

Data Integration Part 2

Resource naming throughout this lab

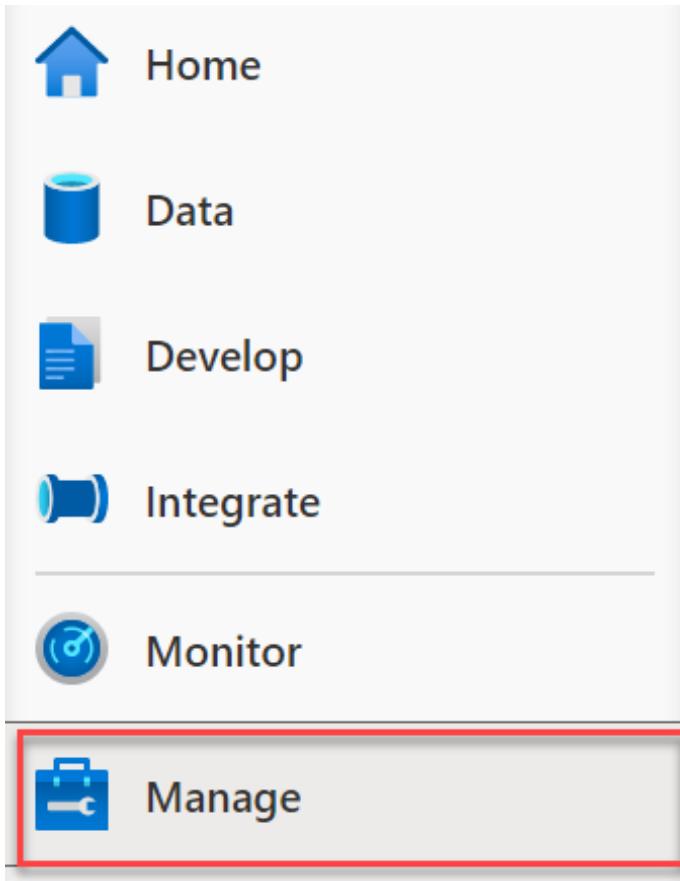
For the remainder of this guide, the following terms will be used for various ASA-related resources (make sure you replace them with actual names and values):

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

Lab prerequisite

Start the SQL Pool in your lab environment.

1. Open the [Synapse Studio workspace](#) and navigate to the **Manage** hub.



The Manage menu item is highlighted.

2. From the center menu, select **SQL pools** from beneath the **Analytics pools** heading. Locate **SQLPool01**, and select the **Resume** button.

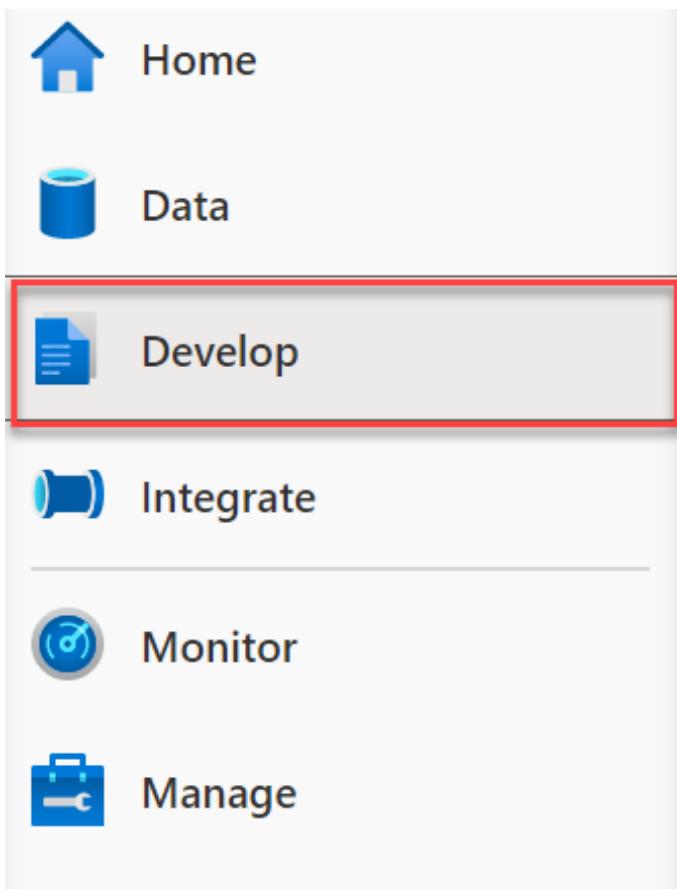
The screenshot shows the Microsoft Azure Synapse Analytics portal interface. The top navigation bar displays 'Microsoft Azure' and the workspace name 'asaworkspacezst123'. Below the navigation bar, there is a left sidebar with several menu items: Home, Data, Develop, Integrate, Monitor, and Manage. The 'Manage' item is highlighted with a red box. The main content area has two tabs: 'Analytics pools' and 'SQL pools'. The 'SQL pools' tab is selected and highlighted with a red box. Under the 'SQL pools' tab, it says 'The serverless SQL pool, Built-in, is immediately available for' and shows a list with one item: 'SQLPool01'. To the right of 'SQLPool01' are three icons: a blue triangle pointing right (highlighted with a red box), a copy icon, and an ellipsis icon.

The Manage menu item is selected, with SQL pools selected from the center menu. The resume button is selected next to the SQLPool01 item.

Exercise 1: Create datasets and SQL tables

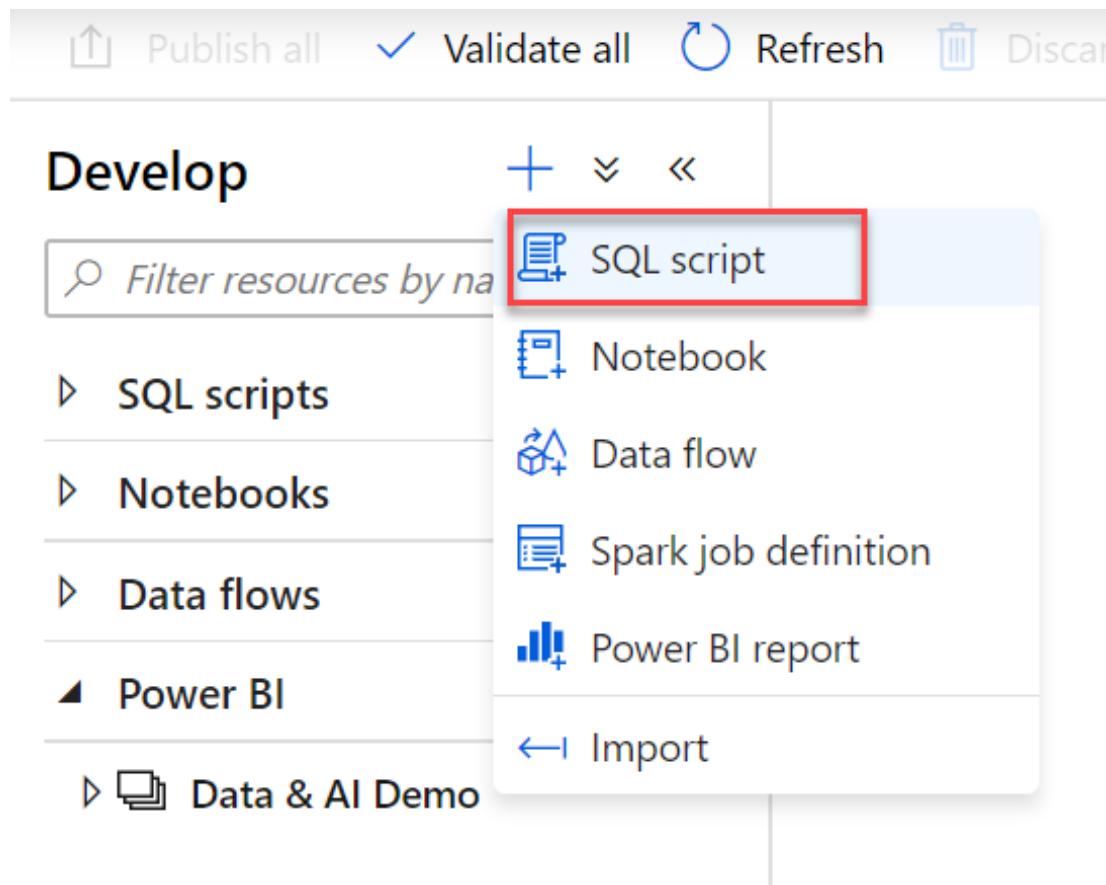
Task 1: Create SQL tables

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



The Develop menu item is highlighted.

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



The SQL script context menu item is highlighted.

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



The connect to option is highlighted in the query toolbar.

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 run button is highlighted in the query toolbar.

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

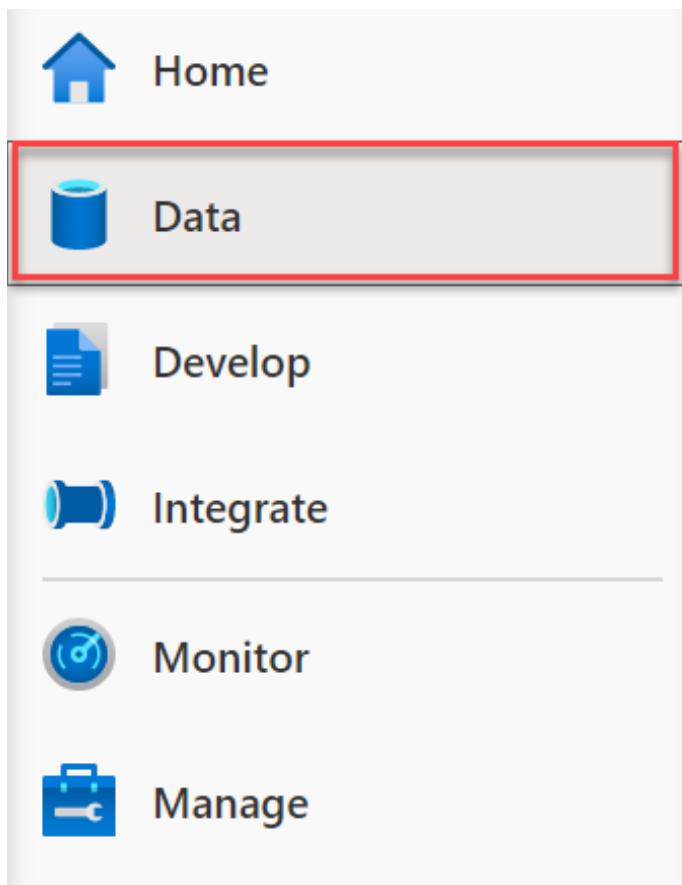
Your organization was provided a poorly formatted CSV file containing marketing campaign data. The file was uploaded to the data lake and now it must be imported into the data warehouse.

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	D&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	D&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	D&Cor	Be Unique	\$8\	284 \$9\		840

Screenshot of the CSV file.

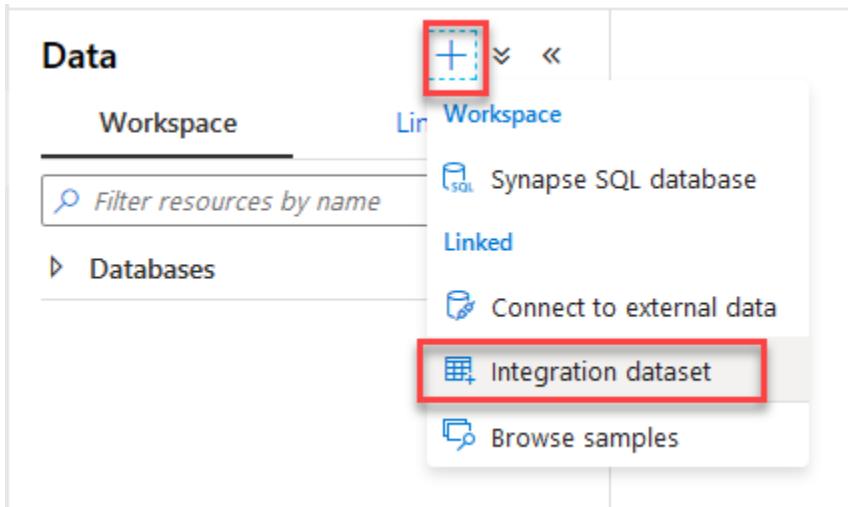
Issues include invalid characters in the revenue currency data, and misaligned columns.

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



The Data menu item is highlighted.

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



Create new Dataset.

- Create a new **Azure Data Lake Storage Gen2** integration dataset with the **DelimitedText** format type with the following characteristics:
 - Name:** Enter `asal400_campaign_analytics_source`.
 - Linked service:** Select the `asadatalakeSUFFIX` linked service.
 - File path:** Browse to the `wwi-02/campaign-analytics/campaignanalytics.csv` path.
 - First row as header:** Leave unchecked. **We are skipping the header** because there is a mismatch between the number of columns in the header and the number of columns in the data rows.
 - Import schema:** Select `From connection/store`.

Set properties

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

Name

Linked service *

File path

  | 

First row as header

Import schema

From connection/store From sample file None

The form properties are configured as described.

- 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 Default (\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.

The screenshot shows the 'Connection' tab selected in a configuration interface. The form contains the following settings:

Linked service *	asadatalake212045	Test connection	Edit	New
File path *	wwi-02 / campaign-analytics / campaignanalytics.csv	Browse	Preview data	
Compression type	none			
Column delimiter	Comma (,)	Edit		
Row delimiter	Auto detect (\r,\n, or \r\n)	Edit		
Encoding	Default(UTF-8)			
Escape character	Backslash (\)	Edit		
Quote character	Double quote (")	Edit		
First row as header	<input type="checkbox"/>			
Null value				

The configuration settings under Connection are set as defined.

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

A preview of the CSV file is displayed.

7. Create a new **Azure Synapse Analytics** integration 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
asal400_wwi_campaign_analytics_asa

Linked service *
 ▼

[Edit connection](#)

Table name
 ↻ ▼

Edit

Import schema
 From connection/store None

OK **Back** **Cancel**

New dataset form is displayed with the described configuration.

Task 3: Create user profile datasets

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

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

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

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

Set properties

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

Name
asal400_wwi_userproductreviews_asa

Linked service *
[dropdown menu]
[Edit connection](#)

Table name
wwi.UserProductReviews
 Edit

Import schema
 From connection/store None

OK **Back** **Cancel**

New dataset form is displayed with the described configuration.

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

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

Set properties

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

Name
asal400_wwi_usertopproductpurchases_asa

Linked service *
[Edit connection](#)

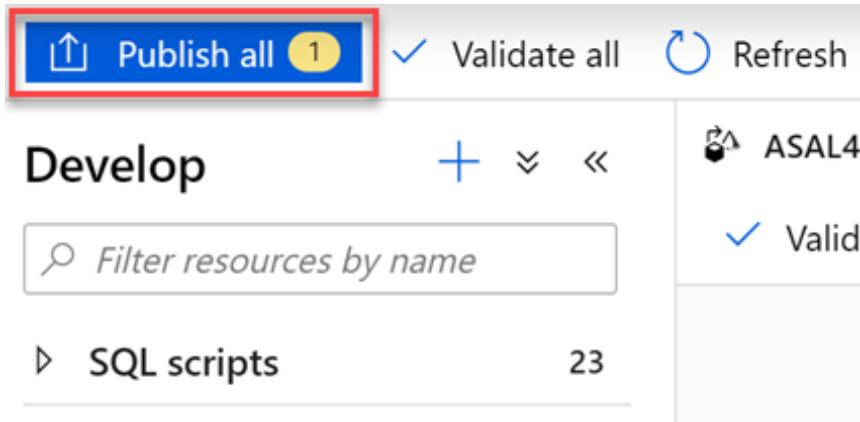
Table name
wwi.UserTopProductPurchases
[Edit](#)

Import schema
 From connection/store None

OK **Back** **Cancel**

New dataset form is displayed with the described configuration.

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



Publish all is highlighted.

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.

Set properties

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

Name
asal400_customerprofile_cosmosdb

Linked service *
asacosmosdb01

[Edit connection](#)

Collection
OnlineUserProfile01

Edit

Import schema
 From connection/store None

New Azure Cosmos DB dataset.

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



CosmosDB Collection (SQL API)
asal400_customerprofile_cosmosdb

Connection	Schema	Parameters
Linked service *	asacosmosdb1	Test connection Edit New
Collection	OnlineUserProfile01	Refresh Preview data
	<input type="checkbox"/> Edit	

The preview data button on the dataset is highlighted.

- 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-23191dbbf72de",  
    "_rid": "OC8bALmbB4kBAQAAAQAAAQAA==",  
    "_self": "dbs/OC8bAA=/colls/OC8bALmbB4k=/docs/OC8bALmbB4kBAQAAAQAAAQAA==/",  
    "_etag": "\"2101cde1-0000-0200-0000-5e969f4b0000\"",  
    "_attachments": "attachments/",  
    "_ts": 1586929483  
  },  
  {  
    "userId": 9079747,  
    "cartId": "5c4dc5dc-a585-41ec-8149-9133caa3a73a",  
    "preferredProducts": [  
      4235,  
      3288,  
      2756  
    ],  
    "productReviews": [
```

A preview of the Azure Cosmos DB data is displayed.

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.

The inferred schema for the Azure Cosmos DB documents is displayed.

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

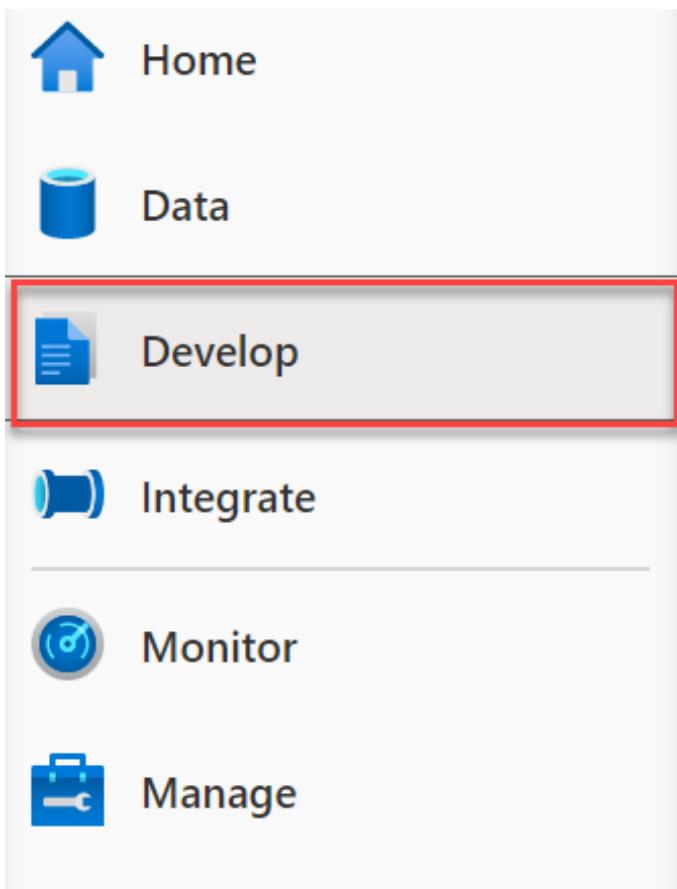
- The inferred schema for the Azure Cosmos DB documents is displayed.*
5. Create a new **Azure Data Lake Storage Gen2** dataset with the **JSON** format type with the following characteristics:
 - **Name:** Enter `asal400_ecommerce_userprofiles_source`.
 - **Linked service:** Select the `asadatalakeXX` linked service that already exists.
 - **File path:** Browse to the `wwi-02/online-user-profiles-02` path.
 - **Import schema:** Select `From connection/store`.
 6. Select **Publish all** to save your new resources.

Publish all is highlighted.

Exercise 2: Create data pipeline to import poorly formatted CSV

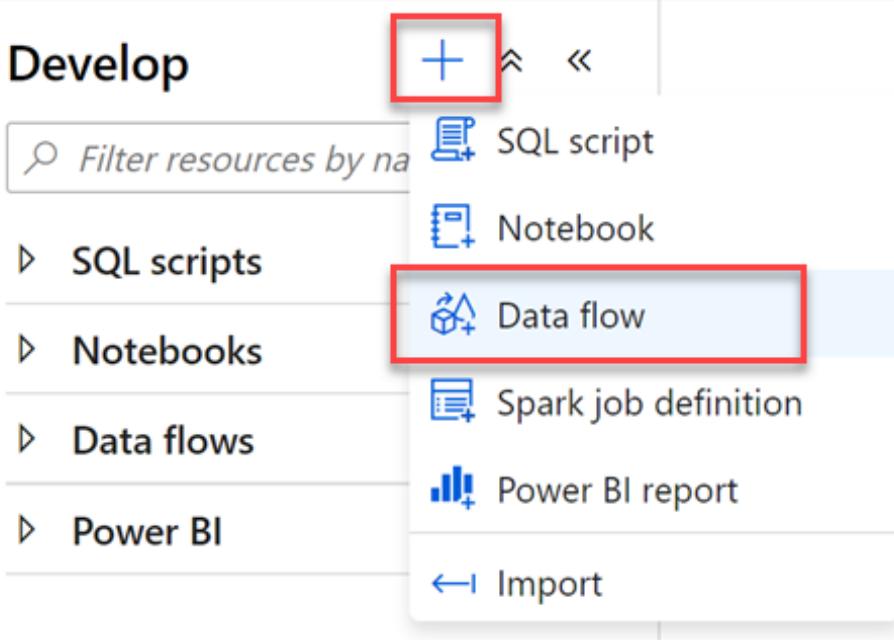
Task 1: Create campaign analytics data flow

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



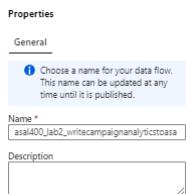
The Develop menu item is highlighted.

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



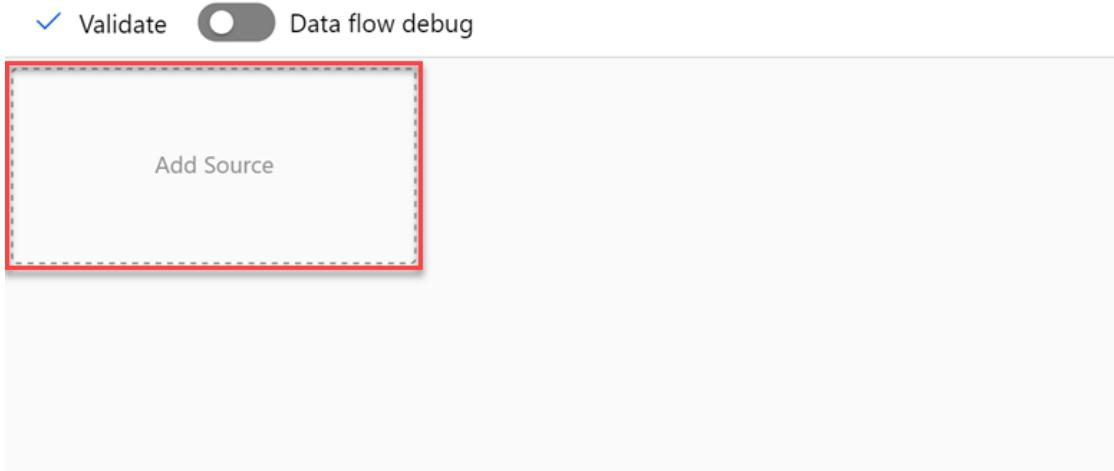
The new data flow link is highlighted.

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



The name field is populated with the defined value.

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



Select Add Source on the data flow canvas.

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

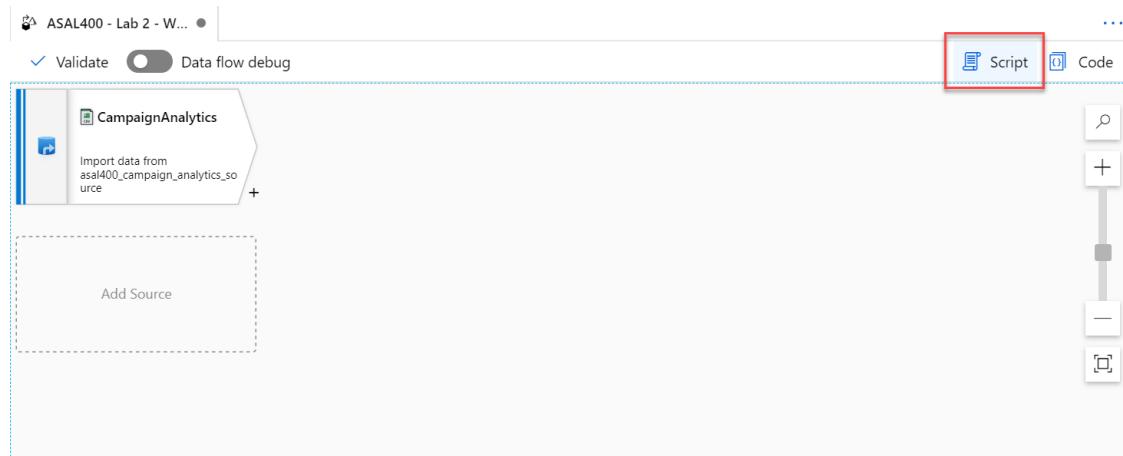
- **Output stream name:** Enter CampaignAnalytics.
- **Source type:** Select Integration dataset.
- **Dataset:** Select asal400_campaign_analytics_source.
- **Options:** Select Allow schema drift and leave the other options unchecked.
- **Skip line count:** Enter 1. This allows us to skip the header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

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

The form is configured with the defined settings.

6. When you create data flows, certain features are enabled by turning on debug, such as previewing data and importing a schema (projection). Due to the amount of time it

takes to enable this option, as well as environmental constraints of the lab environment, we will bypass these features. The data source has a schema we need to set. To do this, select **Script** above the design canvas.



The script link is highlighted above the canvas.

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

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

Your script should match the following:

Data flow name ASAL400 - Lab 2 - Write Campaign Analytic

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

OK

Copy as single line

Cancel

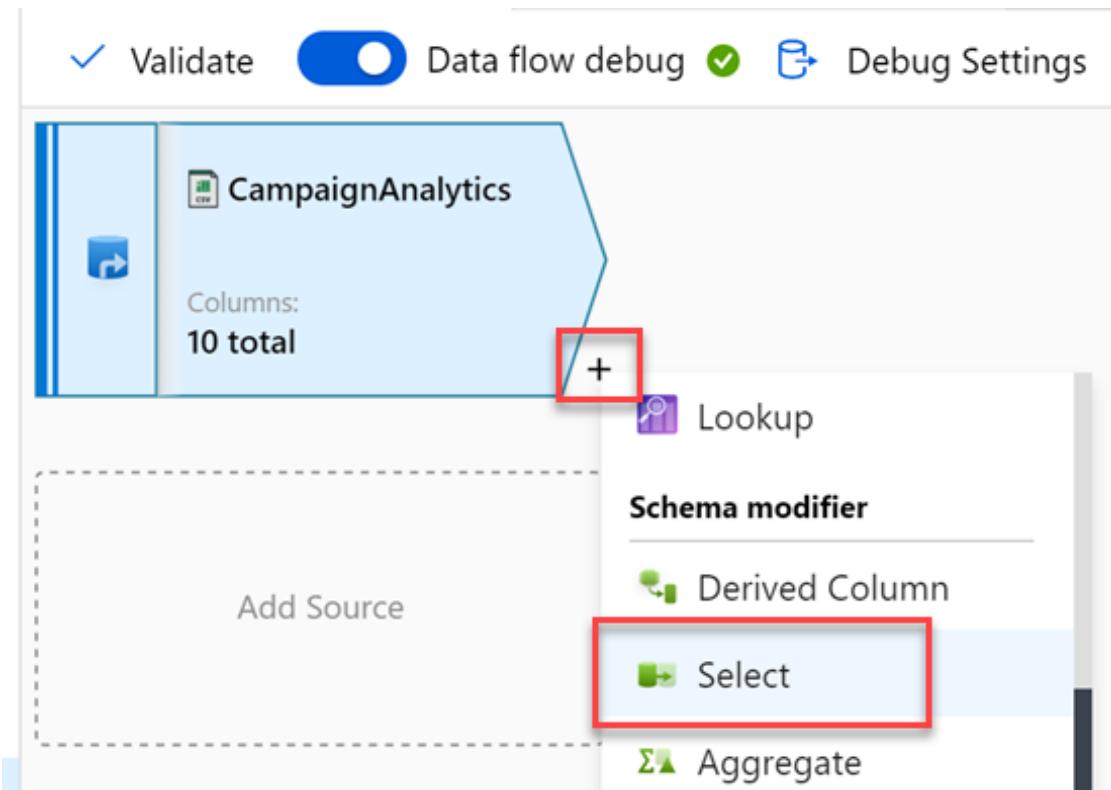
The script columns are highlighted.

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

The imported projection is displayed.

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



The new Select schema modifier is highlighted.

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

- Output stream name:** Enter MapCampaignAnalytics.

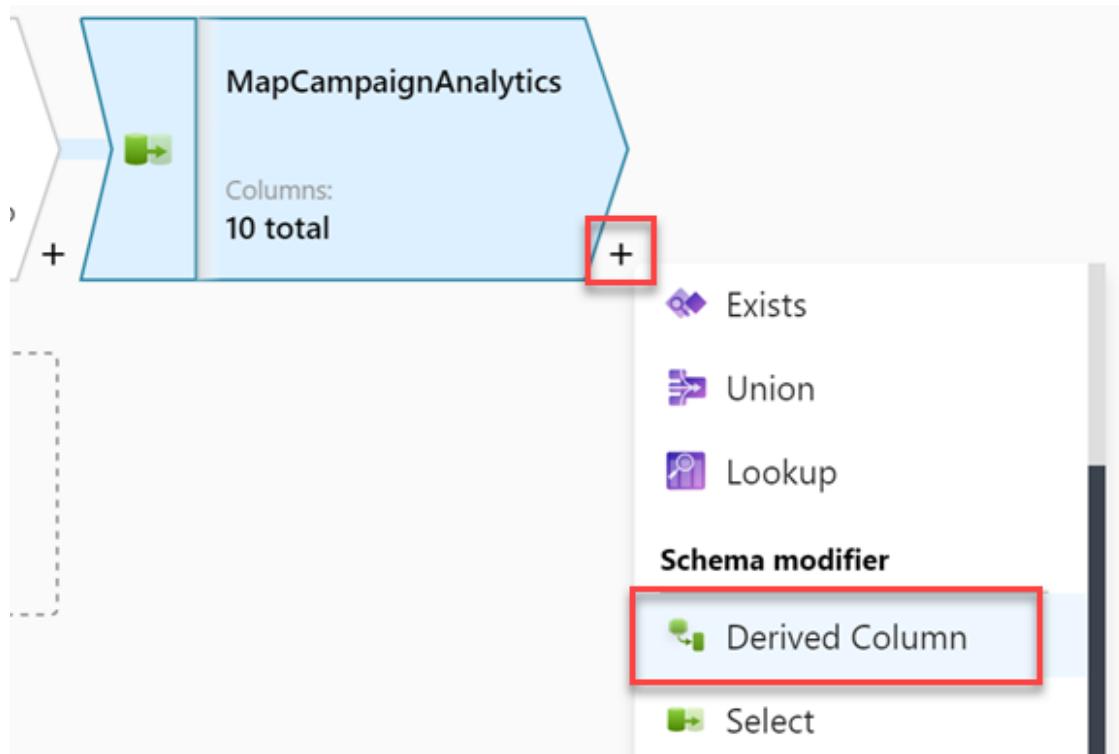
- **Incoming stream:** Select CampaignAnalytics.
- **Options:** Check both options.
- **Input columns:** make sure Auto mapping is unchecked, then provide the following values in the **Name as** fields:
 - Region
 - Country
 - ProductCategory
 - CampaignName
 - RevenuePart1
 - Revenue
 - RevenueTargetPart1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

The screenshot shows the 'Select settings' page for a mapping operation. The 'Output stream name' is set to 'MapCampaignAnalytics'. The 'Incoming stream' is 'CampaignAnalytics'. In the 'Options' section, 'Skip duplicate input columns' and 'Skip duplicate output columns' are checked. Under 'Input columns', the 'Auto mapping' toggle is off. The 'Mappings' table lists 10 mappings from 'CampaignAnalytics's column' to 'Name as':

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

The select settings are displayed as described.

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



The new Derived Column schema modifier is highlighted.

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

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

Column	Expression	Description
Revenue	toDecimal(concat(toString(RevenuePart1), toString(Revenue)), '\\", ''), 10, 2, '\$##,###.##')	Concatenate the RevenuePart1 and Revenue fields, replace the invalid \\ character with a single \, and format the number with 10 digits and 2 decimal places.

cter,
then
conve
rt and
forma
t the
data
to a
decim
al
type.

Conca
tenate
the
Reven
ueTar
getPa
rt1
and
Reven
ueTar
get
fields,
replac
e the
invali
d \
chara
cter,
then
conve
rt and
forma
t the
data
to a
decim
al
type.

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

Derived column's settings Optimize Inspect Data preview ●

Output stream name * ConvertColumnTypeAndValues Learn more

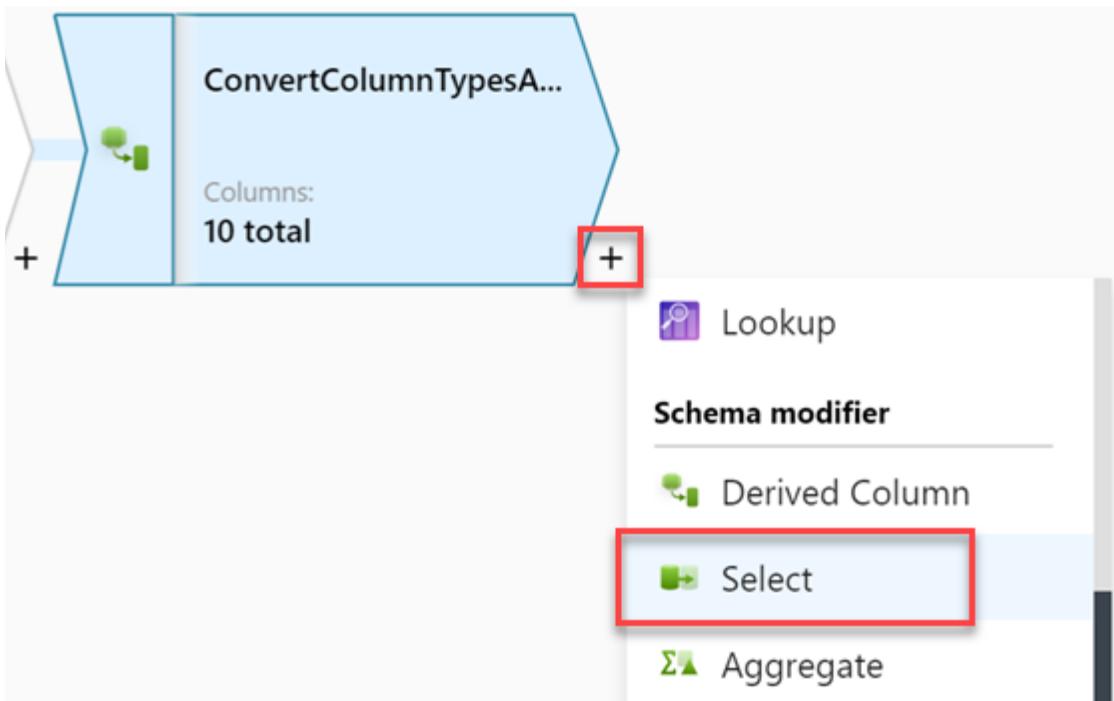
Incoming stream * MapCampaignAnalytics

Columns * Add Duplicate Delete

<input type="checkbox"/> Column	Expression
<input type="checkbox"/> Revenue	toDecimal(replace(concat(toString(RevenuePart1), t... e ^x
<input type="checkbox"/> RevenueTarget	toDecimal(replace(concat(toString(RevenueTargetP... e ^x

The derived column's settings are displayed as described.

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



The new Select schema modifier is highlighted.

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

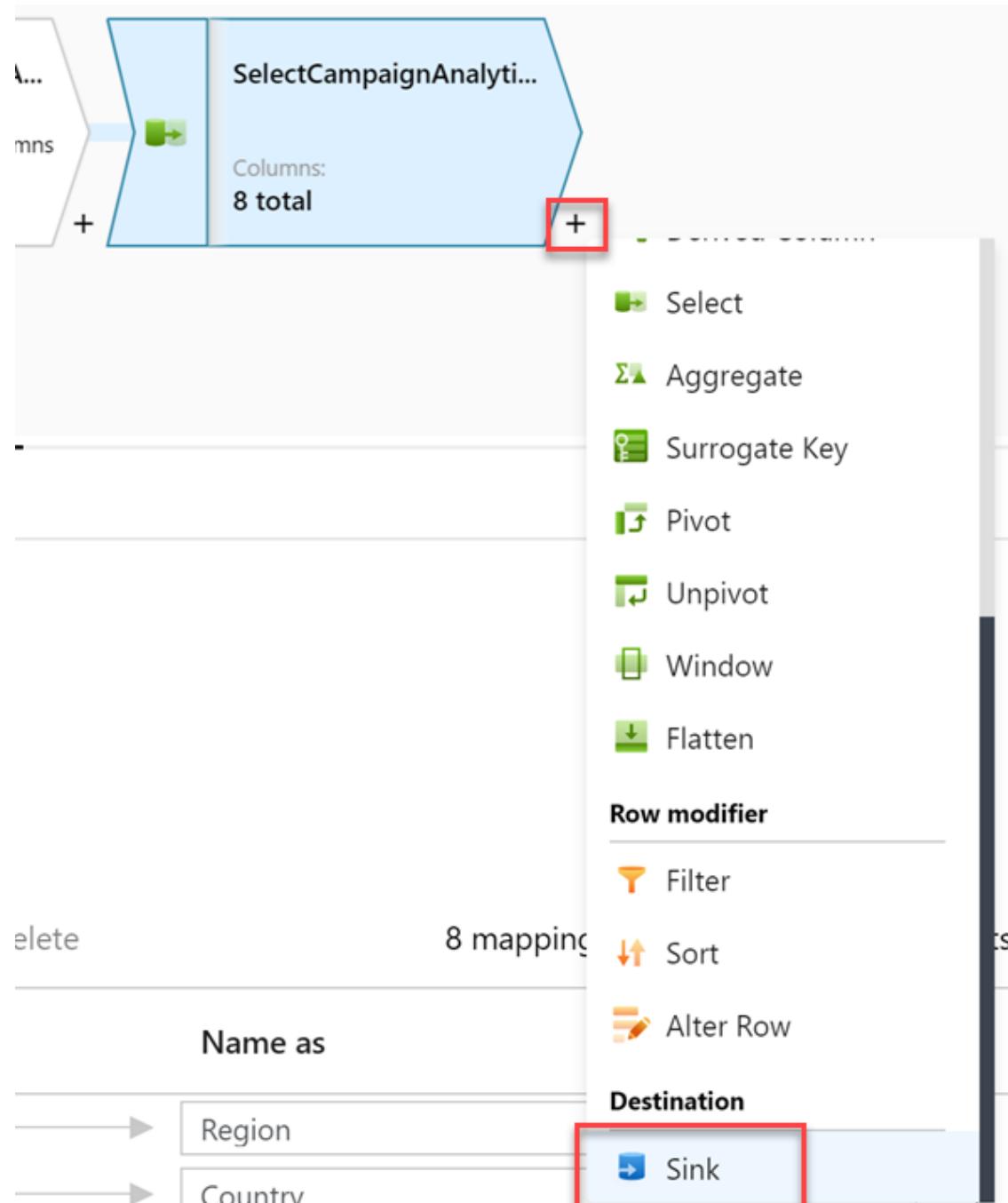
- **Output stream name:** Enter `SelectCampaignAnalyticsColumns`.
- **Incoming stream:** Select `ConvertColumnTypeAndValues`.
- **Options:** Check both options.
- **Input columns:** make sure **Auto mapping** is unchecked, then **Delete** `RevenuePart1` and `RevenueTargetPart1`. We no longer need these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

Screenshot of the 'Select settings' tab in a data flow configuration interface. The 'Output stream name' is set to 'SelectCampaignAnalyticsColumns'. The 'Incoming stream' is 'ConvertColumnTypesAndValues'. Under 'Options', 'Skip duplicate input columns' and 'Skip duplicate output columns' are checked. The 'Input columns' section shows 8 mappings from 2 column(s) left unnamed. The mappings are:

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

The select settings are displayed as described.

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



The new Sink destination is highlighted.

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

- **Output stream name:** Enter CampaignAnalyticsASA.
- **Incoming stream:** Select SelectCampaignAnalyticsColumns.
- **Sink type:** Select Integration dataset.
- **Dataset:** Select asa1400_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 * Learn more [↗](#)

Incoming stream *

Sink type *

Dataset * [↗ Test connection](#) [Open](#) [New](#)

Options Allow schema drift [ⓘ](#) Validate schema [ⓘ](#)

The sink settings are shown.

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 [●](#)

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

Update method Allow insert
 Allow delete
 Allow upsert
 Allow update

Table action None Recreate table Truncate table

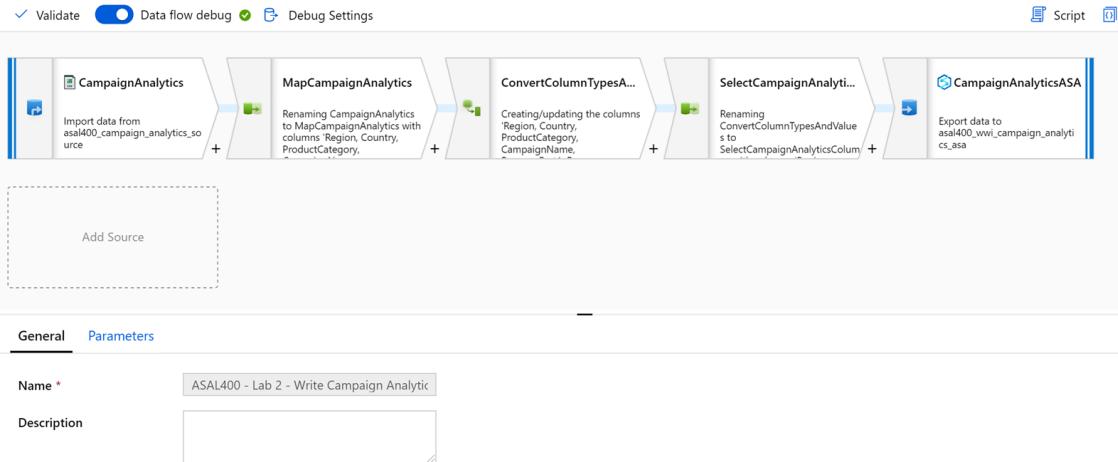
Enable staging

Batch size

Pre SQL scripts

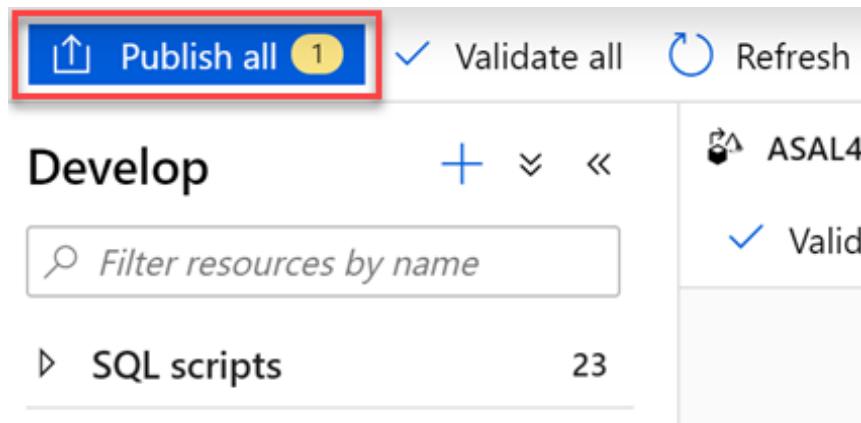
The settings are shown.

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



The completed data flow is displayed.

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

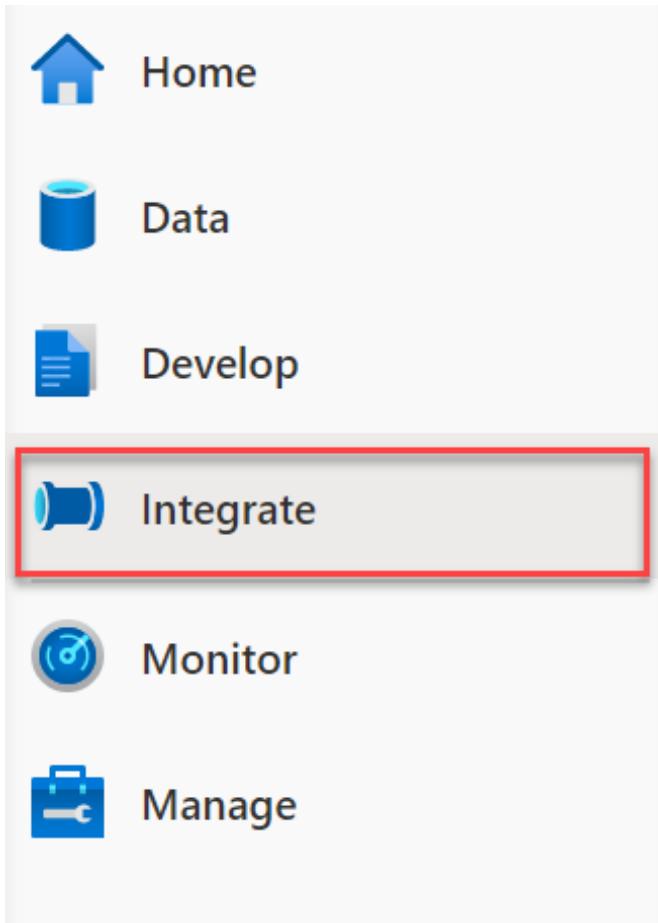


Publish all is highlighted.

Task 2: Create campaign analytics data pipeline

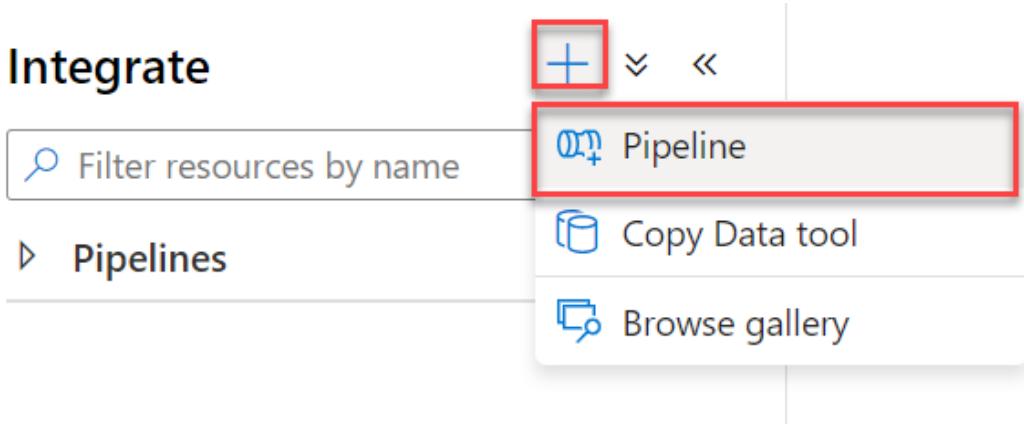
In order to run the new data flow, you need to create a new pipeline and add a data flow activity to it.

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



The Orchestrate hub is highlighted.

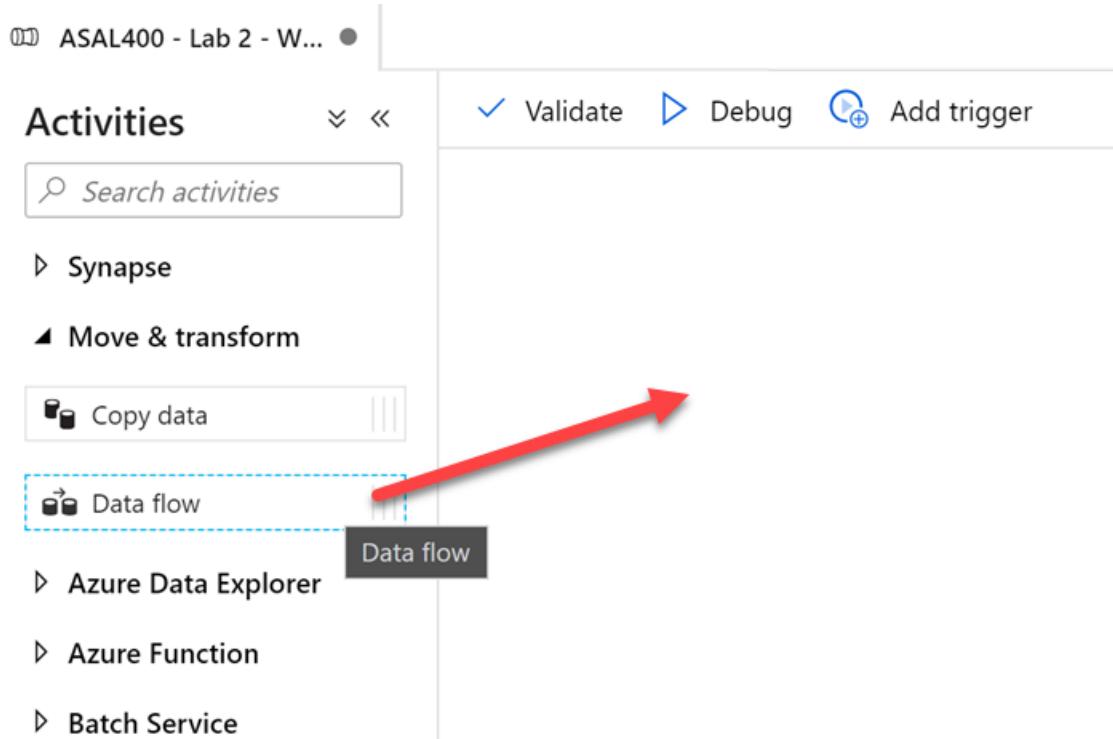
2. Select + then **Pipeline** to create a new pipeline.



The new pipeline context menu item is selected.

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.



Drag the data flow activity onto the pipeline canvas.

5. In the General tab of the Data flow activity, enter **campaign_analytics_data** as the name.

General Settings ¹ Parameters ¹ User properties

Name *	campaign_analytics_data	Learn more
Description		
Timeout ⓘ	7.00:00:00	
Retry ⓘ	0	
Retry interval ⓘ	30	
Secure output ⓘ	<input type="checkbox"/>	
Secure input ⓘ	<input type="checkbox"/>	

The mapping data flow General tab is shown

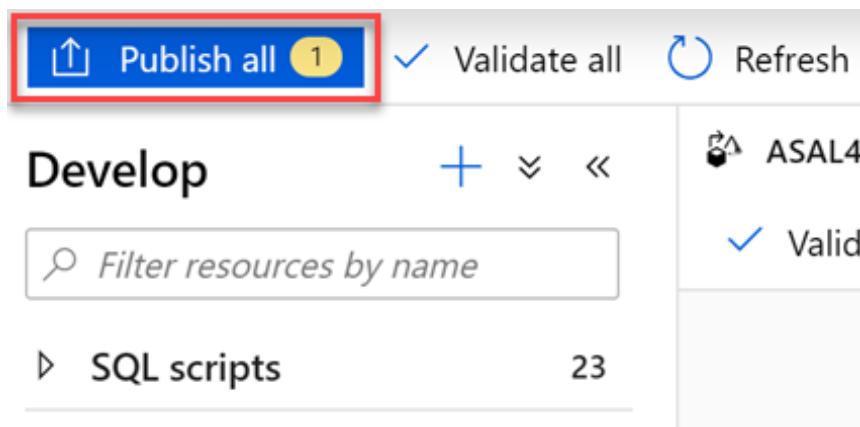
6. Select the **Settings** tab, select `asa1400_lab2_writecampaignanalytics` in the Data flow field, then ensure `AutoResolveIntegrationRuntime` is selected for **Run on (Azure IR)**. Choose the Standard (Memory Optimized) **Compute type** and select **16 (+ 16 cores)** for the **Core count**.

General **Settings** Parameters ¹ User properties

Data flow *	ASAL400 - Lab 2 - Write Campaign A...	Open	New
Run on (Azure IR) * ⓘ	AutoResolveIntegrationRuntime		
Compute type * ⓘ	Standard (Memory Optimized) Add dynamic content [Alt+Shift+D]		
Core count * ⓘ	16 (+ 16 Driver cores)		
Logging level * ⓘ	<input checked="" type="radio"/> Verbose <input type="radio"/> Basic <input type="radio"/> None		
▶ Sink properties ▶ Staging ⓘ			

The custom IR is selected in the mapping data flow activity settings.

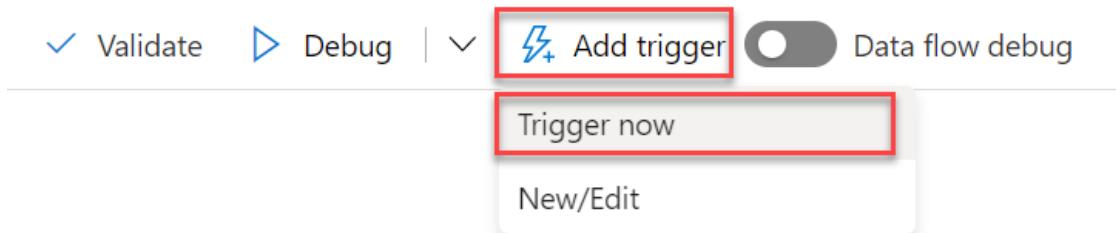
7. Select **Publish all** to save your new pipeline.



Publish all is highlighted.

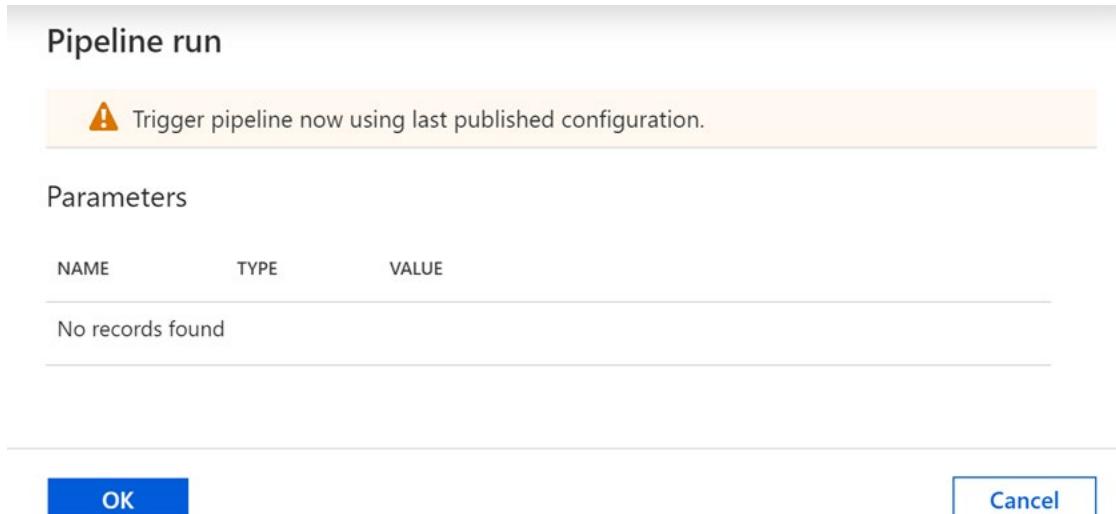
Task 3: Run the campaign analytics data pipeline

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



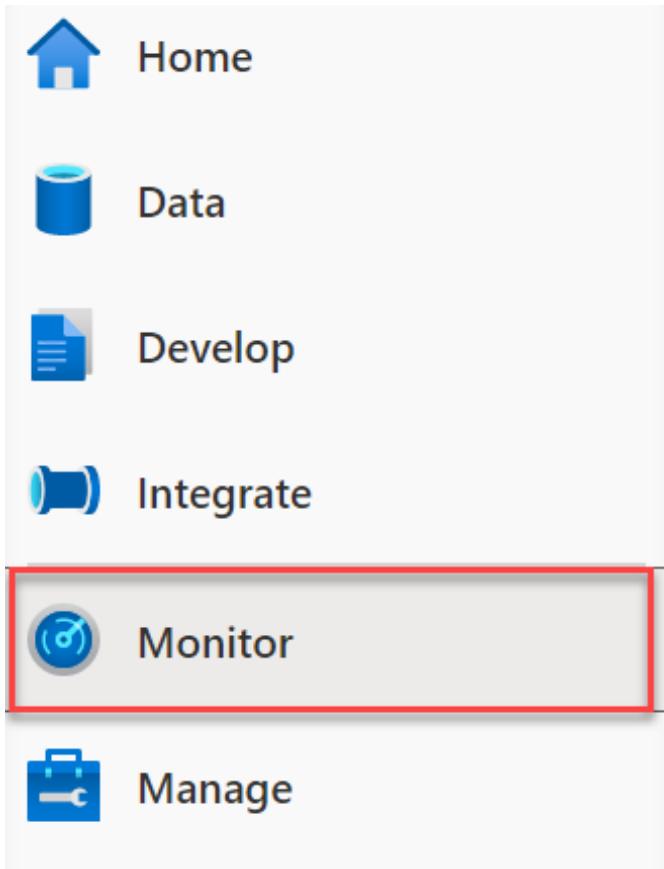
The add trigger button is highlighted.

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



The pipeline run blade is displayed.

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



The *Monitor* hub menu item is selected.

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

A screenshot of the Pipeline runs page. The left sidebar shows categories: Orchestration, Pipeline runs, Trigger runs, Integration runtimes, Activities, Apache Spark applications, and SQL requests. The main area displays pipeline runs with the following details:

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	(green checkmark)

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

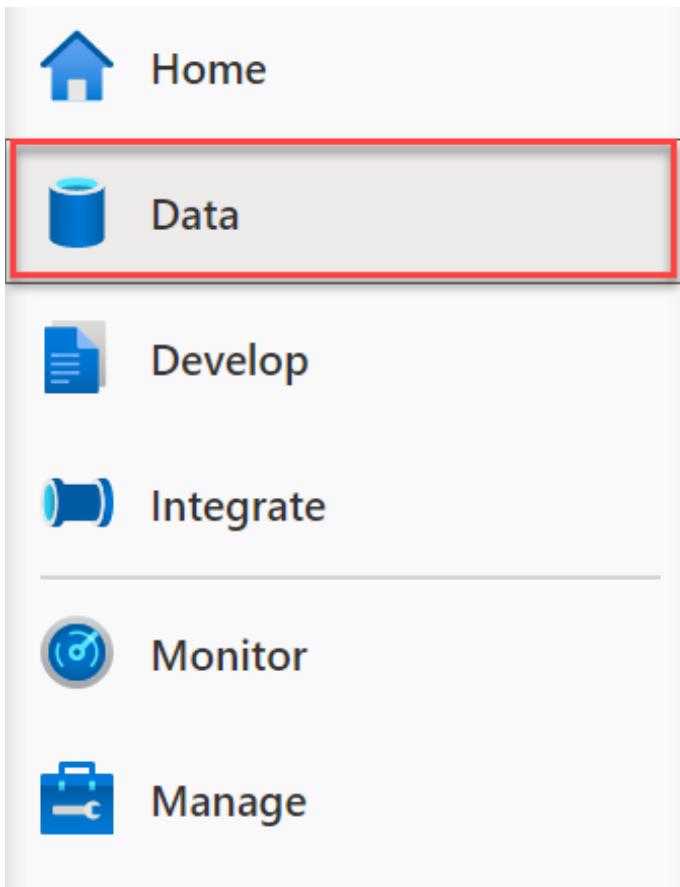
The pipeline run succeeded.

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.



The Data menu item is highlighted.

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

The screenshot shows the Azure Data Studio interface in the 'Data' workspace. On the left, the 'Tables' section is expanded, showing several tables under 'wwi'. One table, 'wwi.CampaignAnalytics', is selected and highlighted with a red box. On the right, a context menu is open for this table, listing options like 'New SQL script', 'New notebook', 'Data flow', 'Dataset', and 'Refresh'. The 'Select TOP 100 rows' option is also present in the menu and is highlighted with a red box.

The Select TOP 100 rows menu item is highlighted.

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

The screenshot shows the Azure Data Studio query editor with a query run against 'SQLPool02'. The query is:

```

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

```

The results are displayed in a table format:

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

The CampaignAnalytics query results are displayed.

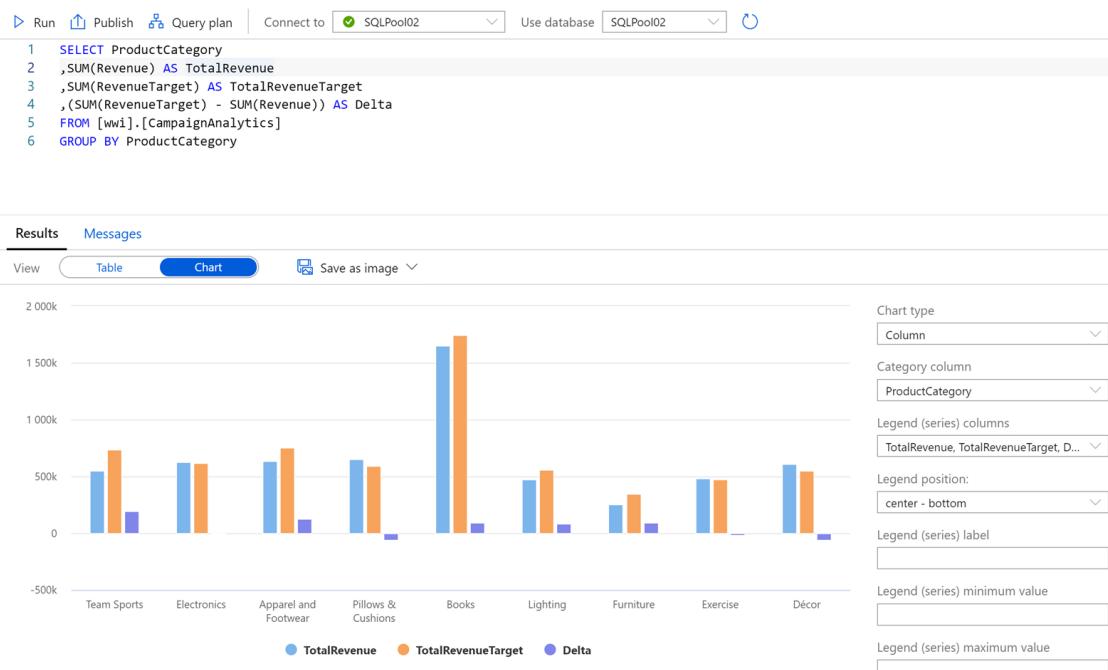
5. Update the query to the following and Run:

```

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.



The new query and chart view are displayed.

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

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

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

The screenshot shows a SQL query editor interface. At the top, there are buttons for Run, Publish, Query plan, Connect to, Use database, and a refresh icon. The query plan dropdown is set to 'SQLPool02'. The use database dropdown is set to 'SQLPool02'. The main area contains the following SQL code:

```

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

```

Below the code, there are tabs for Results and Messages, with Results selected. Under Results, there are options for Table (selected), Chart, and Export results. A search bar is also present. The results table has the following columns: REGION, COUNTRY, PRODUCTCATEGORY, CAMPAIGNNAME, REVENUE, REVENUETARGET, CITY, and STATE. The data is as follows:

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

The CampaignAnalytics query results are displayed.

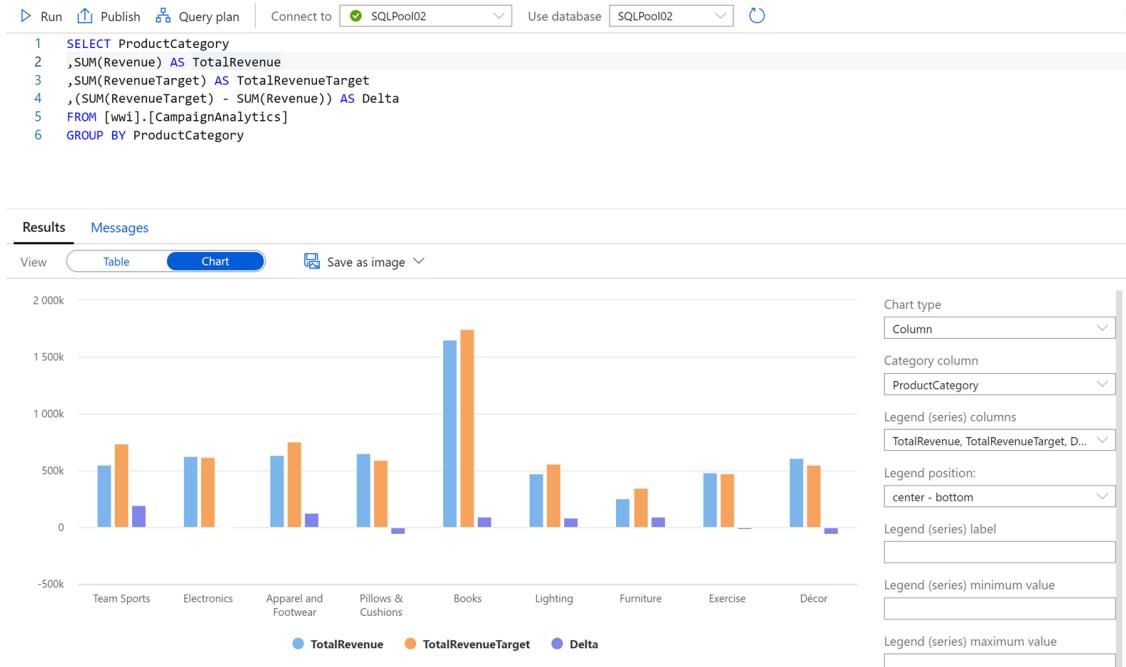
If we wish to view the total revenue compared to the target revenue of each product category, we can update the query as follows:

```

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:

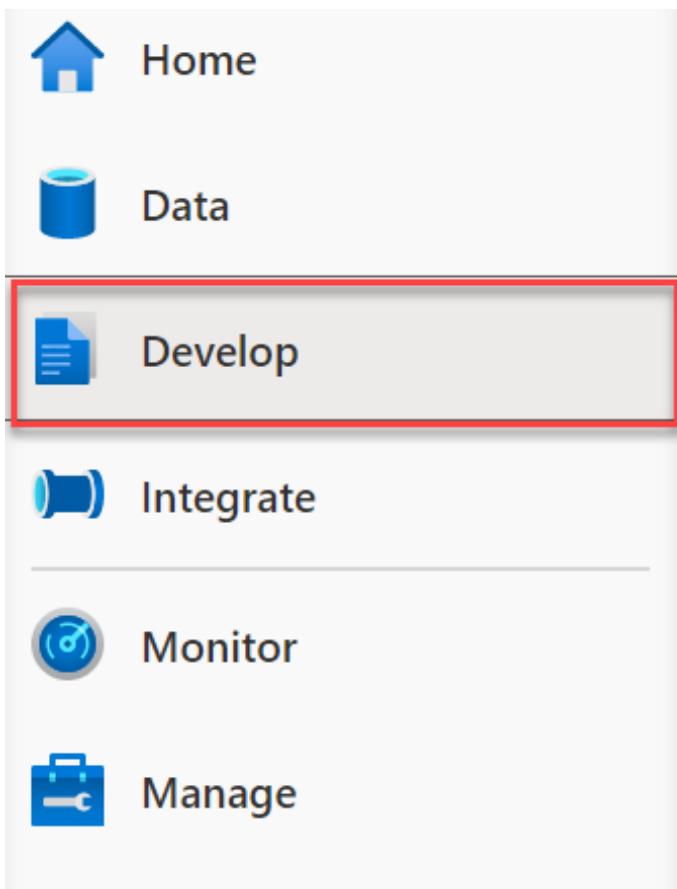


The new query and chart view are displayed.

Exercise 3: Create data pipeline to join disparate data sources

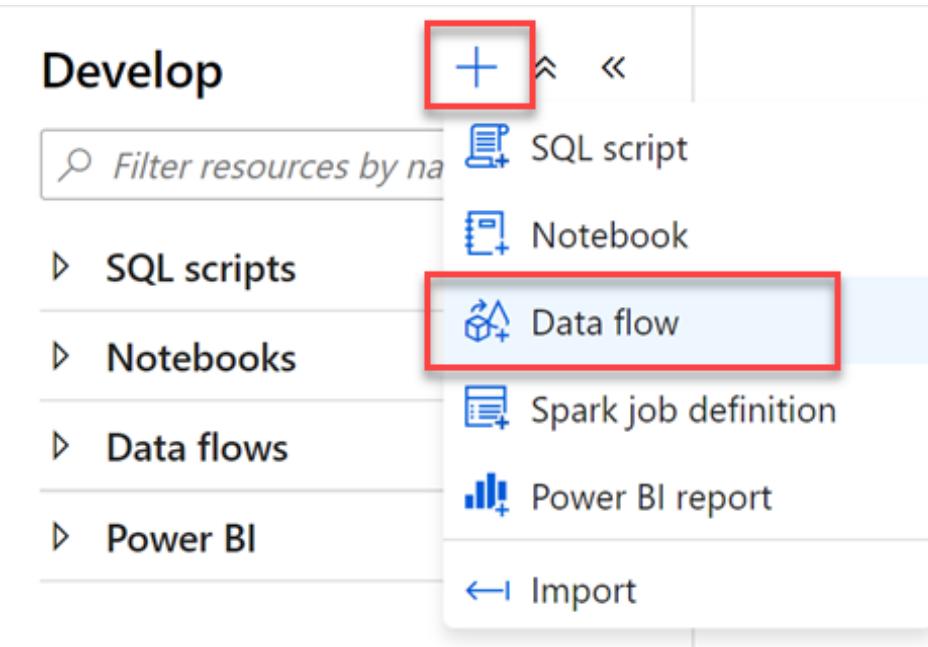
Task 1: Create user profile data flow

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



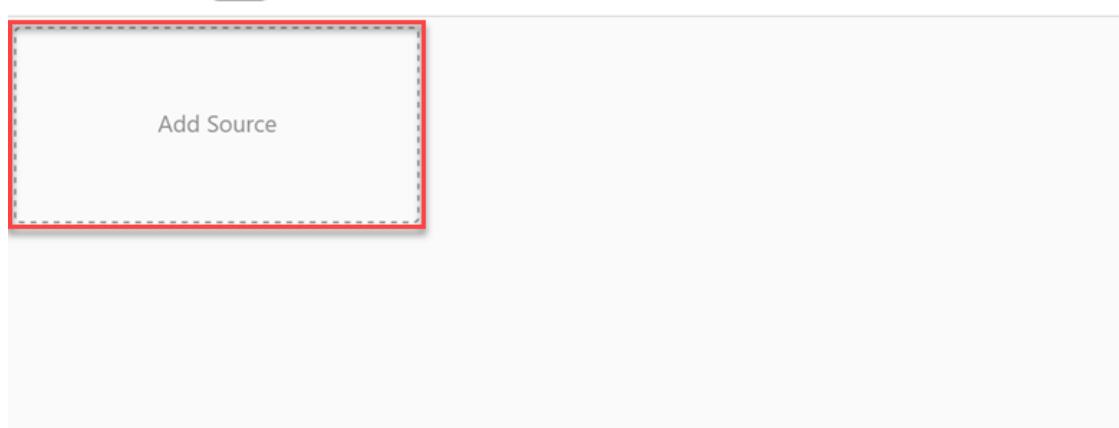
The Develop menu item is highlighted.

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



The new data flow link is highlighted.

3. In the **General** section of the **Properties** pane of the new data flow, update the **Name** to the following: `asal400_lab2_writeuserprofiledataosa`.
4. Select **Add Source** on the data flow canvas.



Select Add Source on the data flow canvas.

5. Under **Source settings**, configure the following:
 - **Output stream name:** Enter `EcommerceUserProfiles`.
 - **Source type:** Select `Integration dataset`.
 - **Dataset:** Select `asal400_ecommerce_userprofiles_source`.

Source settings Source options Projection Optimize Inspect Data preview

Output stream name * EcommerceUserProfiles Learn more [↗](#)

Source type * [Integration dataset](#) [Inline](#) [Workspace DB](#)

Dataset * asal400_ecommerce_userprofiles_so... [↗](#) [Test connection](#) [Open](#) [New](#)

Options

- Allow schema drift [ⓘ](#)
- Infer drifted column types [ⓘ](#)
- Validate schema [ⓘ](#)

Sampling * [ⓘ](#) Enable Disable

The source settings are configured as described.

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 **Array of documents** setting for the **Document form** field. This denotes that each file contains multiple JSON documents contained within an array.

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

Wildcard paths wwi-02 [online-user-profiles-02/*.json](#) [+ ↗](#)

Partition root path [ⓘ](#)

Allow no files found [ⓘ](#)

List of files [ⓘ](#)

Column to store file name [ⓘ](#)

After completion * No action Delete source files Move

Start time (UTC) [ⓘ](#) End time (UTC) [ⓘ](#)

Filter by last modified [ⓘ](#)

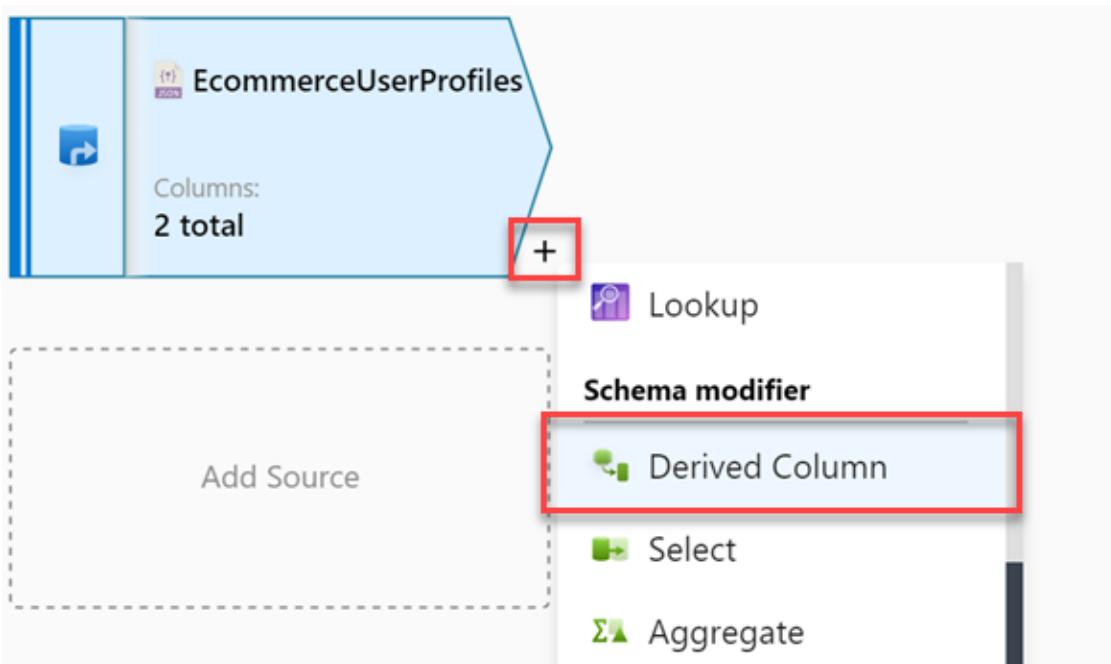
▲ JSON settings

Document form [ⓘ](#) Single document Document per line Array of documents

Unquoted column names [ⓘ](#)

The source options are configured as described.

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



The plus sign and Derived Column schema modifier are highlighted.

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

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

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

Derived column's settings		Optimize	Inspect	Data preview				
Output stream name *	userId	Learn more						
Incoming stream *	EcommerceUserProfiles							
Columns * ⓘ	<input type="button" value="Add"/> <input type="button" value="Duplicate"/> <input type="button" value="Delete"/> <table border="1"> <thead> <tr> <th>Column</th> <th>Expression</th> </tr> </thead> <tbody> <tr> <td>visitorId</td> <td>toInteger(visitorId)</td> </tr> </tbody></table>	Column	Expression	visitorId	toInteger(visitorId)	<input type="button" value="+"/> <input type="button" value="Duplicate"/> <input type="button" value="Delete"/>		
Column	Expression							
visitorId	toInteger(visitorId)							

The derived column's settings are configured as described.

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

userId

Columns:
2 total

+

+ 

Union

Lookup

Schema modifier

Derived Column

Select

Aggregate

Surrogate Key

Pivot

Unpivot

Window

Rank

Formatters

Flatten 

Parse

Row modifier

Filter

Sort

Alter Row

Destination

Sink

Optimize Inspect Data preview

serId

commerceUserProfiles

+ Add  Clone  Delete

Column

visitorId

The plus sign and the Flatten schema modifier are highlighted.

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

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

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

Flatten settings Optimize Inspect Data preview ●

Output stream name * ? Help Learn more i

Incoming stream * *

Unroll by * *

Unroll root

Options Skip duplicate input columns i
 Skip duplicate output columns i

Input columns * ↻ Reset + Add mapping Delete 3 mappings: All input:

userId's column	Name as
<input type="checkbox"/> abc visitorId	<input type="text" value="visitorId"/> + ✖
<input type="checkbox"/> abc topProductPurchases.productId	<input type="text" value="productId"/> + ✖
<input type="checkbox"/> abc topProductPurchases.itemsPurchasedLast...	<input type="text" value="itemsPurchasedLast12Months"/> + ✖

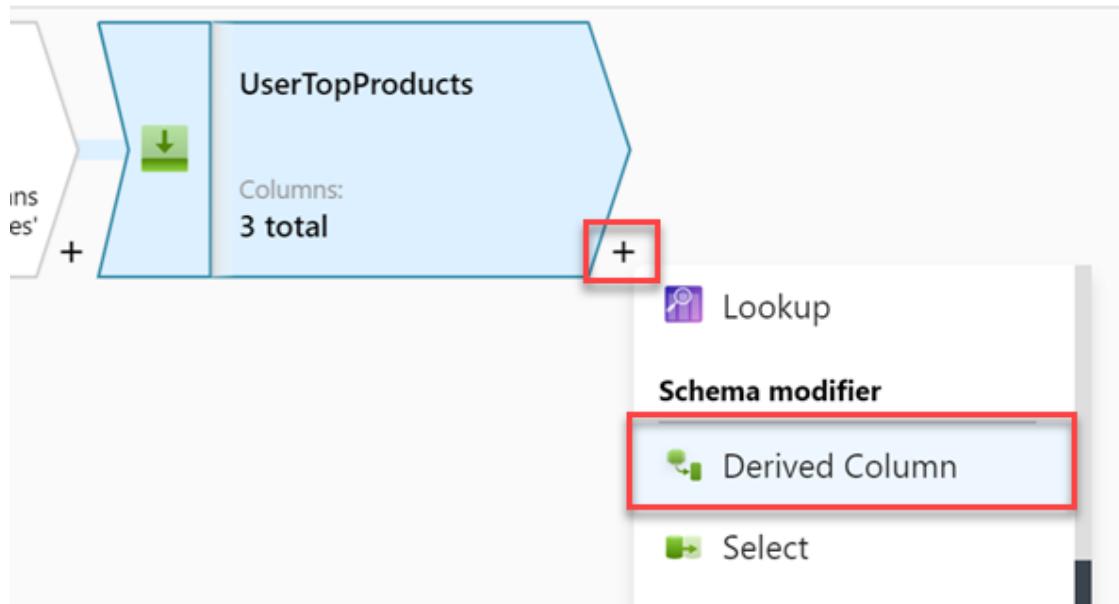
The flatten settings are configured as described.

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	Map drifted	Statistics	X 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	

The data preview tab is displayed with a sample of the file contents.

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



The plus sign and Derived Column schema modifier are highlighted.

- 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
<code>productId</code>	<code>toInteger(productId)</code>	Converts the <code>productId</code>

itemsPurchase
dLast12Months

toInteger(itemsPurch
asedLast12Months)

column
from a
string to an
integer.

Converts
the
itemsPur
chasedLast
12Months
column
from a
string to an
integer.

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

Output stream name * [Learn more](#)

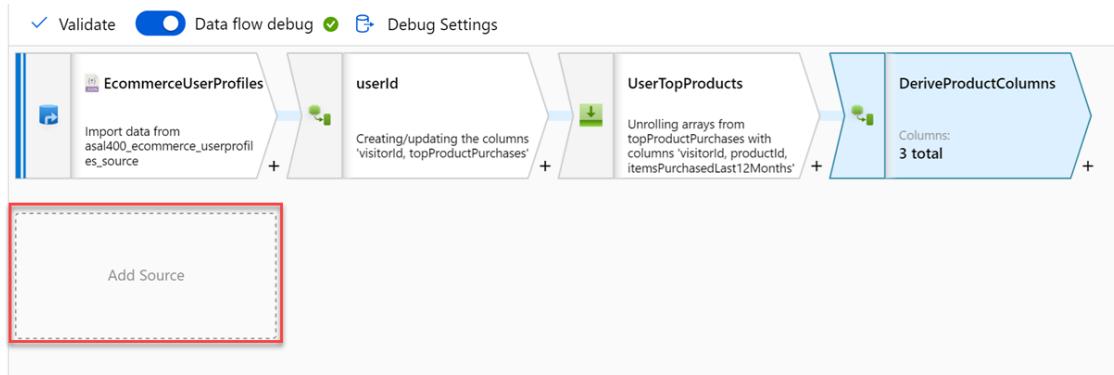
Incoming stream *

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

Column	Expression
productId	toInteger(productId) 123 + ↻ -
itemsPurchasedLast12Months	toInteger(itemsPurchasedLast12Months) 123 + ↻ -

The derived column's settings are configured as described.

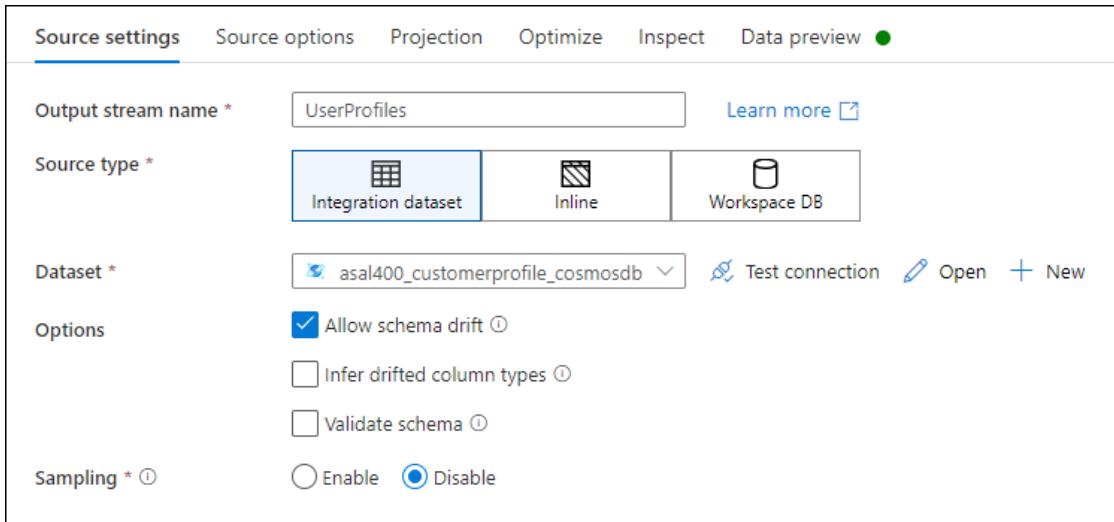
13. Select **Add Source** on the data flow canvas beneath the UserProfiles source.



Select Add Source on the data flow canvas.

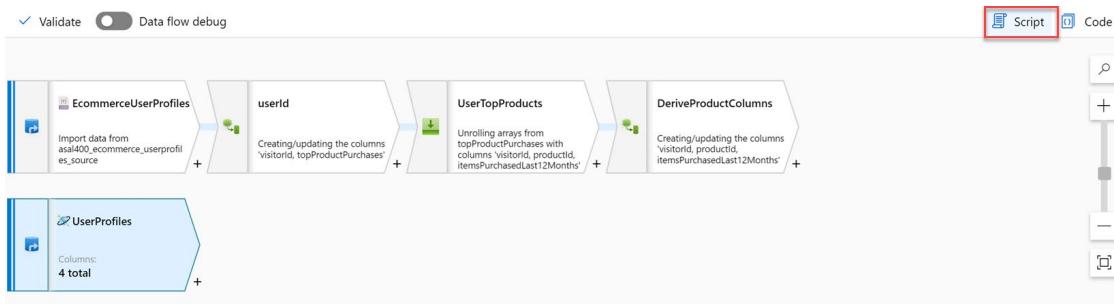
14. Under **Source settings**, configure the following:

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



The source settings are configured as described.

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



The Script link is highlighted above the canvas.

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

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

```

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

```

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

The script view is displayed.

- 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	0	* UPDATE	0	x DELETE	0	* UPSERT	0
↻ Refresh	J Typecast	▼	Map drifted	Statistics	X Remove			Q LOOKUP	0
↑↓	cartId	abc	preferredProducts []	123 preferredProducts[1]: 4444		userId 123			
+	406a06af-e54f-42e9-aad8-9a36f2c7f8ca		[...]	123 preferredProducts[2]: 330		9079954			
+	5c4dc5dc-a585-41ec-8149-9133caa3a73a		[...]			9079747			
+	d79f4b3a-6c66-497a-bf59-eca12dd4f16a		[...]			9079334			
+	acde5617-af9f-4ad9-98ed-e03c793e7bd0		[...]			9079190			
+	7f2742a6-4653-4905-80ad-2d64dae04253		[...]			9078634			
+	2d3c75f6-24ce-4307-9059-902736896c61		[...]			9078467			
+	44f4c3f9-d761-4d10-bbf8-84cd45c3a461		[...]			9078115			

The data preview tab is displayed with a sample of the file contents.

- Select the + to the right of the UserProfiles source, then select the **Flatten** formatter from the context menu.

UserProfiles
Columns:
4 total

Add Source

Source settings

Output stream name *

Source type *

Dataset *

Options

Sampling * ⓘ

Multiple inputs/outputs

- Join
- Conditional Split
- Exists
- Union
- Lookup

Schema modifier

- Derived Column
- Select
- Aggregate
- Surrogate Key
- Pivot
- Unpivot
- Window
- Rank

Formatters

- Flatten
- Parse

Row modifier

+

Search

The plus sign and the Flatten formatter are highlighted.

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

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

UserProfiles's column	Name as
userId	userId
[] preferredProducts	preferredProductId

Flatten settings Optimize Inspect Data preview ●

Output stream name * UserPreferredProducts ? Help Learn more ▾

Incoming stream * UserProfiles ▾

Unroll by * [] preferredProducts ⓘ

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
abc userId	userId
[] preferredProducts	preferredProductId

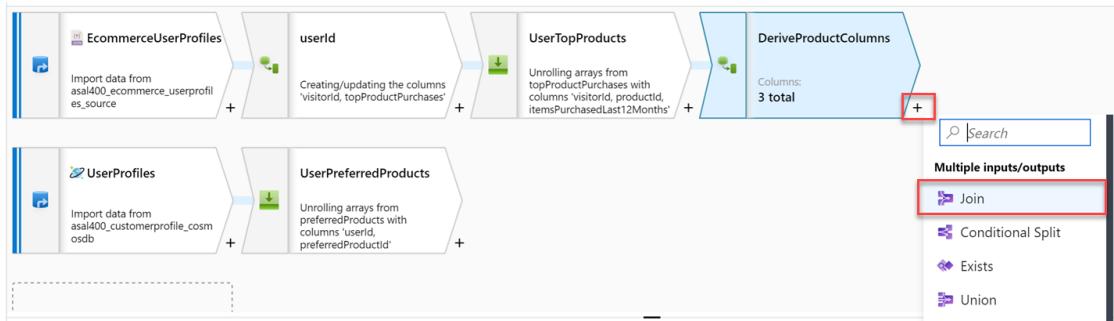
The flatten settings are configured as described.

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
			X Remove
↑↓	userId 123		preferredProductId 123
+	9079747		4235
+	9079747		3288
+	9079747		2756
+	9079334		4074
+	9079334		272
+	9079334		2164
+	9079334		414

The data preview tab is displayed with a sample of the file contents.

- 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.



The plus sign and new Join menu item are highlighted.

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

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

Left: DeriveProductColumns's column

visitorId

Right: UserPreferredProducts's column

userId

Join settings **Optimize** Inspect Data preview

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

Left stream * DeriveProductColumns

Right stream * UserPreferredProducts

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

Join conditions * **Left:** DeriveProductColumns's column
123 visitorId **Right:** UserPreferredProducts's column
123 userId

The join settings are configured as described.

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

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

Join settings **Optimize** Inspect Data preview

Broadcast Auto ⓘ Fixed ⓘ Off ⓘ

Broadcast options * Left: 'DeriveProductColumns' ⓘ
 Right: 'UserPreferredProducts' ⓘ

Partition option * Use current partitioning Single partition Set Partitioning

Partition type * Round Robin Hash Dynamic Range Fixed Range Key

Number of partitions * 30

Column values to hash on * **Column**
123 productId

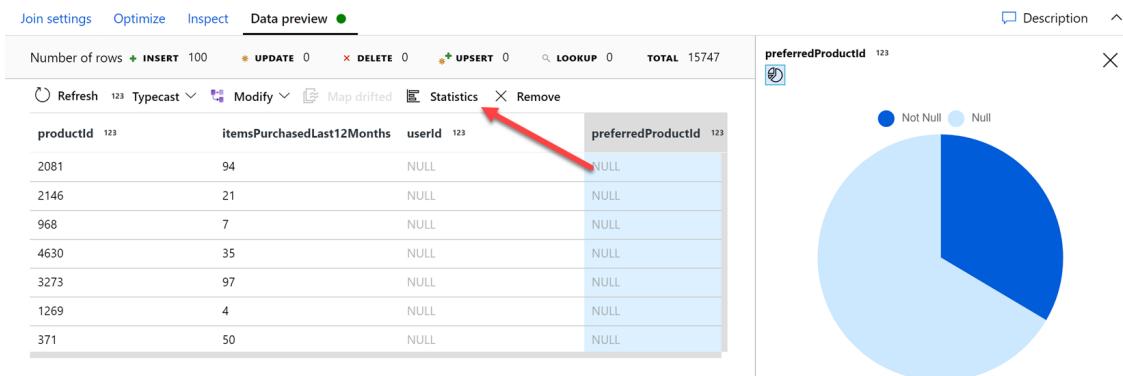
The join optimization settings are configured as described.

23. 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 3		Right: UserPreferredProducts 2		
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	

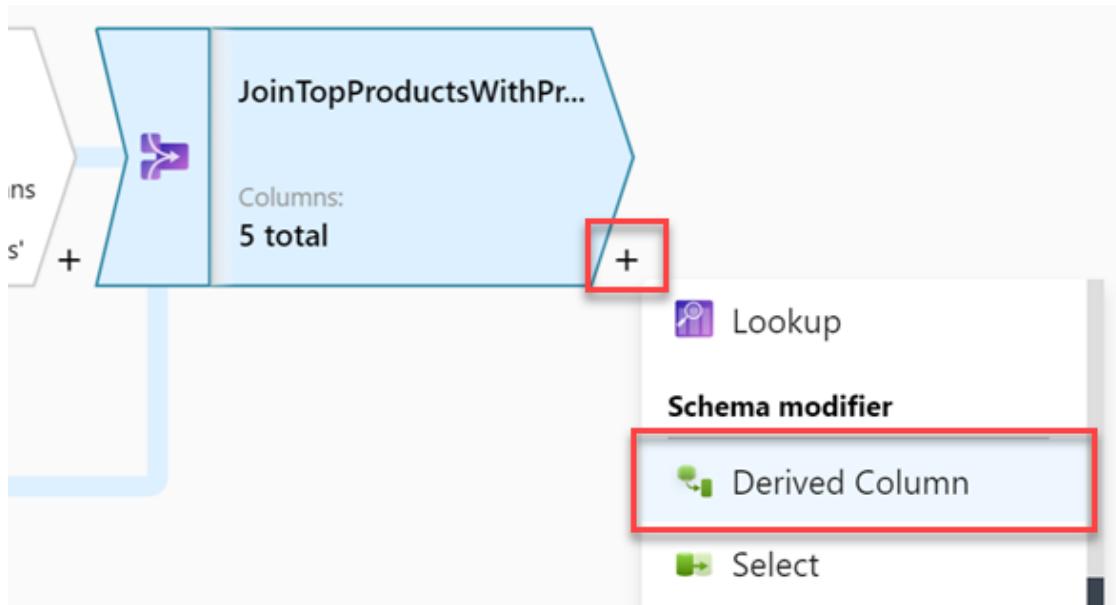
The inspect blade is displayed.

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 data preview results are shown and the statistics for the preferredProductId column is displayed as a pie chart to the right.

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



The plus sign and Derived Column schema modifier are highlighted.

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

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

Column	Expression	Description
<code>isTopProduct</code>	<code>toBoolean(iif(isNull(productId), 'false', 'true'))</code>	Returns true if productId is not null. Recall that productId is fed by the ecommerce top user products data

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

either the visitorId or userId value, depending on whether userId is null.

Derived column's settings Optimize Inspect Data preview ●

Output stream name * DerivedColumnsForMerge Learn more ↗

Incoming stream * JoinTopProductsWithPreferredProdu...

Columns * ⓘ

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

The derived column's settings are configured as described.

The derived column settings provide the following result:

Derived column's settings Optimize Inspect Data preview ● Description

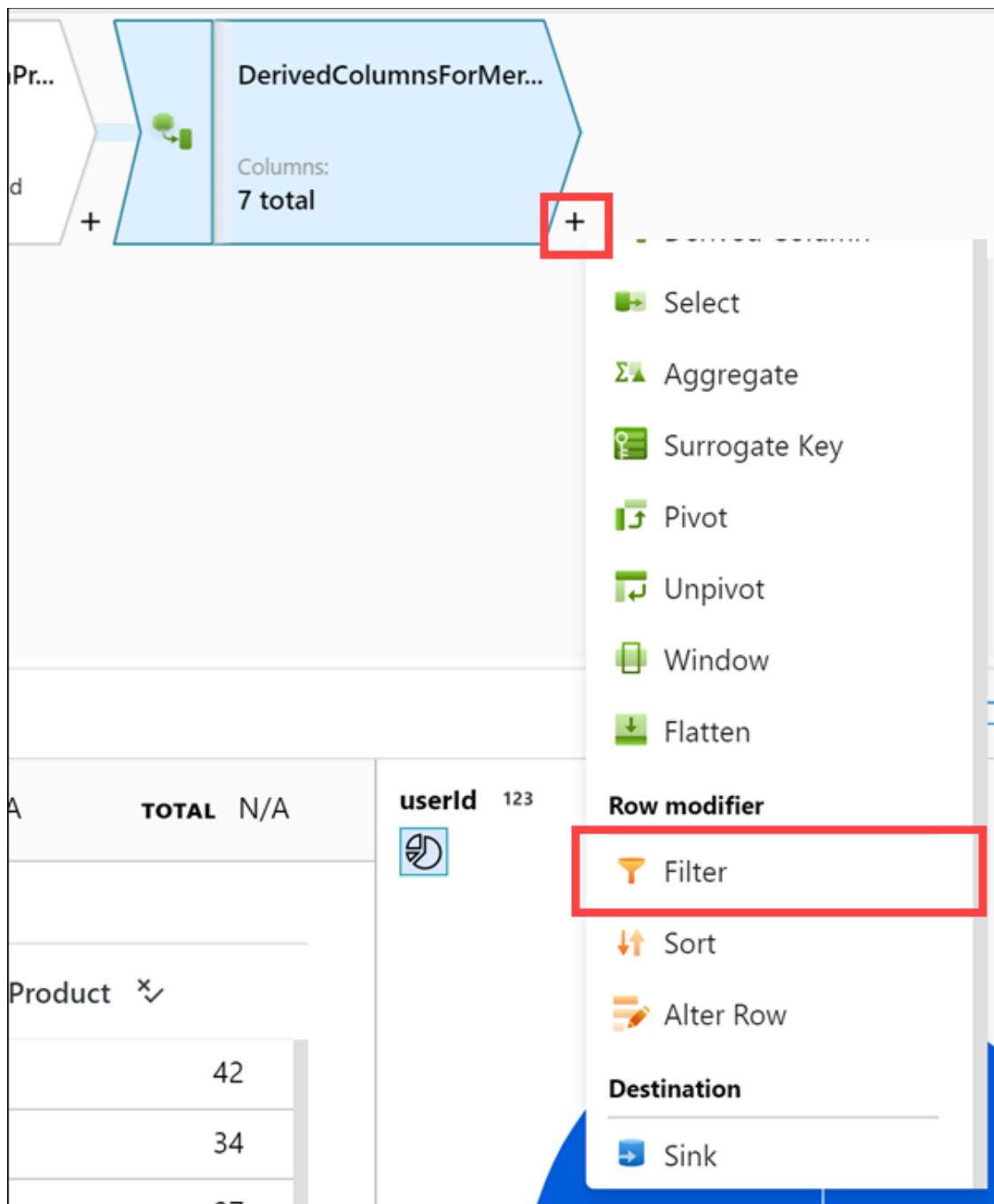
Number of rows + INSERT 100 UPDATE 0 DELETE 0 UPSERT 0 LOOKUP 0 TOTAL 15361

Refresh Typecast Modify Map drifted Statistics Remove

userId	preferredProductId	visitorId	productId	itemsPurchasedLast12Months	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	✓	✗

The data preview is displayed.

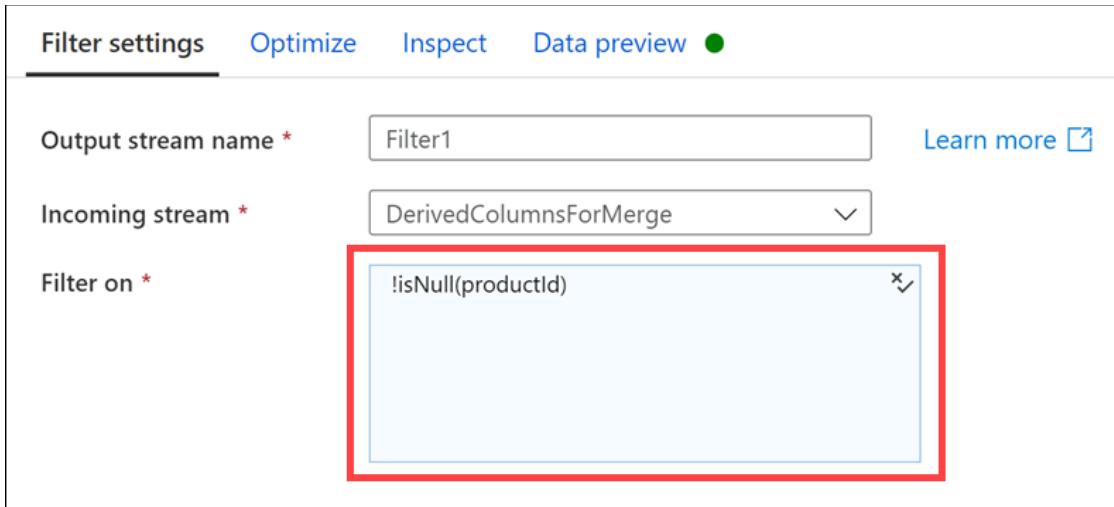
- Select the + to the right of the DerivedColumnsForMerge step, then select the **Filter** destination from the context menu.



The new **Filter** destination is highlighted.

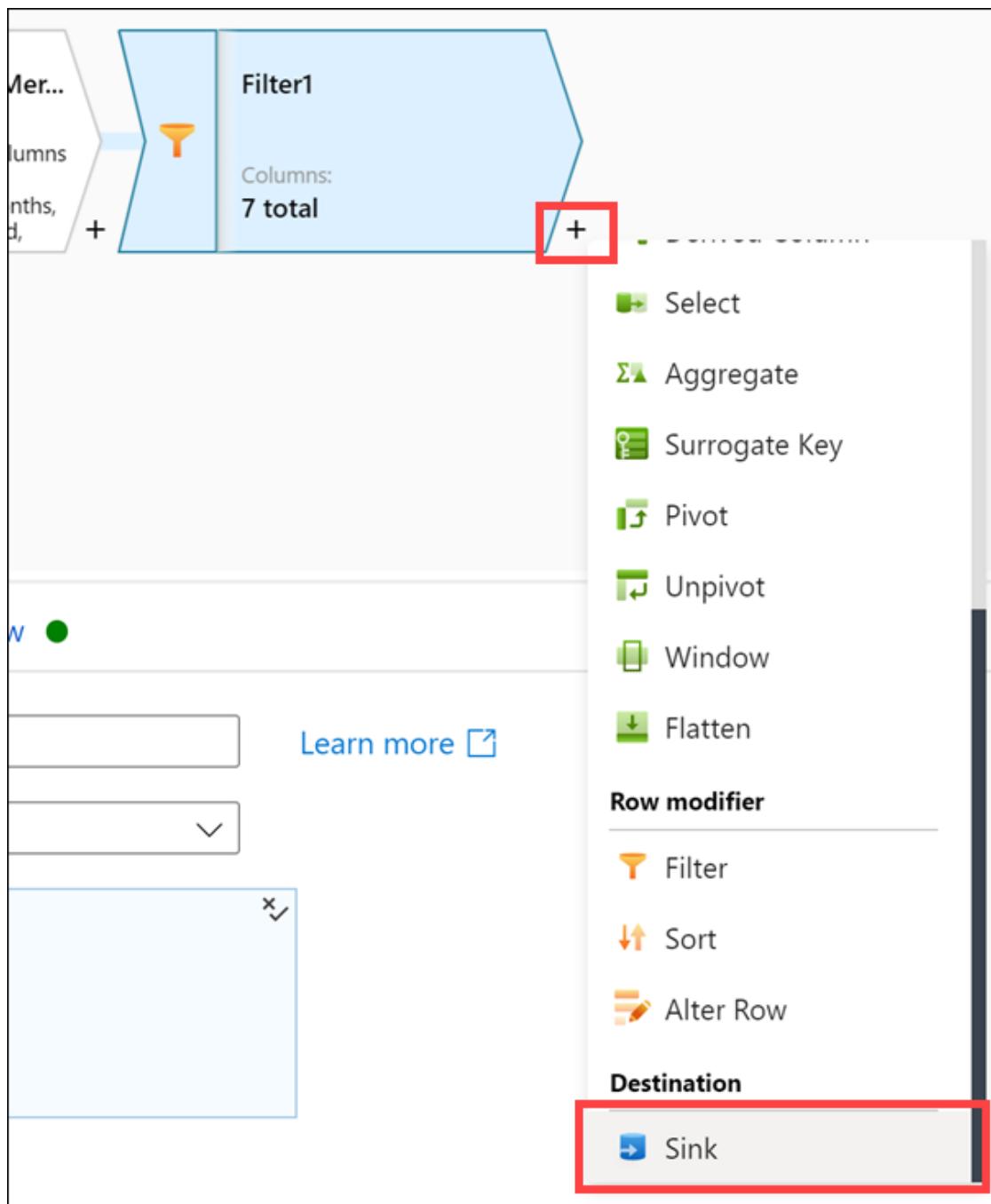
We are adding the Filter step to remove any records where the ProductId is null. The data sets have a small percentage of invalid records, and null ProductId values will cause errors when loading into the UserTopProductPurchases SQL pool table.

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



The filter settings are shown.

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



The new Sink destination is highlighted.

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

- **Output stream name:** Enter UserTopProductPurchasesASA.
- **Incoming stream:** Select Filter1.
- **Sink type:** Select Integration dataset.
- **Dataset:** Select asal400_wwi_usertopproductpurchases_asa, which is the UserTopProductPurchases SQL table.

- **Options:** Check `Allow schema drift` and uncheck `Validate schema`.

The sink settings are shown.

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

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

The settings are shown.

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

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

Input columns	Output columns
userId	UserId

productId	ProductId
itemsPurchasedLast12Months	ItemsPurchasedLast12Months
isTopProduct	IsTopProduct
isPreferredProduct	IsPreferredProduct

Sink Settings Mapping Optimize Inspect Data preview Description

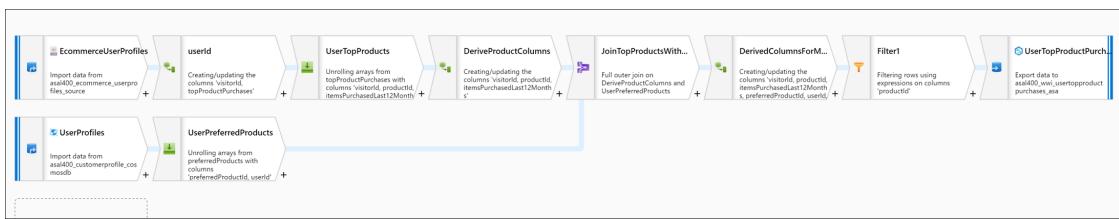
Options Skip duplicate input columns Skip duplicate output columns Auto mapping Reset + Add mapping Output format 5 mappings: All outputs mapped

Input columns Output columns

- userid → userid
- productId → ProductId
- ItemsPurchasedLast12Months → ItemsPurchasedLast12Months
- isTopProduct → IsTopProduct
- isPreferredProduct → IsPreferredProduct

The mapping settings are configured as described.

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



The completed data flow is displayed.

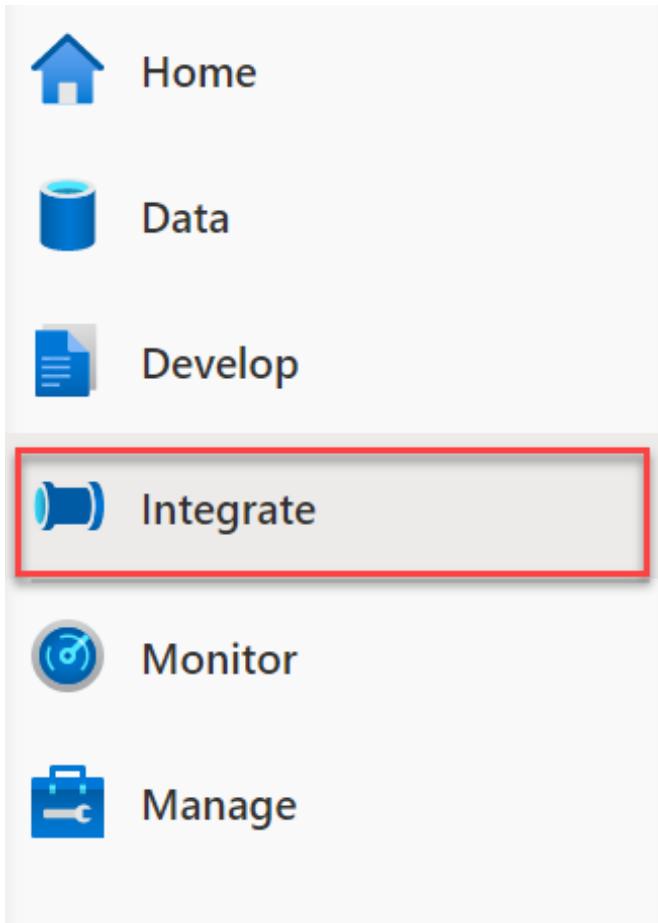
33. Select **Publish all** to save your new data flow.

Publish all is highlighted.

Task 2: Create user profile data pipeline

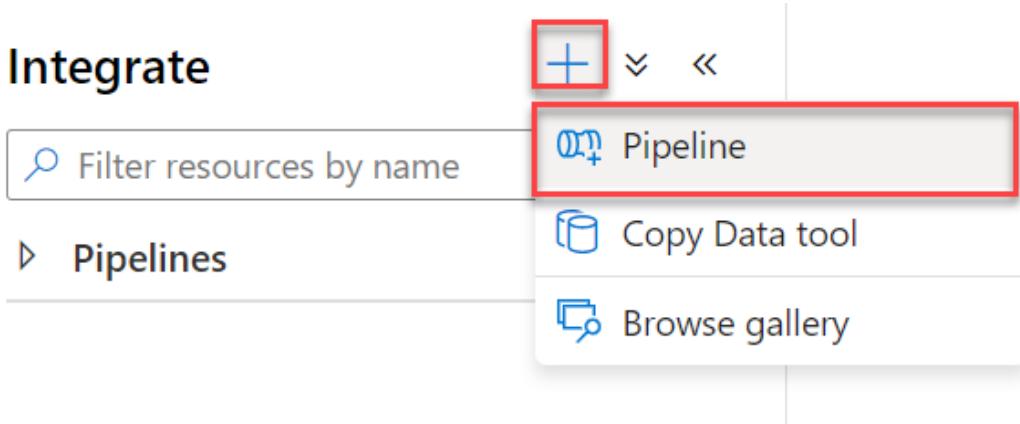
In order to run the new data flow, you need to create a new pipeline and add a data flow activity to it.

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



The Integrate hub is highlighted.

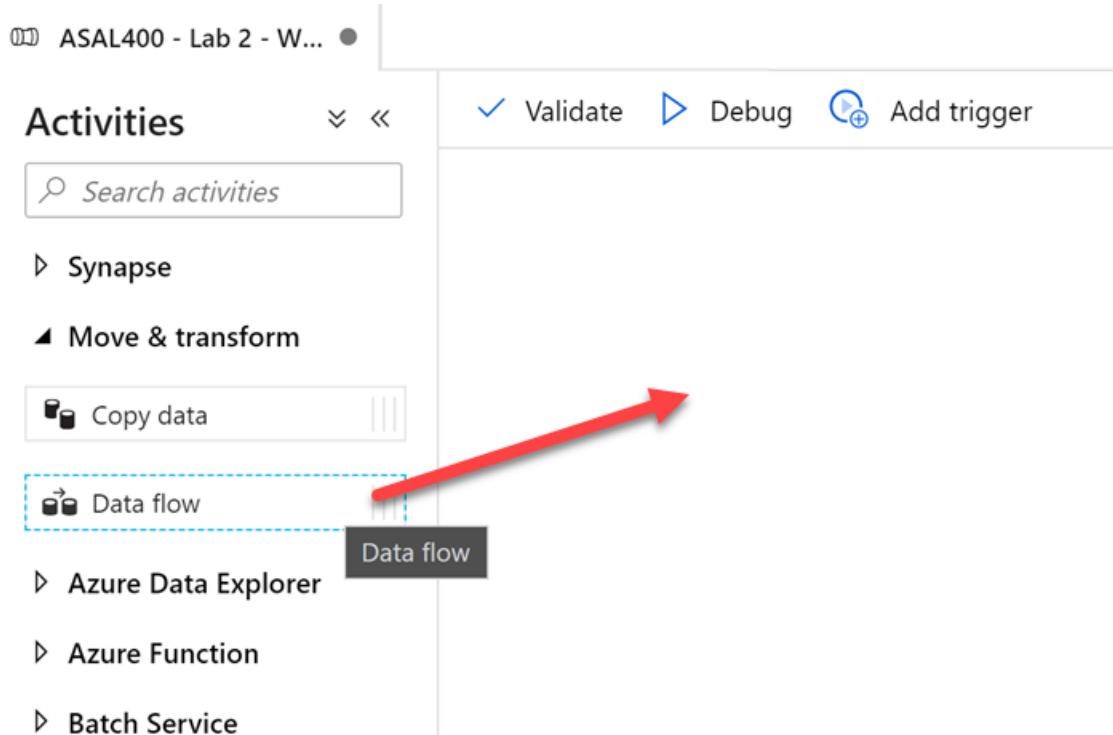
2. Select + then **Pipeline** to create a new pipeline.



The new pipeline context menu item is selected.

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.



Drag the data flow activity onto the pipeline canvas.

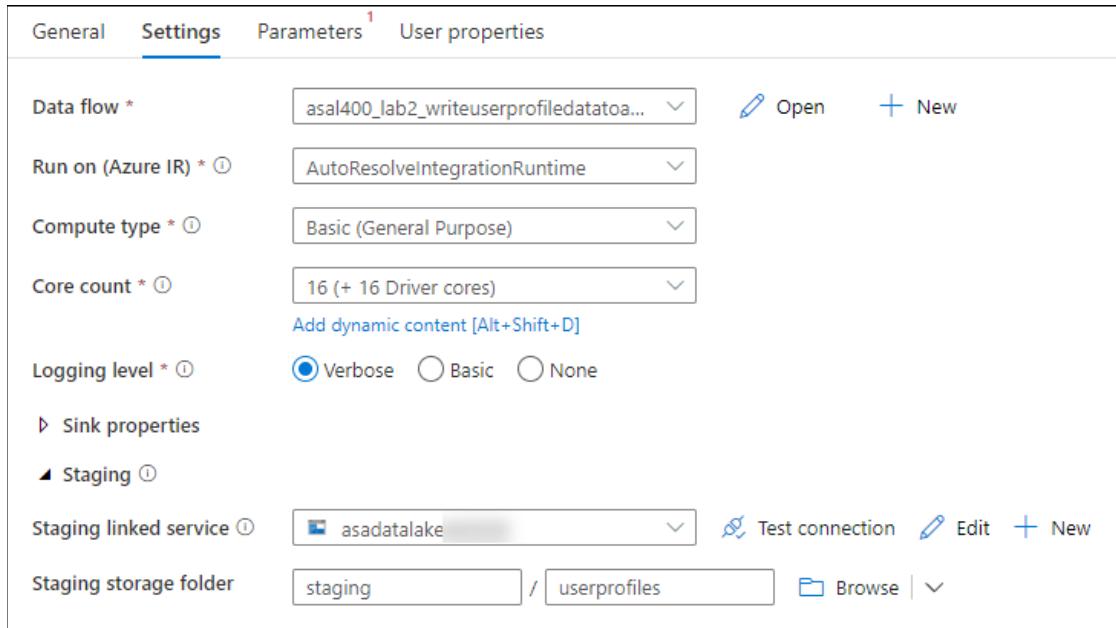
5. In the Data flow activity **General** tab, name the activity **user_profile_data**.

General Settings ¹ Parameters ¹ User properties

Name *	<input type="text" value="user_profile_data"/> Learn more
Description	<input type="text"/>
Timeout ⓘ	<input type="text" value="7.00:00:00"/>
Retry ⓘ	<input type="text" value="0"/>
Retry interval ⓘ	<input type="text" value="30"/>
Secure output ⓘ	<input type="checkbox"/>
Secure input ⓘ	<input type="checkbox"/>

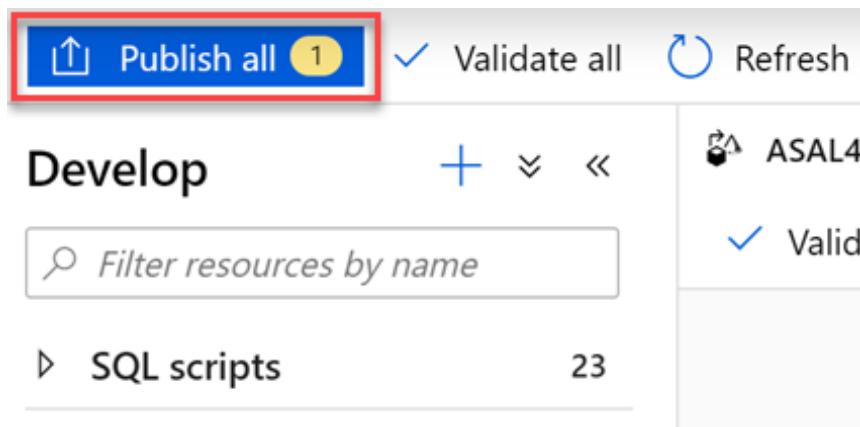
The adding data flow form is displayed with the described configuration.

6. Select the **Settings** tab, and select `asal400_lab2_writeuserprofiledata` to `asa` in the Data flow field.
7. Remaining on the **Settings** tab, then ensure `AutoResolveIntegrationRuntime` is selected for **Run on (Azure IR)**. Choose the **Basic (General Purpose)** **Compute type** and select **16 (+ 16 cores)** for the **Core count**.
8. Expand **staging** 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 mapping data flow activity settings are configured as described.

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



Publish all is highlighted.

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.

✓ Validate ▶ Debug ⚡ Add trigger ⚡ Data flow debug ✓

Trigger now

New/Edit

The trigger now menu item is selected.

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

Pipeline run

⚠ Trigger pipeline now using last published configuration.

Parameters

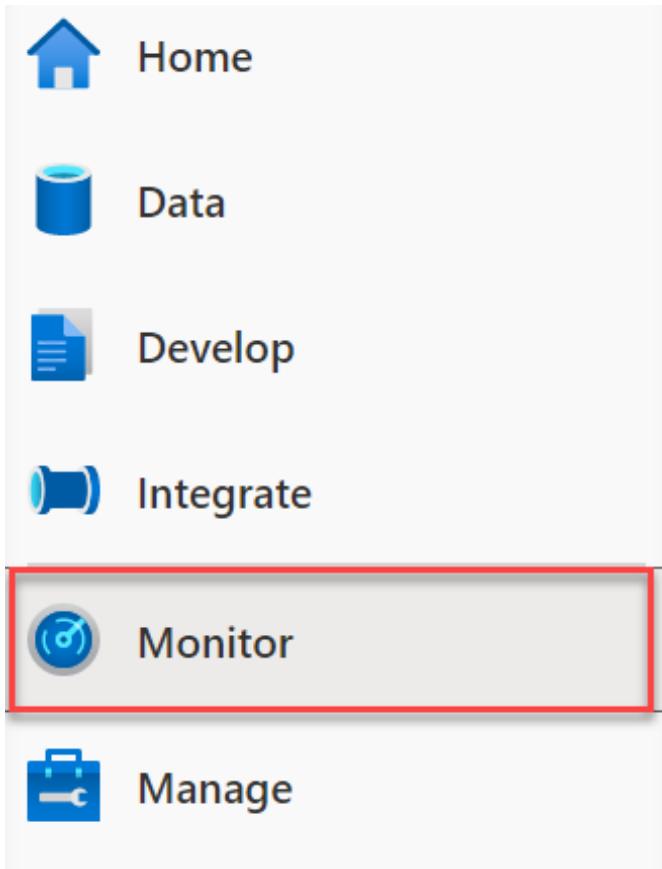
NAME	TYPE	VALUE
No records found		

OK

Cancel

The pipeline run blade is displayed.

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



The *Monitor* hub menu item is selected.

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

A screenshot of the Pipeline runs page in the Azure Data Factory portal. The left sidebar shows categories: Analytics pools, Activities, and Integration. Under Integration, 'Pipeline runs' is selected. The main area displays a table of pipeline runs. One row is shown with the following details:

Pipeline name	Run start	Run end	Duration	Triggered by	Status
ASAL400 - Lab 2 - Write User ...	9/27/21, 5:50:07 PM	9/27/21, 5:50:07 PM	00:04:00	Manual trigger	Succeeded

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

The pipeline run succeeded.

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' section of the Azure Data Factory interface. At the top, there's a 'Mapping Data Flow' card for 'asal400_lab2_writeuser_profileddataosa'. Below it, the 'Activity runs' table lists one item:

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)

A red box highlights the 'Data flow details' link under the last column of the table.

The data flow details icon is highlighted.

6. The data flow details displays the data flow steps and processing details. In our example, processing time took around 30 seconds to process and output around 1 million rows. You can see which activities took longest to complete. The cluster startup time contributed over two minutes to the total pipeline run.

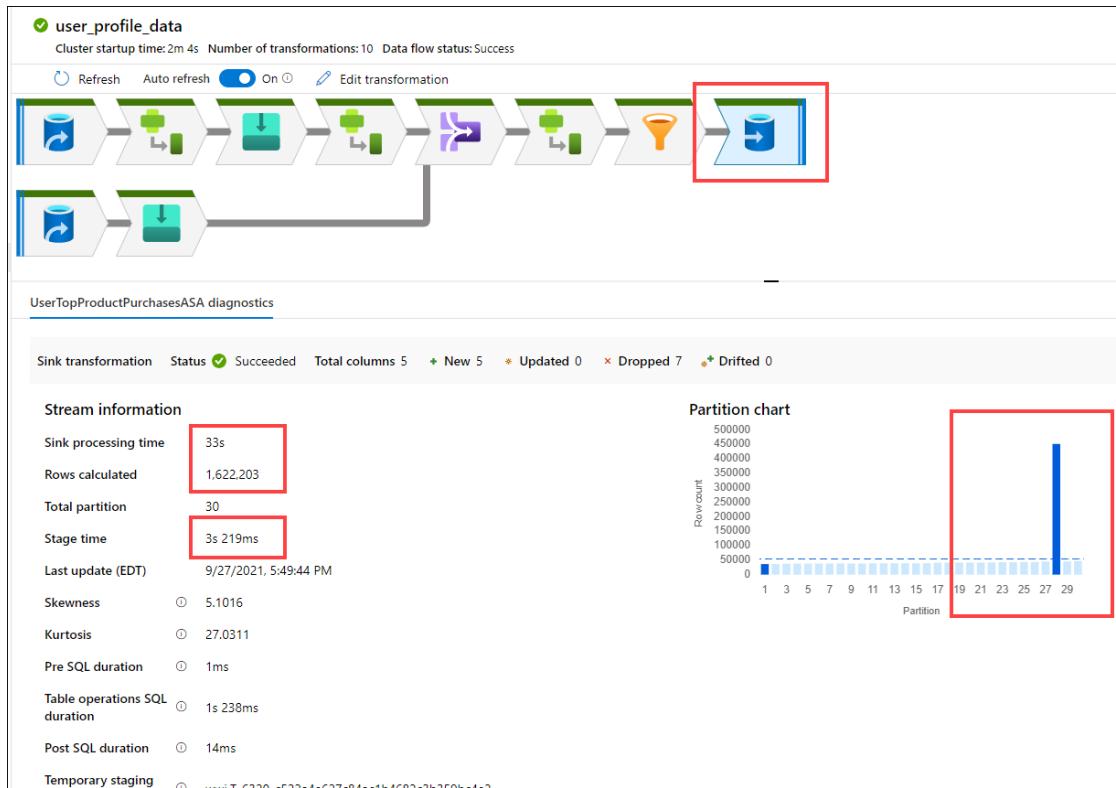
The screenshot shows the 'Data flow details' page for the 'user profile data' activity. It displays the data flow structure with various stages and transformations. Below the diagram, the 'Sinks' table provides detailed processing statistics:

Sink	Status	Processing time ↑	Highest processing time ↑	Rows written ↑	Stages	Lineage
UserTopProductPurchasesASA	Succeeded	33s	18s	1m	E	E

The data flow details are displayed.

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 3 seconds to stage the data in ADLS Gen2 prior to writing the data to the SQL table. The total sink processing time in our case was around 30 seconds. It is also apparent that we have a hot partition that is significantly larger than the others. If we need to squeeze extra performance out of this pipeline, we can re-evaluate data partitioning to more evenly spread the partitions to better facilitate parallel data loading and filtering. We could

also experiment with disabling staging to see if there's a processing time difference. Finally, the size of the SQL Pool plays a factor in how long it takes to ingest data into the sink.

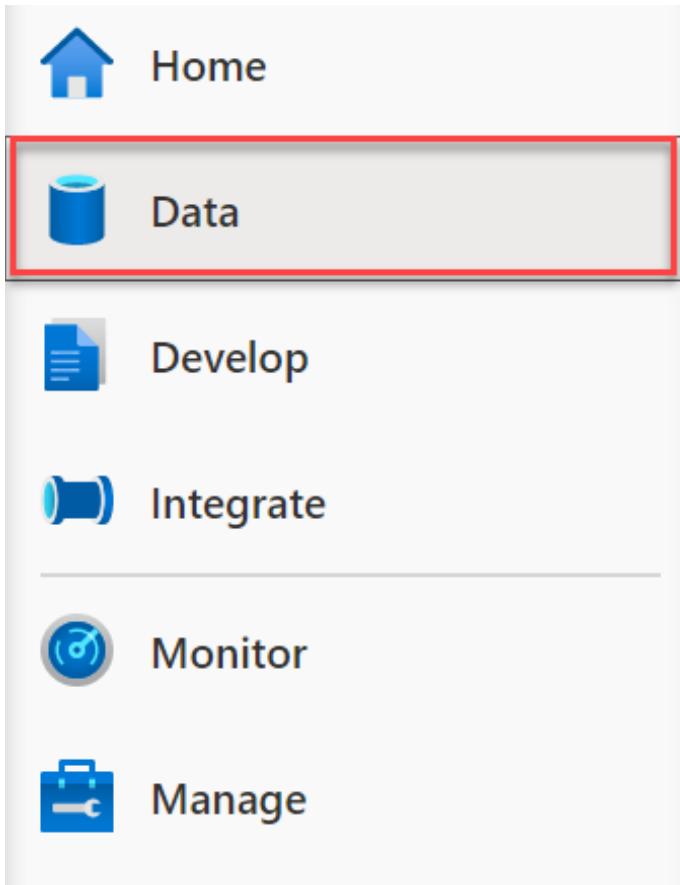


The sink details are displayed.

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**.



The Data menu item is highlighted.

2. 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. If you don't see the table listed, select Refresh above.

The screenshot shows the Azure Data Studio interface. On the left, the Data Explorer pane displays a workspace named 'Workspace' containing a database named 'SQLPool01 (SQL pool)'. Inside this database, there are several tables: 'wwi.CampaignAnalytics', 'wwi.Date', 'wwi.Product', 'wwi.Sale', 'wwi.SaleSmall', 'wwi.UserProductReviews', and 'wwi.UserTopProductPurchases'. The 'wwi.UserTopProductPurchases' table is selected and highlighted with a red box. A context menu is open over this table, listing options such as 'New SQL script', 'New notebook', 'Data flow', 'Dataset', and 'Refresh'. The 'New notebook' option is also highlighted with a red box.

The load to DataFrame new notebook option is highlighted.

3. Attach the notebook to a Spark pool.

The screenshot shows the Jupyter Notebook toolbar. The 'Attach to' dropdown menu is open, displaying a list of available Apache Spark pools. The first item in the list is 'SparkPool01' with the description 'Small, 3 to 4 nodes'. This option is highlighted with a red box. Below the dropdown, the code editor contains three lines of Scala code:

```

1 %%spark
2 val df = spark.read.sqlanalytics("SQLPool01.wwi.UserTopProductPurchases")
3 df.write.mode("overwrite").saveAsTable("default.t1")

```

The attach to Spark pool menu item is highlighted.

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 the **+ Code** button beneath the notebook cell.



The Add Code menu option is highlighted.

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

```
df.head(10)
```

```
df.createTempView("df")
```

The output should look similar to the following:

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

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

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

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

Cell 3

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

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

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

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

The cell code and output are displayed.

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

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

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

topPreferredProducts.show(100)
```



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

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

> Job execution Succeeded Spark 2 executors 8 cores



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

The cell code and output are displayed.

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

```
%%sql
```

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

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

purchased products for each user where those products are *also* identified as their favorite products, according to their user profile stored in Azure Cosmos DB.

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

```
%%pyspark
```

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

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

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

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

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

File Edit View Insert Cell Kernel Help

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

The top five preferred products are displayed per user.

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())
```

```
before filter: 1215643 , after filter: 178450.
```

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	6038
2805	5868
2486	5849
2746	5676
2119	5607

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

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* icon is highlighted.

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

user profile data Cluster startup time: 2m 45s Number of transformations: 10 Data flow status: Success

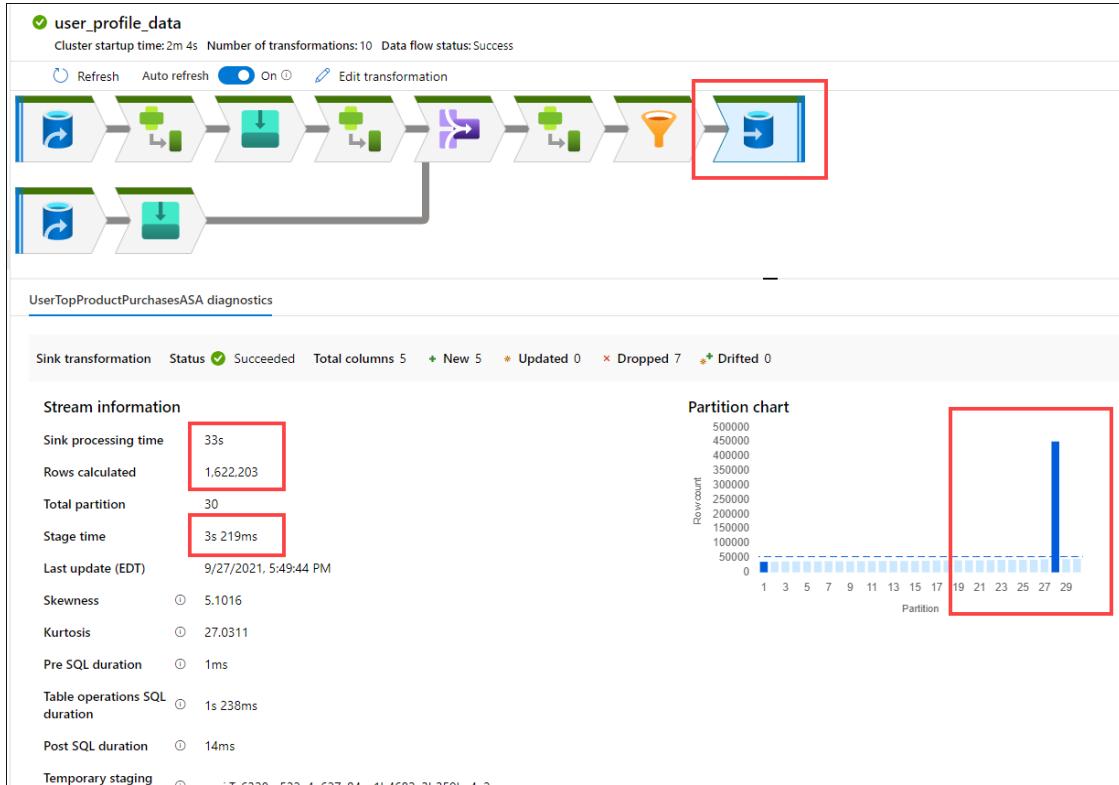
Refresh Auto refresh On Edit dataflow

Sink	Status	Processing time ↑	Highest processing time ↑	Rows written ↑	Stages	Lineage
UserTopProductPurchasesASA	✓ Succeeded	33s	18s	1m	�	�

The *data flow details* are displayed.

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.



The sink details are displayed.

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 `wvi.UserTopProductPurchases` table, then select the **Load to DataFrame** menu item under the New notebook context menu.

The screenshot shows the Azure Data Studio interface. On the left, the Data Explorer pane displays a tree structure of databases and tables. A specific table, 'wwi.UserTopProductPurchases', is selected and highlighted with a red box. On the right, the Activities pane lists various data processing options. A context menu is open over the selected table, with the 'Load to DataFrame' option also highlighted with a red box.

The load to DataFrame new notebook option is highlighted.

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

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

```
df.createTempView("df")
```

The output looks like the following:

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

```
[20934,1395,null,false,true], [20934,891,null,false,true],  
[20934,657,null,false,true])
```

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

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

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

Cell 3

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

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

► Job execution Succeeded Spark 2 executors 8 cores

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

The cell code and output are displayed.

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

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

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

topPreferredProducts.show(100)
```



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

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

› Job execution Succeeded Spark 2 executors 8 cores



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

The cell code and output are displayed.

This cell creates a new temporary view by using SQL:

```
%%sql
```

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

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

each user where those products are *also* identified as their favorite products, according to their user profile stored in Azure Cosmos DB.

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

```
%%pyspark
```

```
top5Products = sqlContext.table("top_5_products")
```

```
top5Products.show(100)
```

The output displays the top five preferred products per user:

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

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

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

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

The top five preferred products are displayed per user.

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

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

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

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

```
%pyspark
```

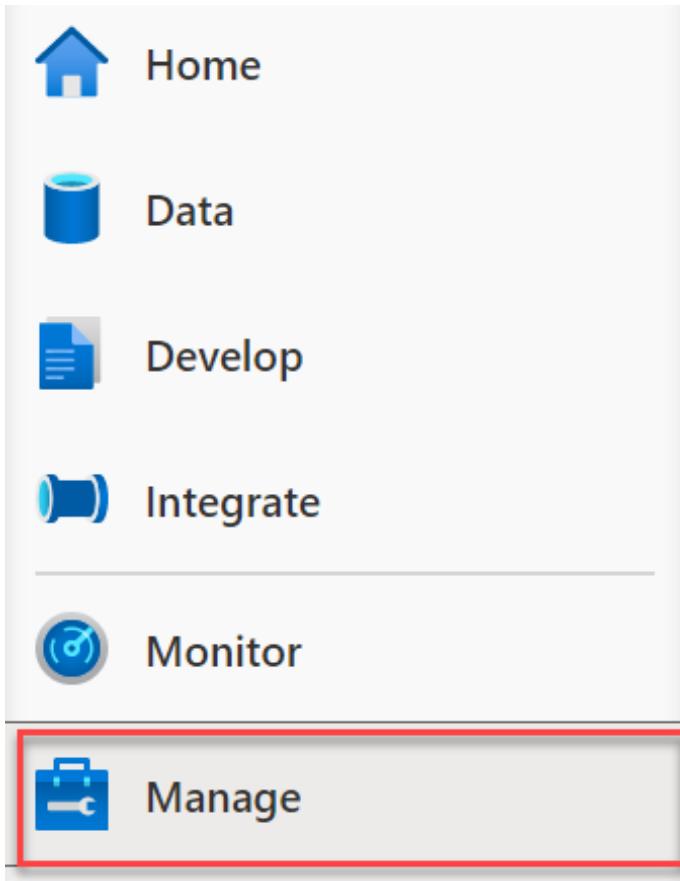
```
top5ProductsOverall =  
(top5Products.select("ProductId", "ItemsPurchasedLast12Months")  
    .groupBy("ProductId")  
    .agg( sum("ItemsPurchasedLast12Months").alias("Total") )  
    .orderBy( col("Total").desc() )  
    .limit(5))  
  
top5ProductsOverall.show()
```

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

```
+-----+-----+  
|ProductId|Total|  
+-----+-----+  
| 2107 | 91 |  
| 4833 | 83 |  
| 347 | 53 |  
| 3459 | 32 |  
| 4246 | 28 |  
+-----+-----+
```

Cleanup: Pause the dedicated SQL pool

1. Navigate to the **Manage** hub.



The Manage menu item is highlighted.

2. From the center menu, select **SQL pools** from beneath the **Analytics pools** heading. Locate **SQLPool01**, and select the **Pause** button.

A screenshot of the 'SQL pools' blade in the Azure Synapse Analytics workspace. The left sidebar shows 'Analytics pools' with 'SQL pools' selected, indicated by a red box. The main area displays a table of SQL pools. The first row is 'Built-in' (Serverless, Online). The second row is 'SQLPool01' (Dedicated, Online). A red box highlights the 'Pause' button next to the 'SQLPool01' entry. The table columns are Name, Type, and Status.

The Manage menu item is selected, with SQL pools selected from the center menu. The resume button is selected next to the SQLPool01 item.

3. When prompted, select **Pause**.