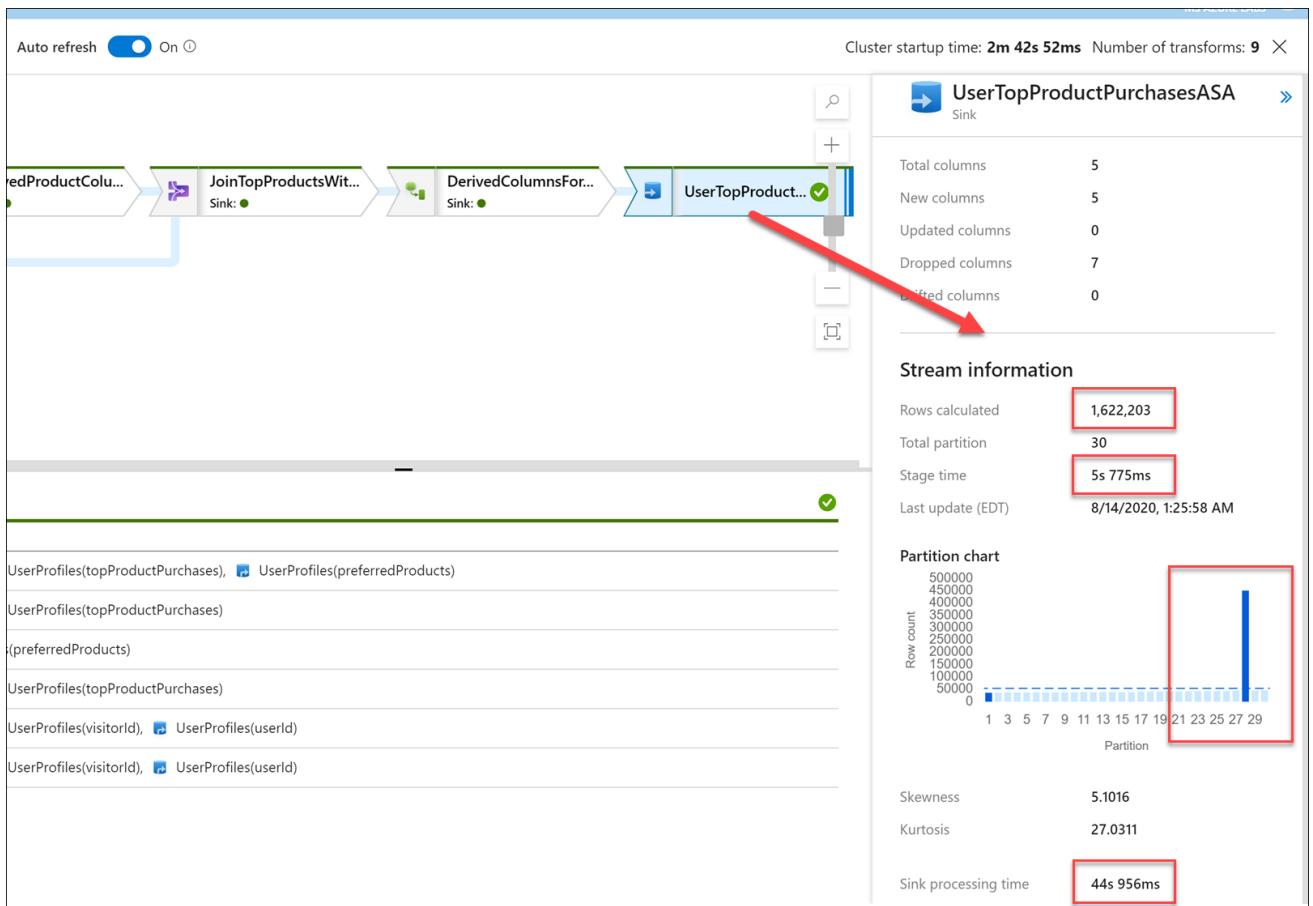
5. Select the name of the pipeline to view the pipeline's activity runs. Hover over the data flow activity name in the **Activity runs** list, then select the **Data flow details** icon.

The screenshot shows the 'Activity runs' page for the pipeline 'ASAL400 - Lab 2 - Write User Profile Data to ASA'. It includes tabs for 'List' and 'Gantt'. Below the tabs are buttons for 'Rerun', 'Rerun from activity', 'Rerun from failed activity', and 'Refresh'. A modal window titled 'Mapping Data Flow' is open, showing a single step named 'asal400_lab2_writeuserprofiledatatoasa'. In the main list, there is one item: 'Pipeline run ID 3e13d221-4aaa-449d-9f76-3d1e05036ee5'. Underneath, it says 'Showing 1 - 1 of 1 items'. The table has columns: ACTIVITY NAME, ACTIVITY TYPE, RUN START, DURATION, STATUS, and INTEGRATION RUNTIME. One row is listed: 'asal400_lab2_... ExecuteDataFlow 8/14/20, 1:21:35 AM 00:04:51 Succeeded AutoResolveIntegrationRuntime (West US 2)'. A red box highlights the 'Data flow details' icon (a blue square with a white gear symbol) next to the activity name in the table row.

6. The data flow details displays the data flow steps and processing details. In our example, processing time took around 45 seconds to process and output around 15 million rows. You can see which activities took longest to complete. The cluster startup time contributed almost three minutes to the total pipeline run.



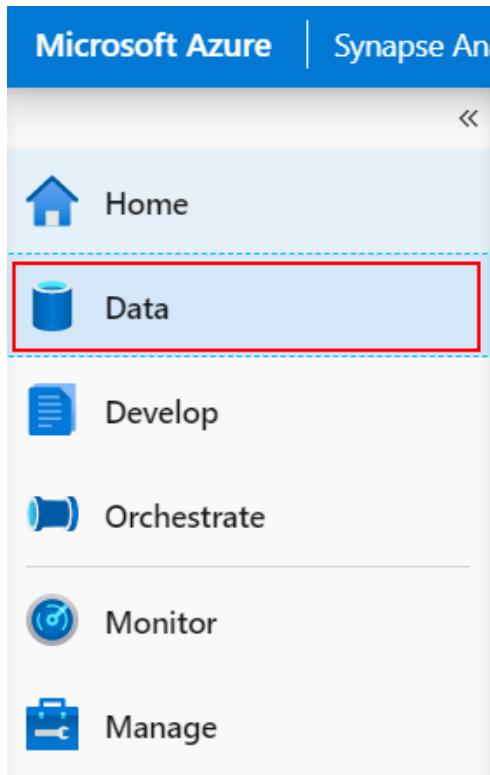
7. Select the **UserTopProductPurchasesASA** sink to view its details. We can see that 1,622,203 rows were calculated with a total of 30 partitions. It took around six seconds to stage the data in ADLS Gen2 prior to writing the data to the SQL table. The total sink processing time in our case was around 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.



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. On the left, the 'Databases' section is expanded, showing '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 highlighted with a red box. A context menu is open over this table, with the 'Load to DataFrame' option also highlighted with a red box. The right side of the interface shows the 'Activities' pane with various options like Synapse, Move & transform, and Azure Function.

3. Attach the notebook to a Spark pool.

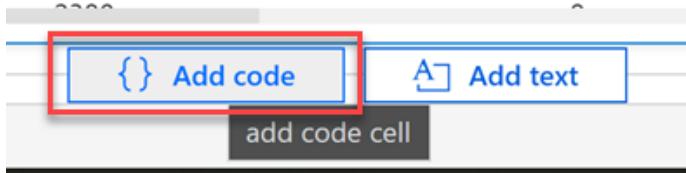
The screenshot shows the Azure Data Studio notebook interface. The toolbar at the top includes 'Cell', 'Run all', 'Undo', 'Publish', 'Attach to', and language settings ('Spark (Scala)'). A dropdown menu labeled 'Select Spark pool' is open, listing several Spark pools: SparkPool01, SparkPool02, SparkPool03, SparkPool04, SparkPool05, SparkPool06, and 'Manage pools'. The 'Select Spark pool' dropdown is highlighted with a red box.

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:

```
res4: Array[org.apache.spark.sql.Row] = Array([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])
```

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 both `IsTopProduct` and `IsPreferredProduct` are true:

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

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

topPreferredProducts.show(100)
```

Cell 4

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

Command executed in 7s 788ms by joel on 04-20-2020 15:21:55.224 -04:00

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

UserId	ProductId	ItemsPurchasedLast12Months	IsTopProduct	IsPreferredProduct
9596264	4431	99	true	true
9628230	4690	99	true	true
9596264	4431	99	true	true
9596264	4431	99	true	true
9596264	4431	99	true	true
9596264	4431	99	true	true
9596264	4431	99	true	true
9596264	4431	99	true	true
9578837	3814	99	true	true
9578837	3814	99	true	true
9578837	3814	99	true	true
9578837	3814	99	true	true
9578837	3814	99	true	true
9578837	3814	99	true	true
9578837	3814	99	true	true
9578837	3814	99	true	true
9591502	3813	99	true	true
9591502	3812	99	true	true

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 and IsPreferredProduct = 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.

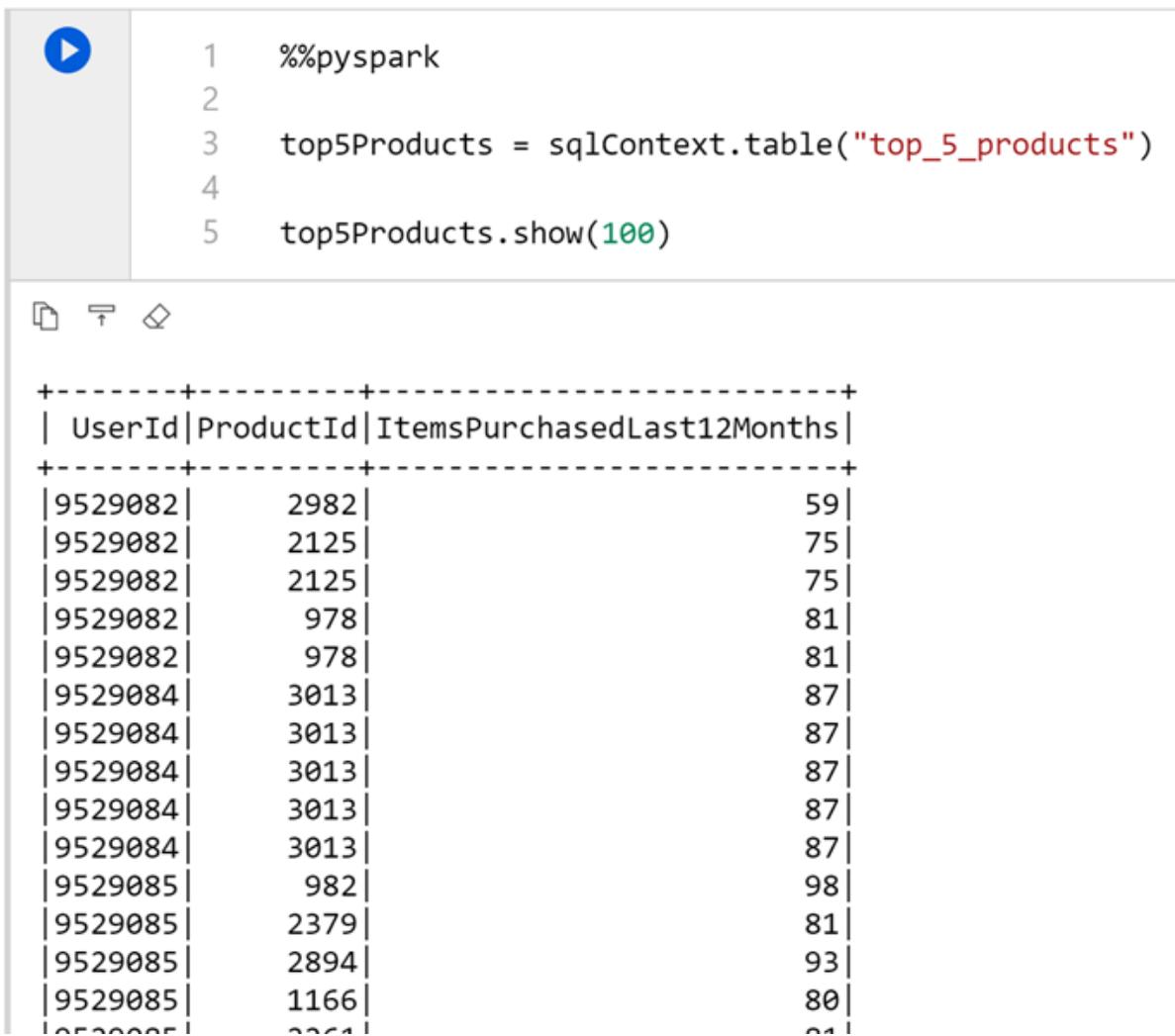
10. Execute the following in a new cell to create and display a new DataFrame that stores the results of the `top_5_products` temporary view you created in the previous cell:

```
%%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:



The screenshot shows a Jupyter Notebook cell with the following content:

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

Below the code, the output is displayed as a table:

User Id	Product Id	Items Purchased Last 12 Months
9529082	2982	59
9529082	2125	75
9529082	2125	75
9529082	978	81
9529082	978	81
9529084	3013	87
9529084	3013	87
9529084	3013	87
9529084	3013	87
9529084	3013	87
9529085	982	98
9529085	2379	81
9529085	2894	93
9529085	1166	80
9529085	2261	81

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: 9662384 , after filter: 822044`.

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	4538
4833	4533
347	4523
3459	4233
4246	4155

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 ✓
 asal400_lab2_writeuserprofiledatatoasa

+ - [0%] []

Activity runs

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

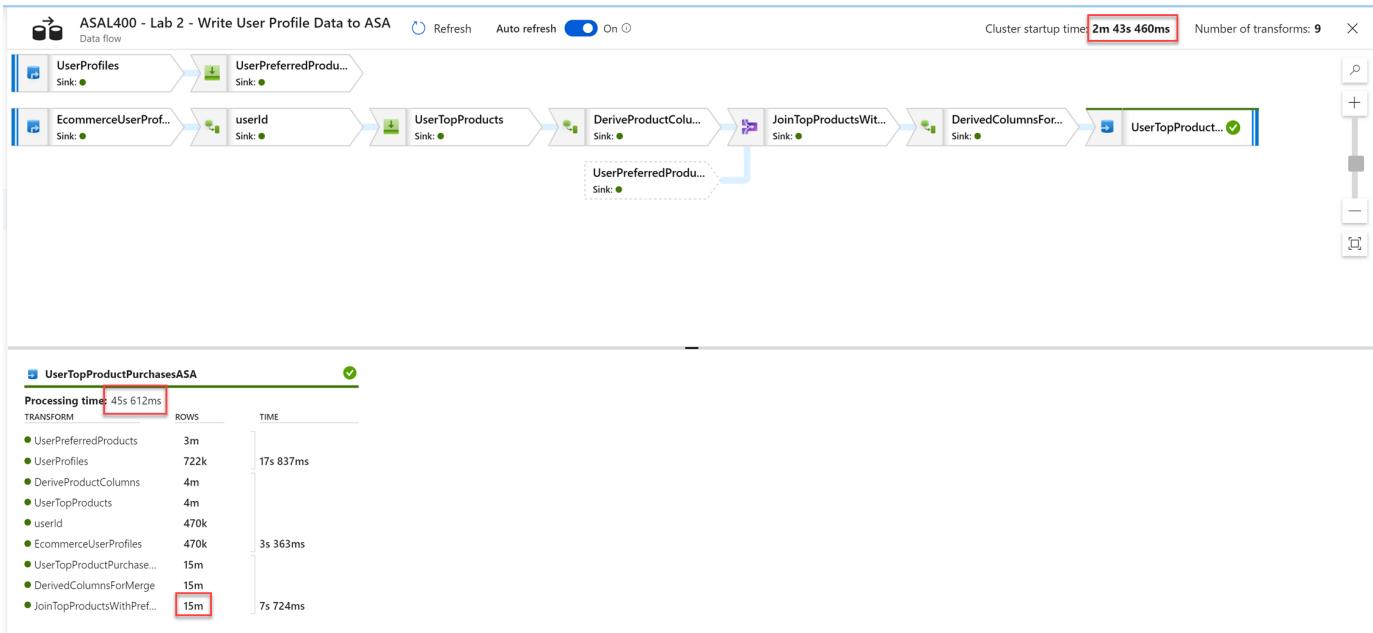
All status ▾

Showing 1 - 1 of 1 items

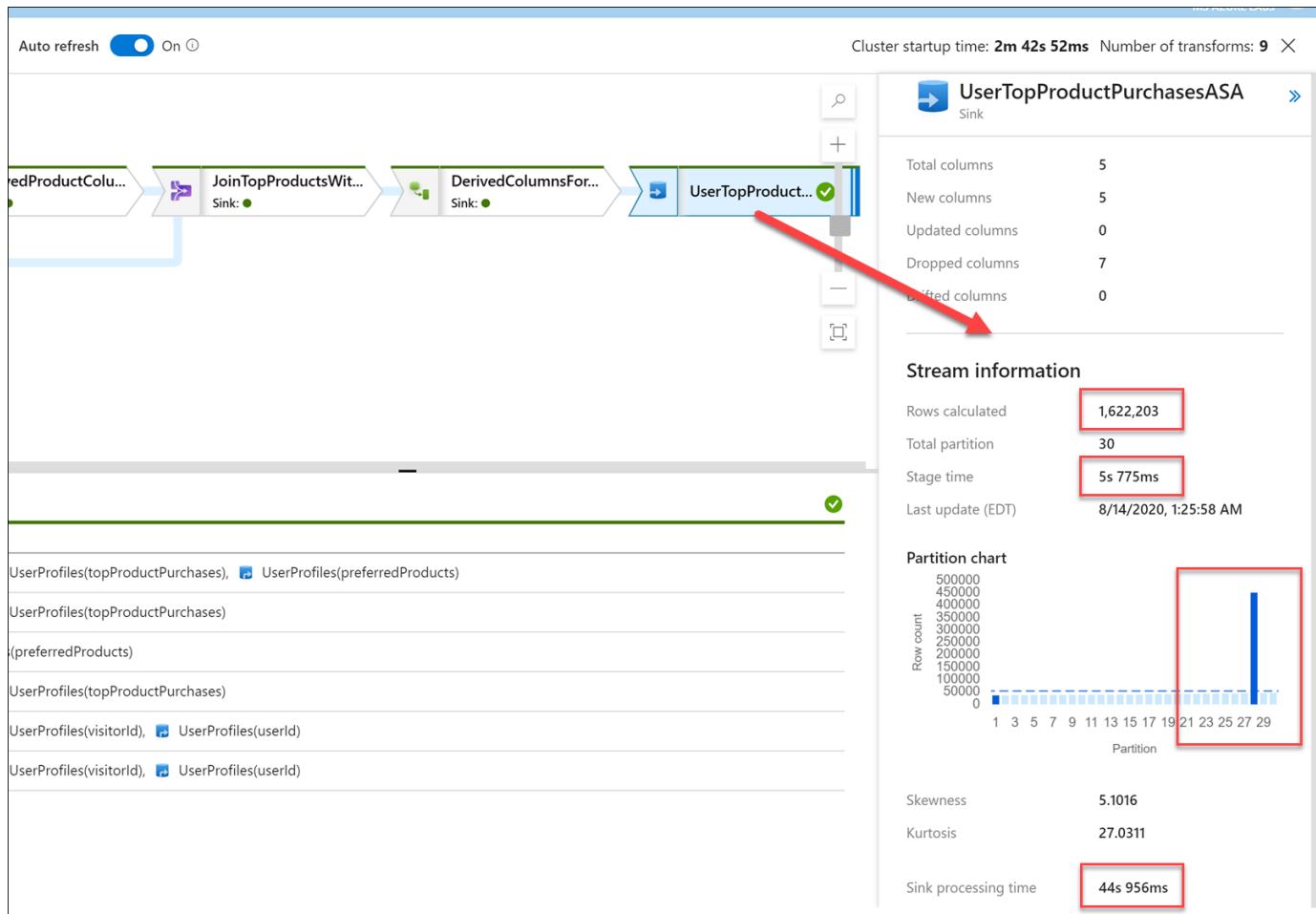
ACTIVITY NAME	ACTIVITY TYPE	RUN START ↑	DURATION	STATUS	INTEGRATION RUNTIME
asal400_lab2_...	ExecuteDataFlow	8/14/20, 1:21:35 AM	00:04:51	✓ Succeeded	AutoResolveIntegrationRuntime (West US 2)

[Data flow details](#)

The data flow details displays the data flow steps and processing details. In our example, processing time took around 45 seconds to process and output around 15 million rows. You can see which activities took longest to complete. The cluster startup time contributed almost three 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("SQLPool02.wwi.UserTopProductPurchases")
df.head(10)

df.createTempView("df")
```

The output looks like the following:

```
df: org.apache.spark.sql.DataFrame = [UserId: int, ProductId: int ... 3 more fields]
res2: Array[org.apache.spark.sql.Row] = Array([9527760,3414,null,false,true],
[9527760,684,null,false,true], [9527760,179,null,false,true], [9527760,2390,null,false,true],
[9527760,2680,null,false,true], [9527760,2264,null,false,true], [9434312,3623,null,false,true],
[9434312,3654,null,false,true], [9434312,1968,null,false,true], [9434312,4107,null,false,true])
```

Although the language for this notebook is Scala, 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, we 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

Since we want to work from a DataFrame that holds only top preferred products, as indicated where both `IsTopProduct` and `IsPreferredProduct` are true, we execute the following in a new cell:

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

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

topPreferredProducts.show(100)
```

Cell 4

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

Command executed in 7s 788ms by joel on 04-20-2020 15:21:55.224 -04:00

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

UserId	ProductId	ItemsPurchasedLast12Months	IsTopProduct	IsPreferredProduct
9596264	4431	99	true	true
9628230	4690	99	true	true
9596264	4431	99	true	true
9596264	4431	99	true	true
9596264	4431	99	true	true
9596264	4431	99	true	true
9596264	4431	99	true	true
9596264	4431	99	true	true
9578837	3814	99	true	true
9578837	3814	99	true	true
9578837	3814	99	true	true
9578837	3814	99	true	true
9578837	3814	99	true	true
9578837	3814	99	true	true
9578837	3814	99	true	true
9578837	3814	99	true	true
9591502	3813	99	true	true
9591502	3812	99	true	true

Synapse notebooks allows you to switch the language for a given cell. We want to use SQL syntax to easily perform aggregates and store the results in a new temporary view:

```
%%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 and IsPreferredProduct = true
    order by a.UserId
```

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.

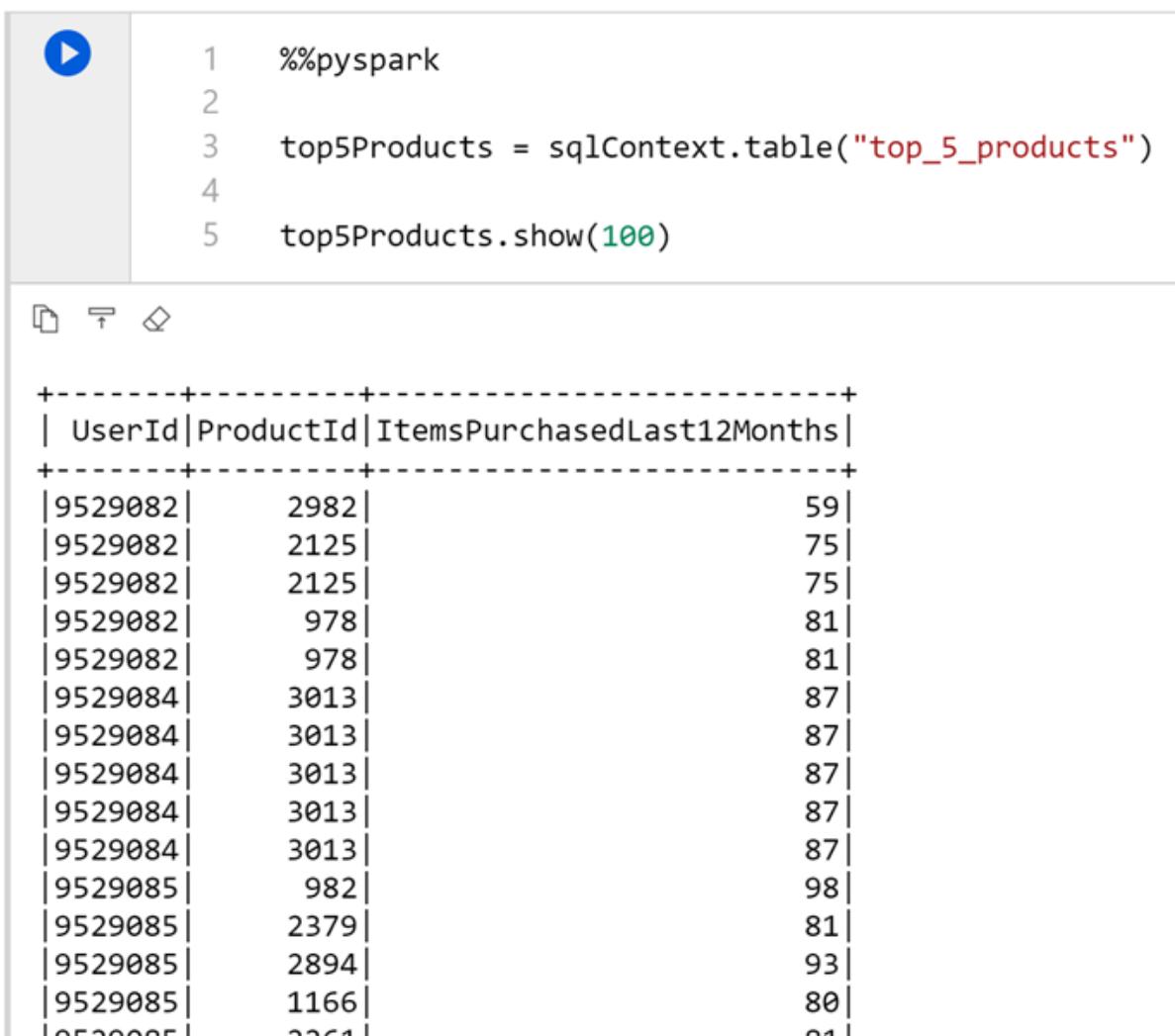
We can use the following method like we did earlier to create and display a new DataFrame that stores the results of the temporary view. The following code declares a `top_5_products` DataFrame and populates it with results of the temporary view we created in the previous cell:

```
%%pyspark

top5Products = sqlContext.table("top_5_products")

top5Products.show(100)
```

This results in the following output, which displays the top five preferred products per user:



The screenshot shows a Jupyter Notebook cell with the following content:

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

Below the code, there is a table output:

UserId	ProductId	ItemsPurchasedLast12Months
9529082	2982	59
9529082	2125	75
9529082	2125	75
9529082	978	81
9529082	978	81
9529084	3013	87
9529084	3013	87
9529084	3013	87
9529084	3013	87
9529085	982	98
9529085	2379	81
9529085	2894	93
9529085	1166	80
9529085	2261	61

Next, we create 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 of this cell is: before filter: 9662384 , after filter: 822044.

Finally, we calculate 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()
```

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.

This is the output of the query:

ProductId	Total
2107	4538
4833	4533
347	4523
3459	4233
4246	4155