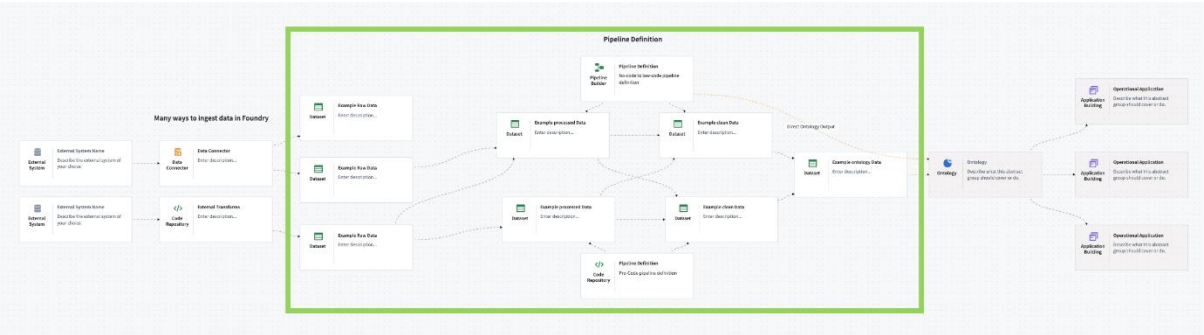
Building Your First Pipeline

Introduction

What is a data pipeline?

A data pipeline is a series of data processing steps that systematically transform raw data into valuable insights. Data pipelines clean, process, and integrate data, powering data-driven analyses and decision making within an organization.

An example end-to-end data flow in Foundry could look something like the below diagram, where the outputs of your data pipeline feed directly into the Ontology to power operational applications.



Why are data pipelines important?

In Foundry, pipelines are the backbone of the flow and transformation of data from its source form (raw) into a state that is operationally valuable. Mastering pipelines is mastering the art of turning data into actionable intelligence. It's the difference between having a sea of indecipherable data points and a well-organized dataset ready for strategic use.

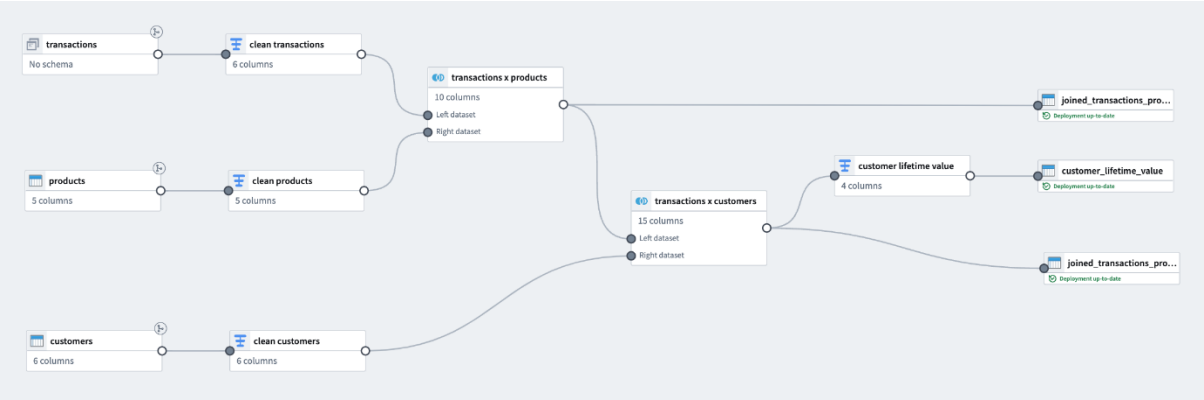
What is Pipeline Builder?

Pipeline Builder is Foundry's key low-code/no-code tool for effectively creating and managing of data pipelines. With Pipeline Builder, you can construct robust data processing workflows without the need for extensive coding. It's an intuitive interface that provides access to a suite of pre-built Spark modules, enabling you to tackle common data challenges with ease.

The Training

This training will guide you through the creation of a pipeline and the logic that powers it in Pipeline Builder. By the end of this session, you'll be equipped to take data from its raw state

and transform it into something operationally valuable. The training will also briefly touch on surrounding aspects of pipeline construction — such as branching, builds, schedules, and data validation. By the end of the training, you would have built a fully functional pipeline with minimal, if any, coding requirements, similar to the below:



Setting up your Project and Folder

Introduction

All of the resources you create in Foundry need to live inside a Project. For production use cases, these resources likely need to be shared across your organization and it is recommended you create a Project for each stage of the workflow.

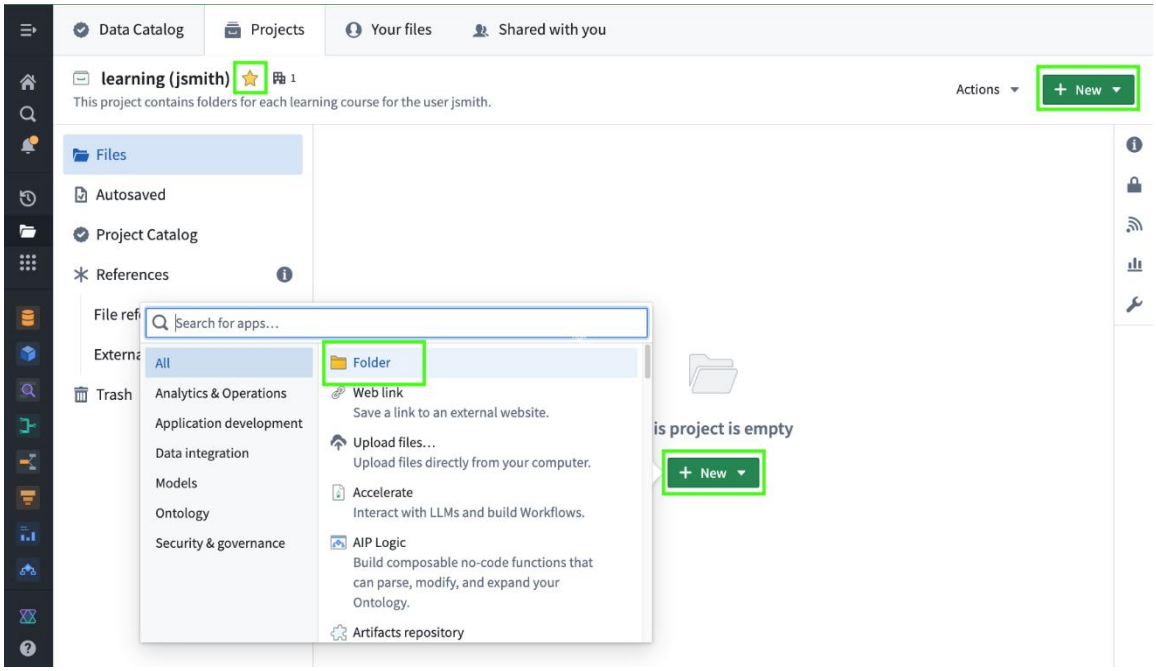
During this learning course, you will be creating resources that will be specific to your learning experience. Depending on how your environment is set up, you will want to use a naming convention to easily distinguish your resources from those of other users. We will use the convention <username> throughout this course to represent your user login id. Wherever you see <username> please replace with your own user login id.

We advise using a single Foundry Project to store all your training materials for all courses. You should then create a dedicated Folder within your training Project for each learning course. If your organization prefers a different Project set-up convention, you may need to adapt some instructions.

Create a Course-Specific Training Folder

Step 1: Create your Folder

1. In the top left, click on the star to favorite your Project. This will allow you to find it quickly later on.
2. Click on **New.**
3. Click on **Folder.**



Step 2: Name your Folder

1. Name the new folder after the current learning course
    - For example: Deep Dive: Building your first Pipeline

Data Preprocessing and Cleaning

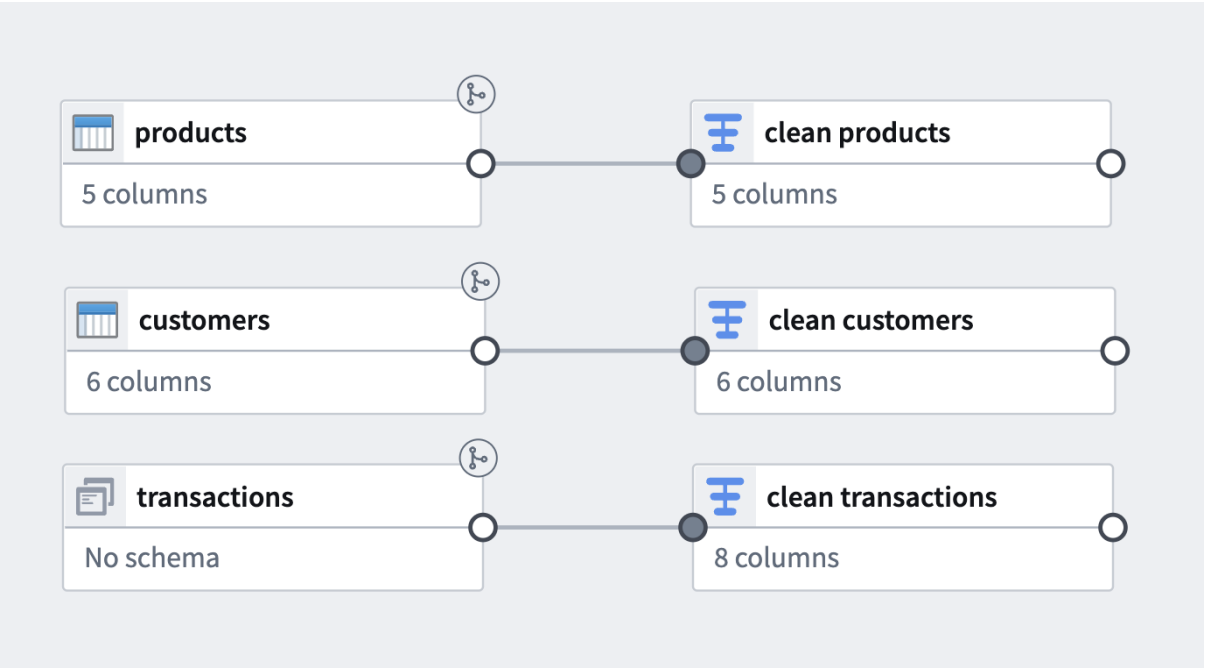
Introduction

In this module, you will focus on fundamental aspects of building a data pipeline in Foundry. The exercises in this section revolve around cleaning raw data, but the concepts and techniques discussed here are applicable beyond just data cleaning and preprocessing.

Building a data pipeline involves ensuring that the raw data coming directly from the source is consistent, accurate, and ready for analysis. This is crucial for downstream tasks, such as operational insights or further data integration. Foundry's Pipeline Builder simplifies these tasks by providing pre-built Spark modules that make it easy to address common data processing challenges, such as data type casting, file format conversion, string formatting, and many more.

In this module, you will walk through key data pipeline building concepts and understand how to leverage them using Pipeline Builder.

At the end of this module, you will have implemented a cleaning step for each of the uploaded raw datasets.



Download the Data Sources

Now that you have your training location set up, you can upload the data you'll be working with. Below are the different files you'll need to download and then upload into Foundry. Click the names of the files below to download them to your device.

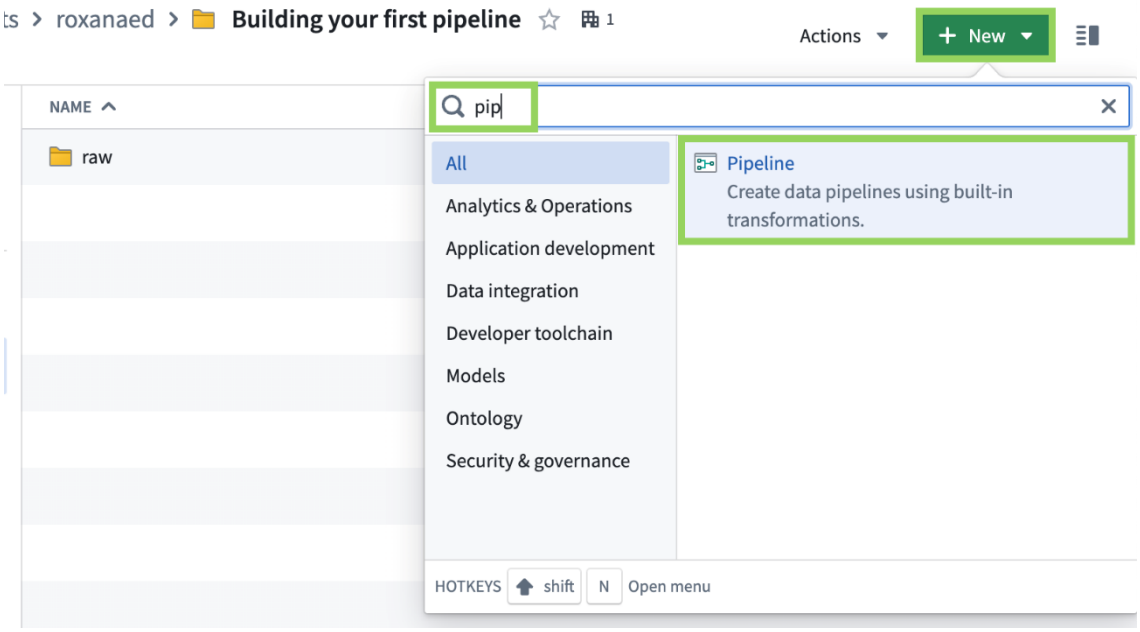
- products_raw.snappy.parquet: A raw parquet file containing information about the products sold at the store.
- customers_raw.csv: A raw csv file with basic information about the customers.
- transactions_raw.json: A raw json file containing transaction records, with details about the product, the customer, date, and quantity sold.

Create a New Pipeline

In this step, you will create a new batch pipeline using Pipeline Builder, where you will add the raw data as input, and will clean the data to be later used for analysis and further data integration. The pipeline is the central place where you will be transforming your data.

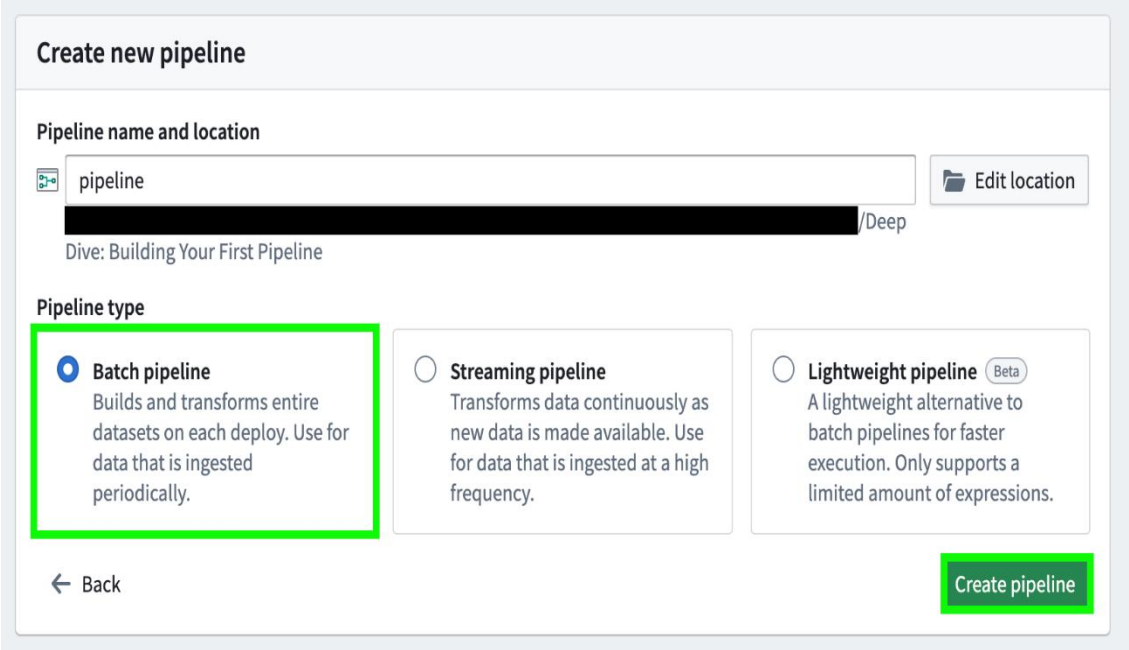
Step 1: Add a new pipeline

1. In your working directory, create a new pipeline.



Step 2: Rename and select batch

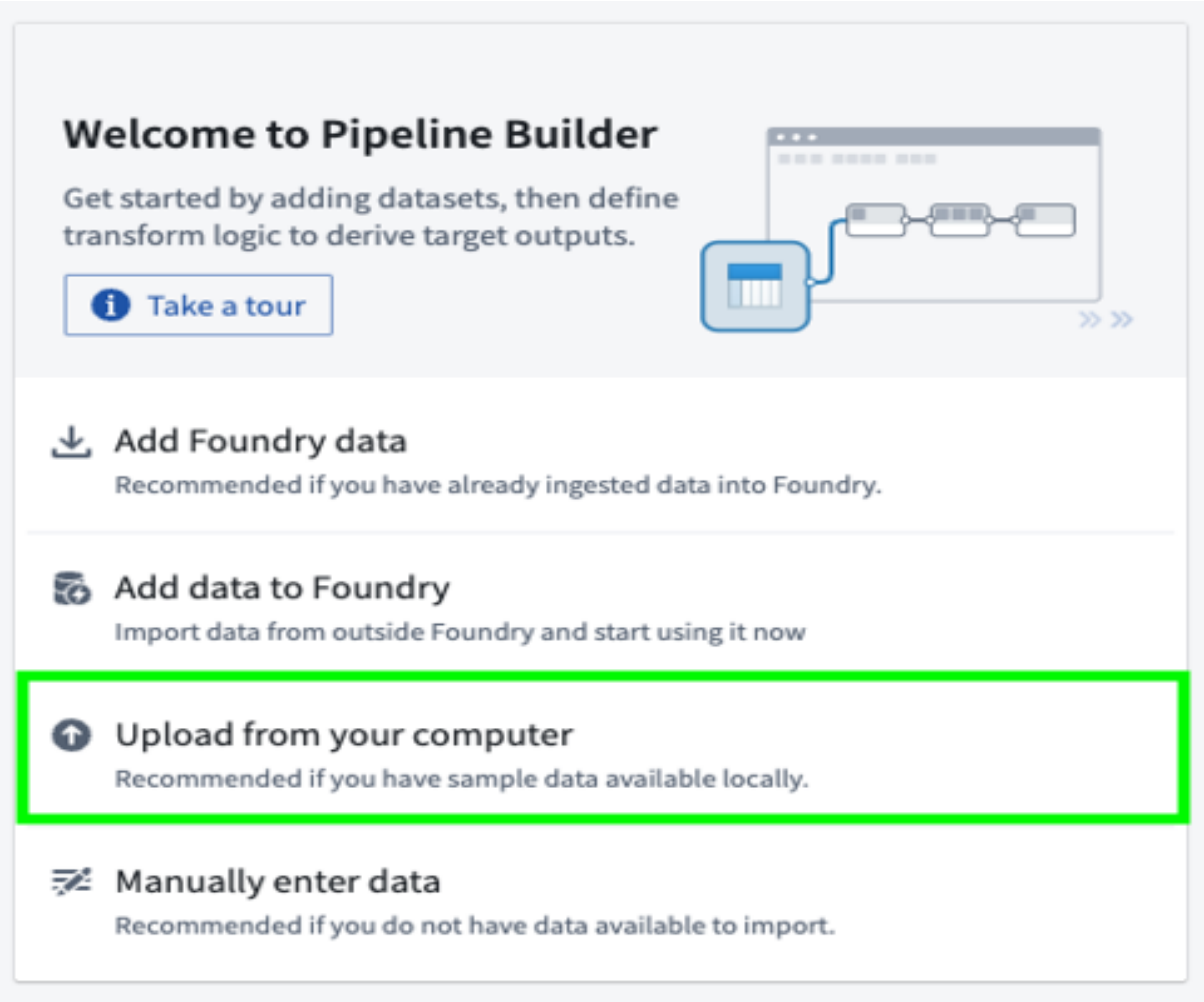
1. Name your new pipeline "*Pipeline*"
2. Select the pipeline style as *Batch pipeline*
3. Click on *Create pipeline*



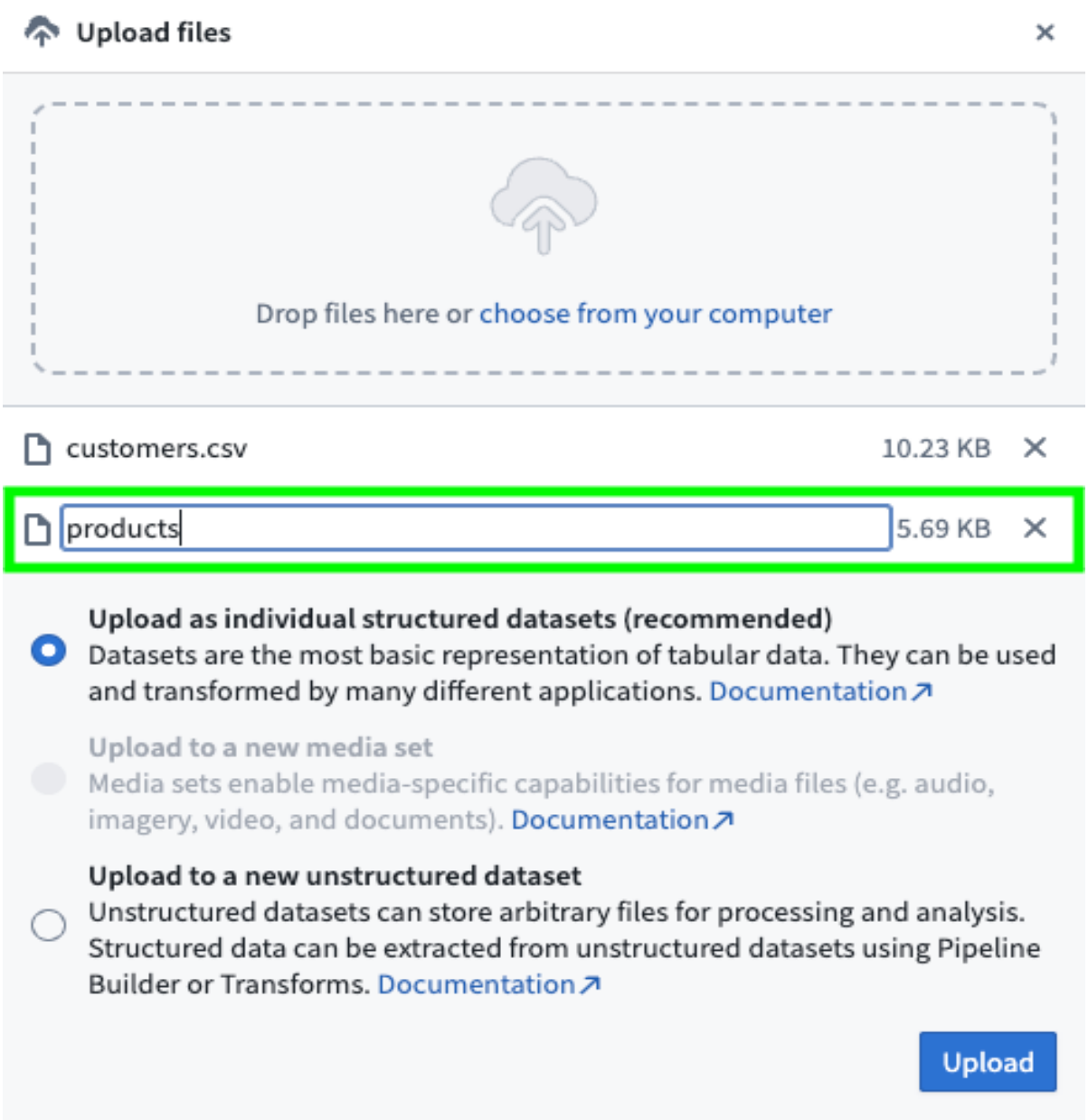
Step 3: Add the data for the pipeline

1. When opening Pipeline Builder, choose **Upload data from your computer**.
2. When the window opens, select the **products** (parquet) and **customers** (CSV) datasets from your device.

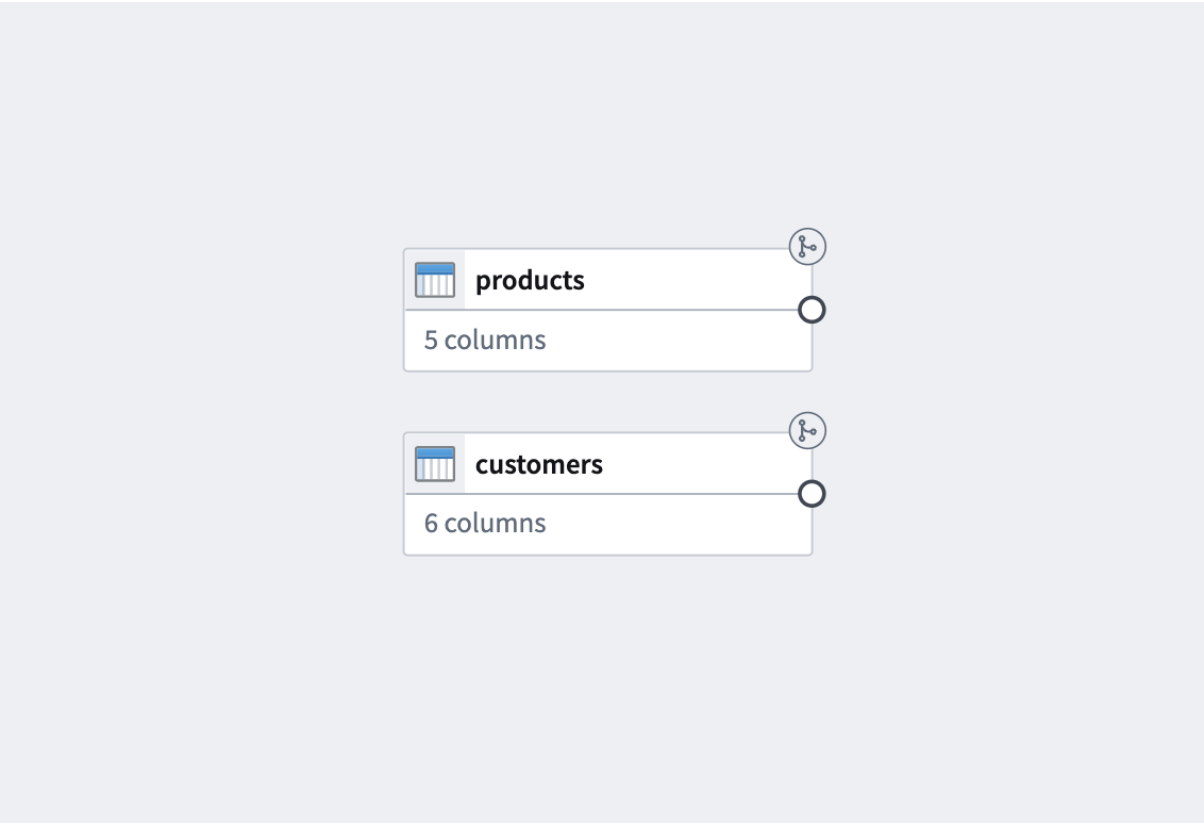
Note: We'll upload these two files together as the platform is able to infer the schema (column names) of these two data types. The import option for the json file will look slightly different in the next steps.



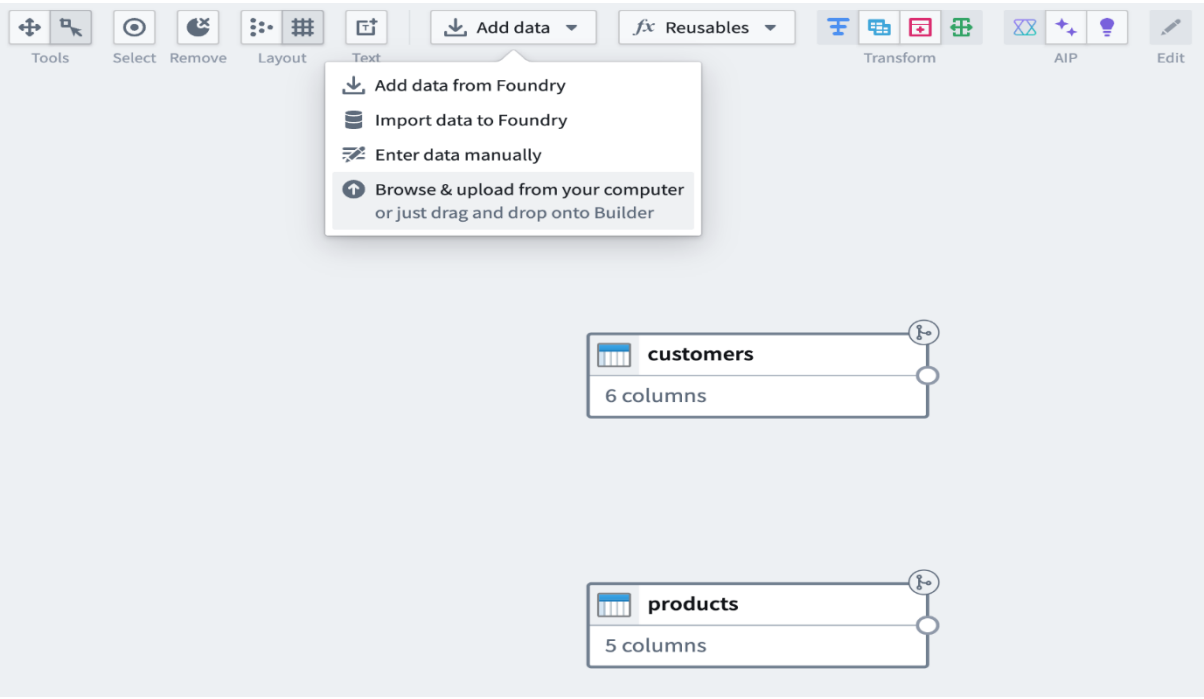
3. In the following window, rename the datasets by clicking into the existing name. Name the parquet file products and the CSV file customers.
4. Select **Upload as individual structured datasets**.
5. Click **Upload**.



6. You should now see the products and customers datasets in your pipeline represented as rectangular nodes



7. Next, we need to upload the json file for **transactions.**
8. From the top left bar, select **Add Data** and **Browse & upload from your computer**

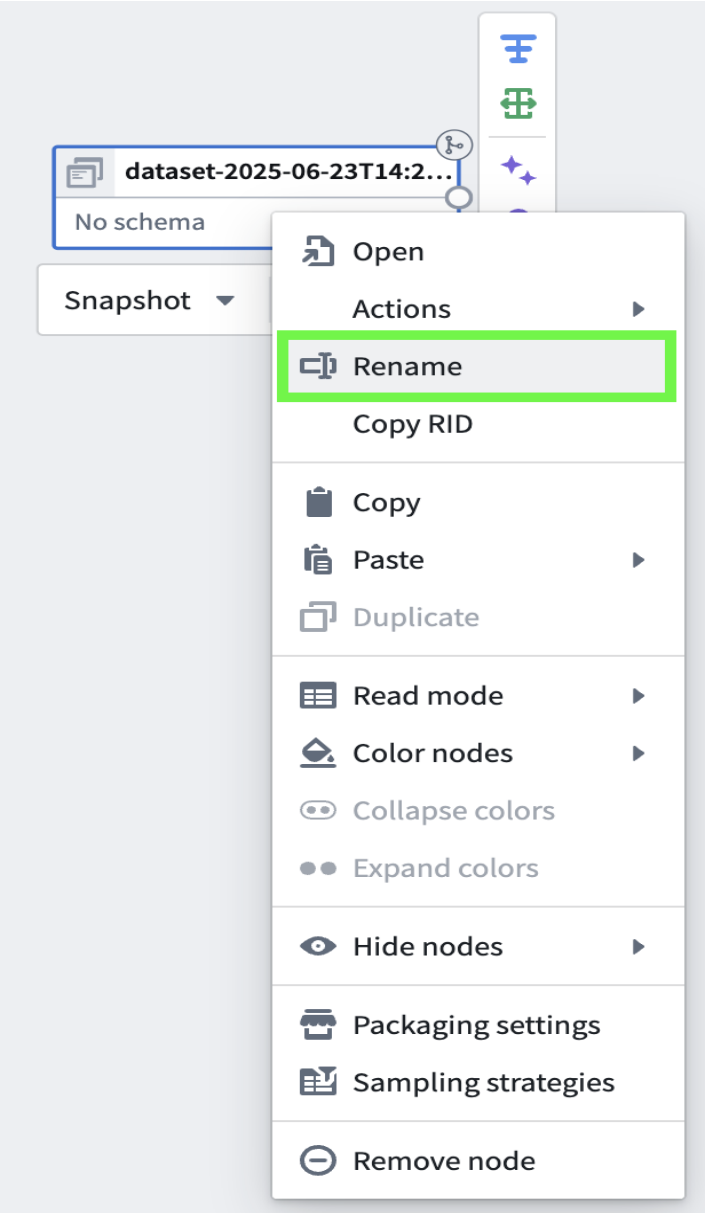


9. Find and select the json file for transactions on your device
10. Rename the file to transactions

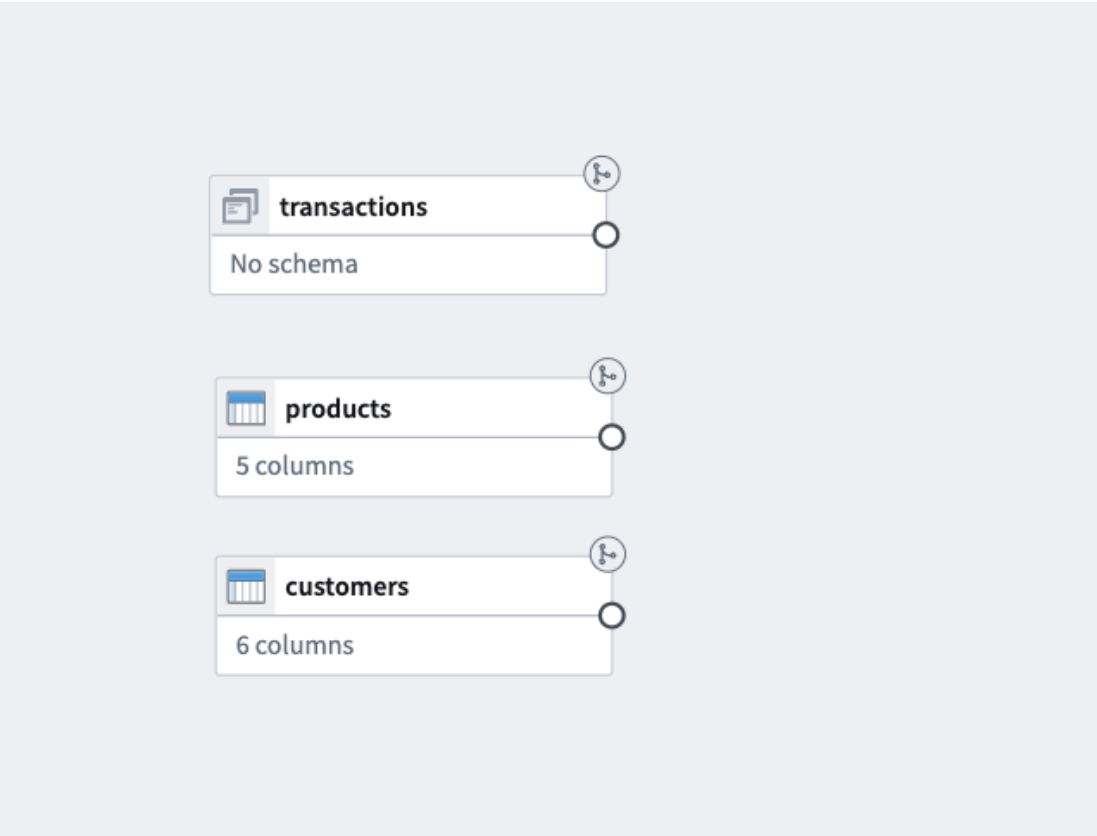
11. Select **Upload to a new unstructured dataset**
12. Click **Upload**

Note that the 'transactions' name doesn't carry over to the name of the node representing the dataset in your pipeline as we need to complete some extra steps to extract the schema of the json file into a proper dataset. For now, we will manually rename the node for naming consistency.

13. Right / control-click on the node representing the transactions data
14. Click **Rename** and input transactions



15. You should now have three nodes on your screen for customers, products, and transactions



Basic Transforms: Clean the Products Dataset

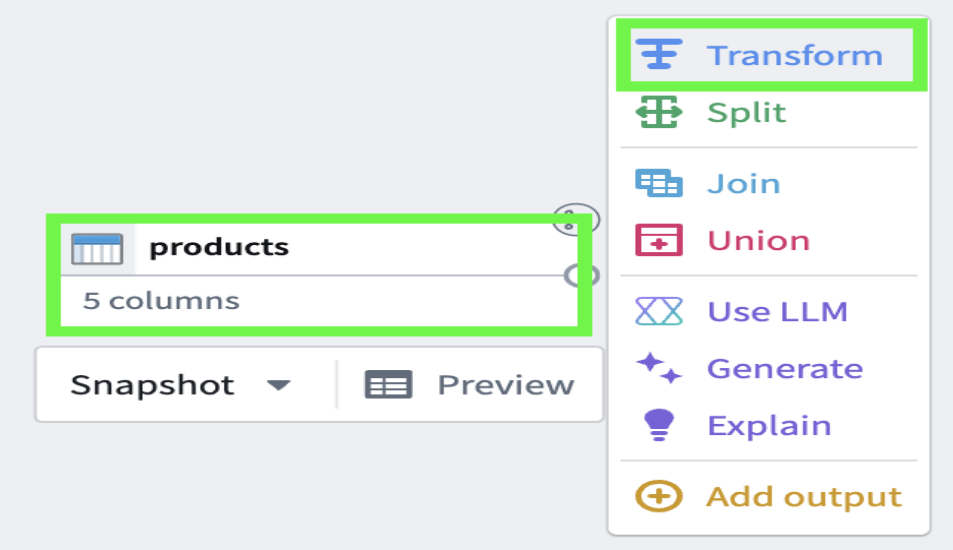
You are ready to start transforming your data right away. Typically, raw data is likely to present certain challenges, stemming either from the ingestion phase or directly from anomalies in the storage system of the source. The products dataset has some minor issues which can be addressed with pre-built Pipeline Builder modules in a cleaning transform. These out-of-the-box transforms make it easy to perform common data cleaning tasks quickly and efficiently.

In this section, you will solve the following issues:

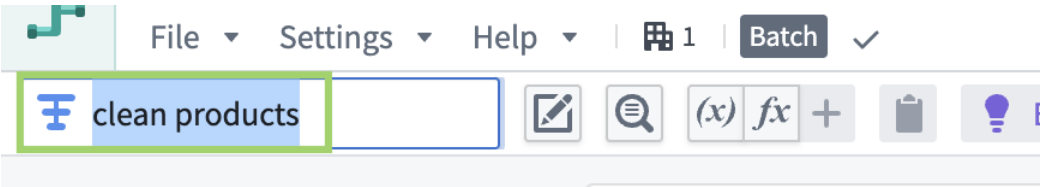
- The product name column has trailing whitespaces.
- The price column has been ingested as a string, but it should be a double data type.

Step 1: Create a new transform for raw products dataset

1. Select the **products** dataset and choose **Transform**.

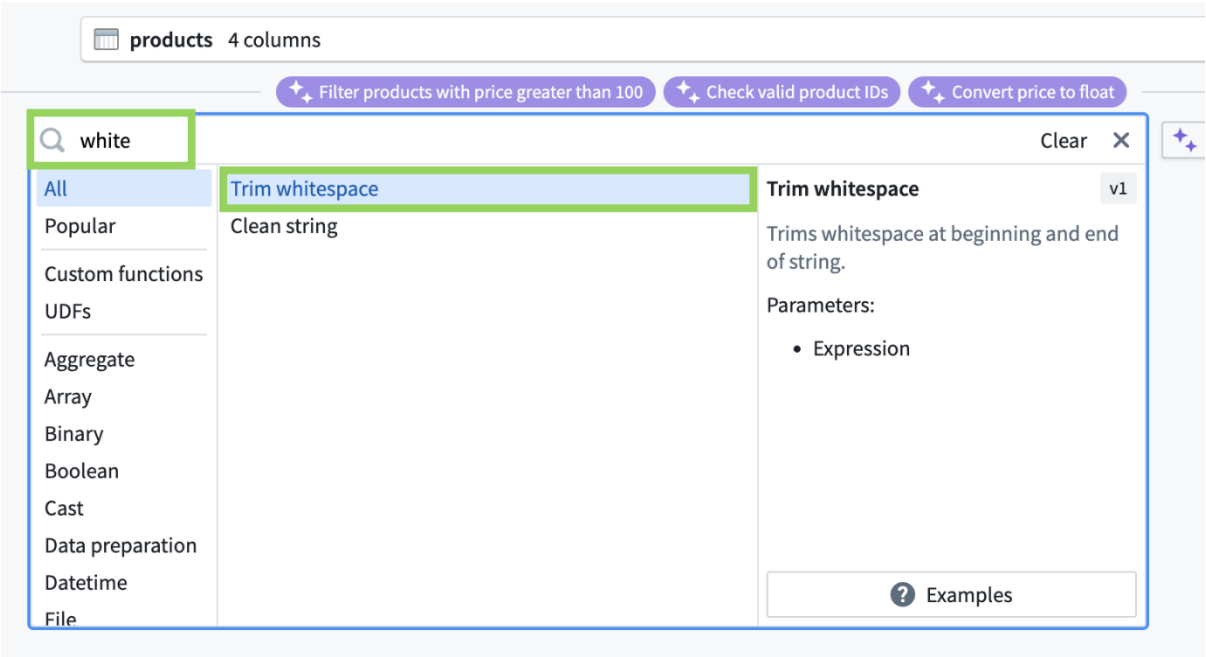


2. Give the transform a relevant name.

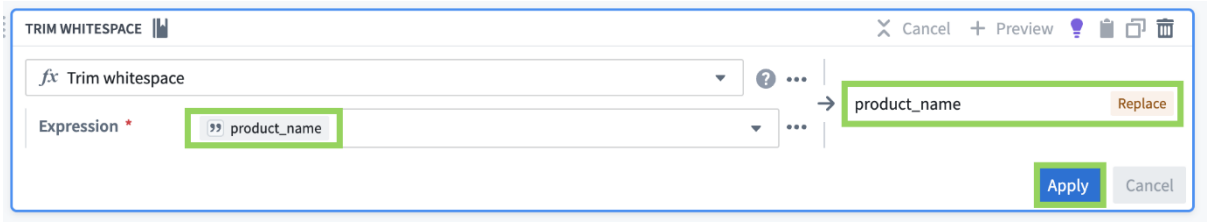


Step 2: Remove whitespaces in product name

1. Search for the **Trim whitespace** transformation.

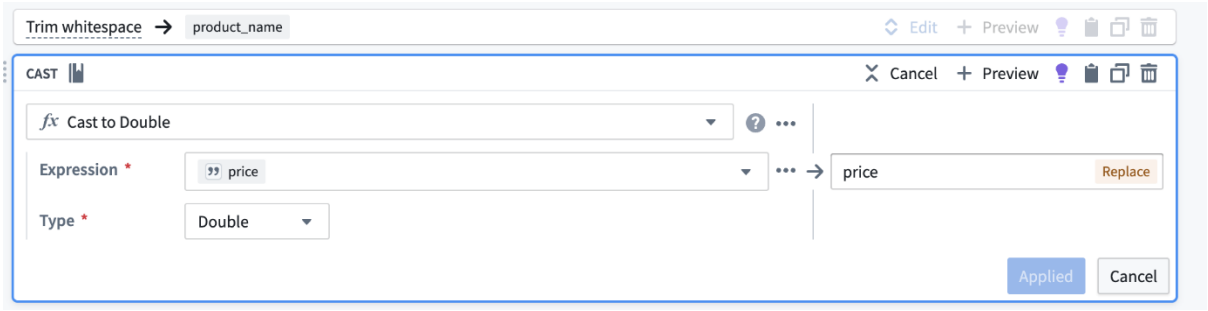


2. Select column **product_name** and **Apply**.



Step 3: Cast price to double

1. Search for the Cast transform, select it, and cast the **price** column to **double**.



Step 4: Apply all changes

1. Click **Apply all changes.**
2. Click **Close** to take you back to the page displaying dataset nodes.

Advanced Transforms: Clean the Customers Dataset

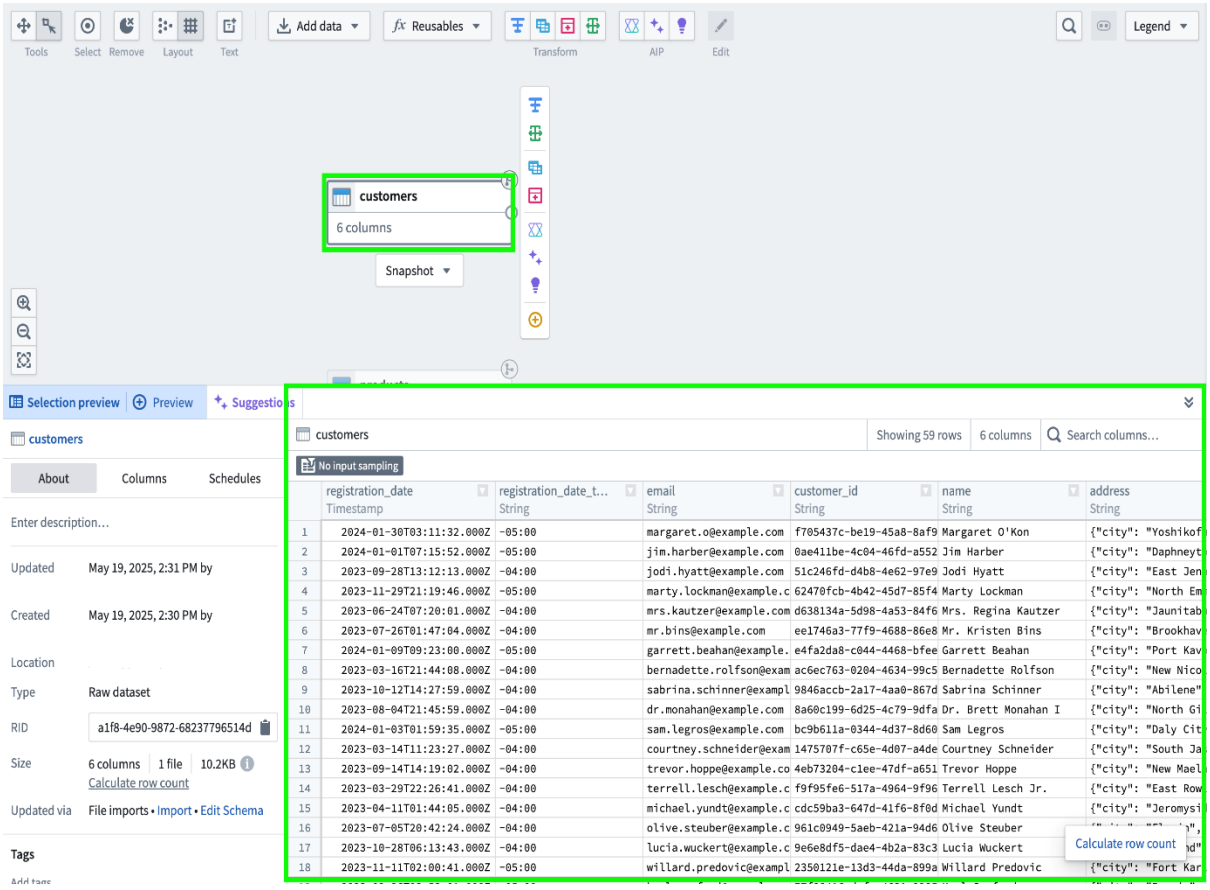
In this section, you will analyze the shape of your data to identify specific cleaning and processing requirements. Datasets can vary in complexity, and understanding the structure and characteristics of your data is crucial for determining the appropriate cleaning steps.

Preview in Pipeline Builder is a great tool to identify aspects of your data that need to be cleaned or transformed. By utilizing this preview functionality, you can gain insights into potential issues or inconsistencies within your data, allowing you to make informed decisions about the necessary cleaning processes.

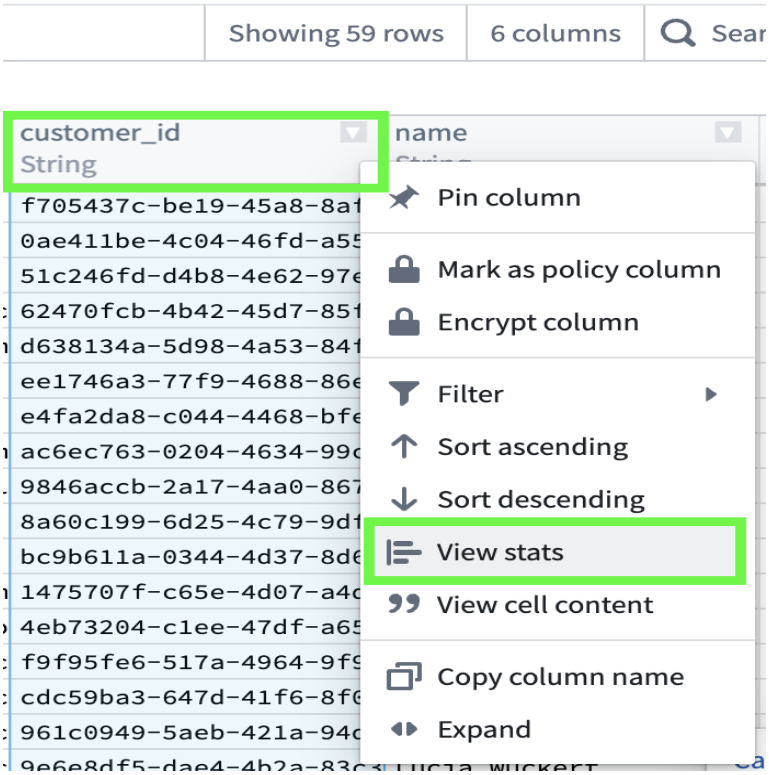
Additionally, you will walk through less straightforward transformations, to mimic real world examples whereby raw data can come in various shapes. This section will allow you to deepen your understanding of the Pipeline Builder transform boards, by constructing complex transformations and nesting the results.

Step 1: Explore customers dataset using Preview

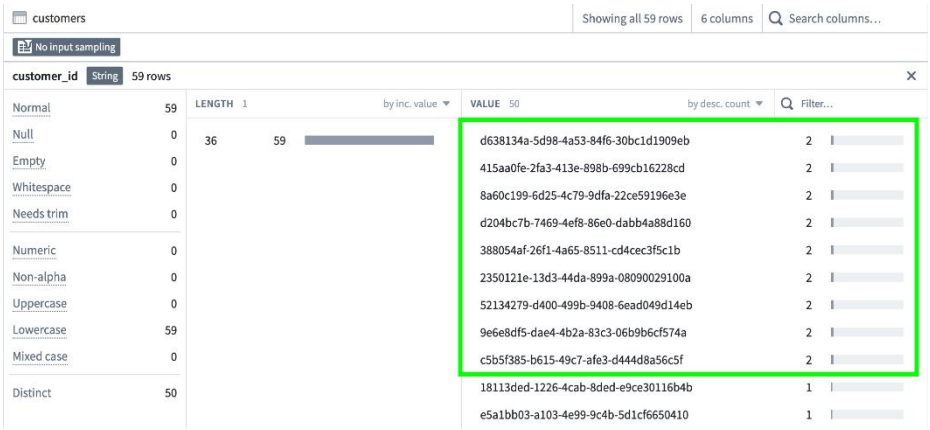
1. Click on the **customers** dataset to open Preview.



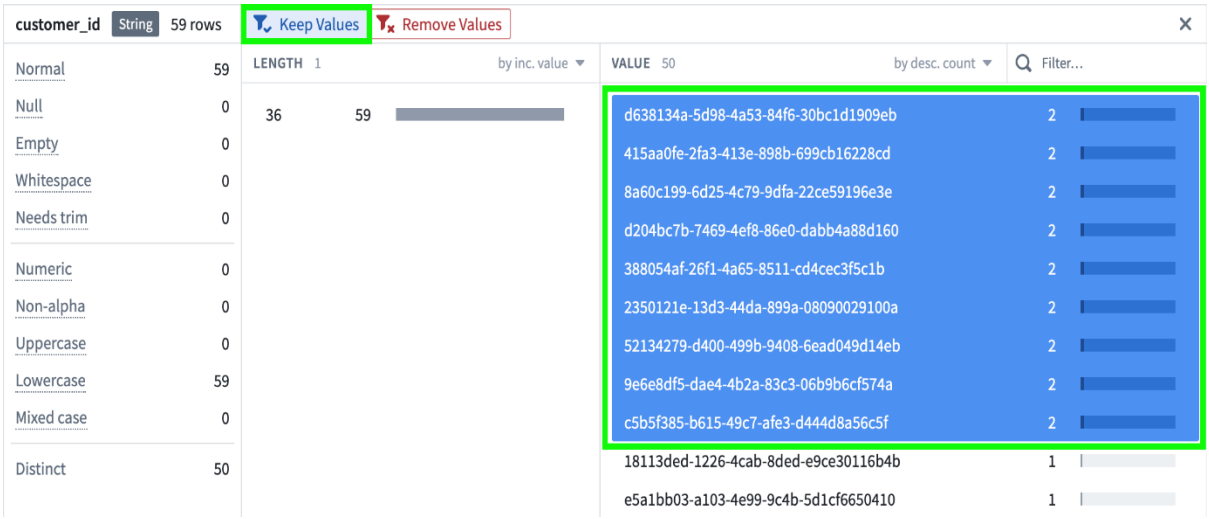
2. View stats on the **customer_id** column.

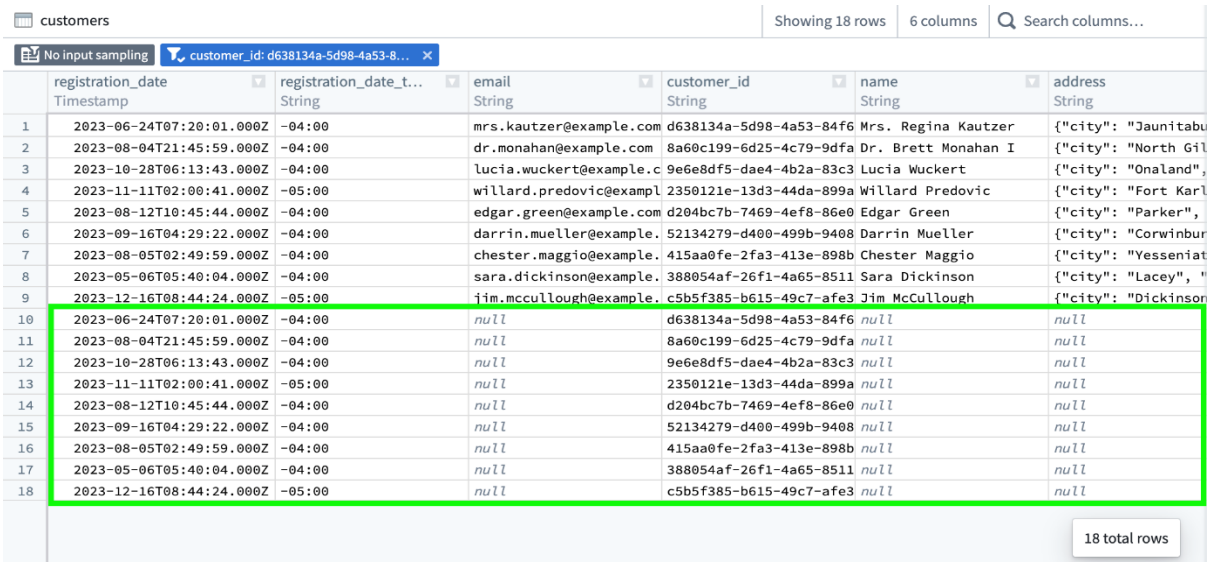


3. Notice the values of some of the primary key are not unique, meaning that there are duplicates in the data.



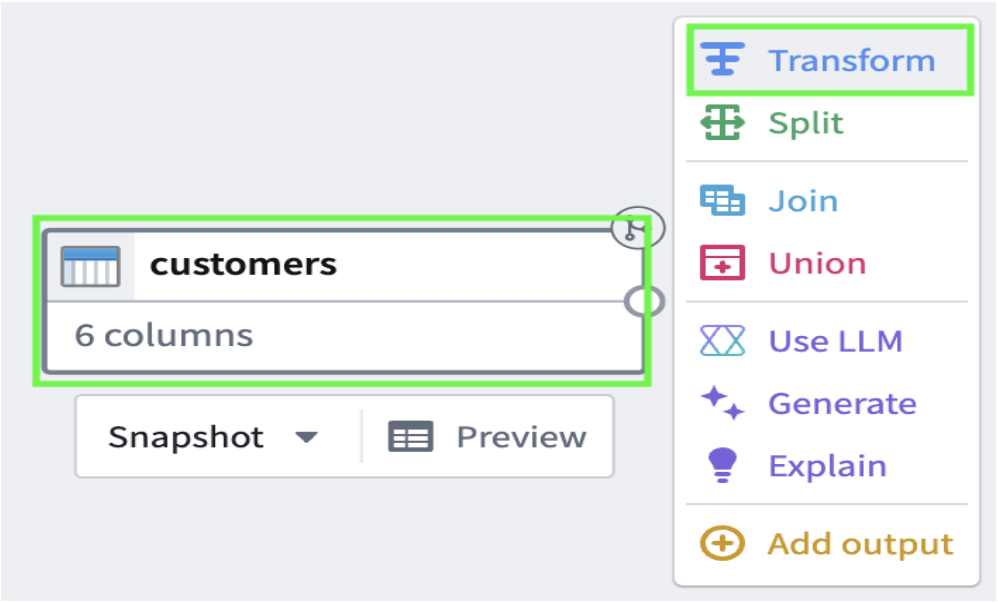
4. Filter to those duplicated values to decide on the deduplication strategy. You can click one at a time or use shift + click to highlight multiple rows at once.



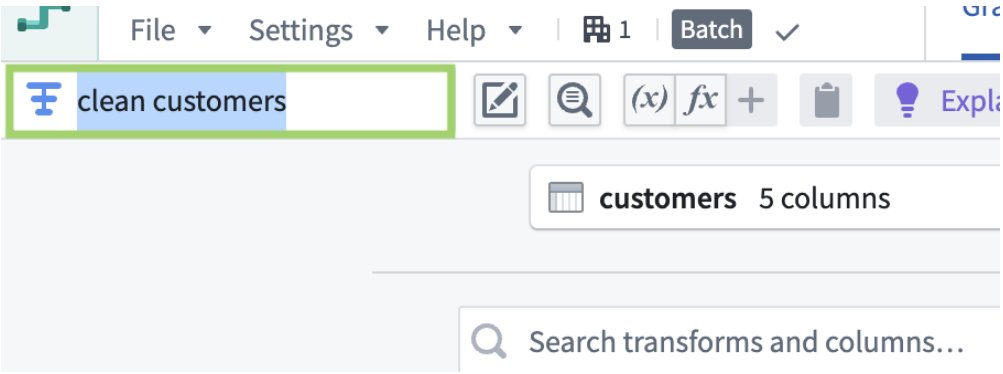


Step 2: Drop duplicates

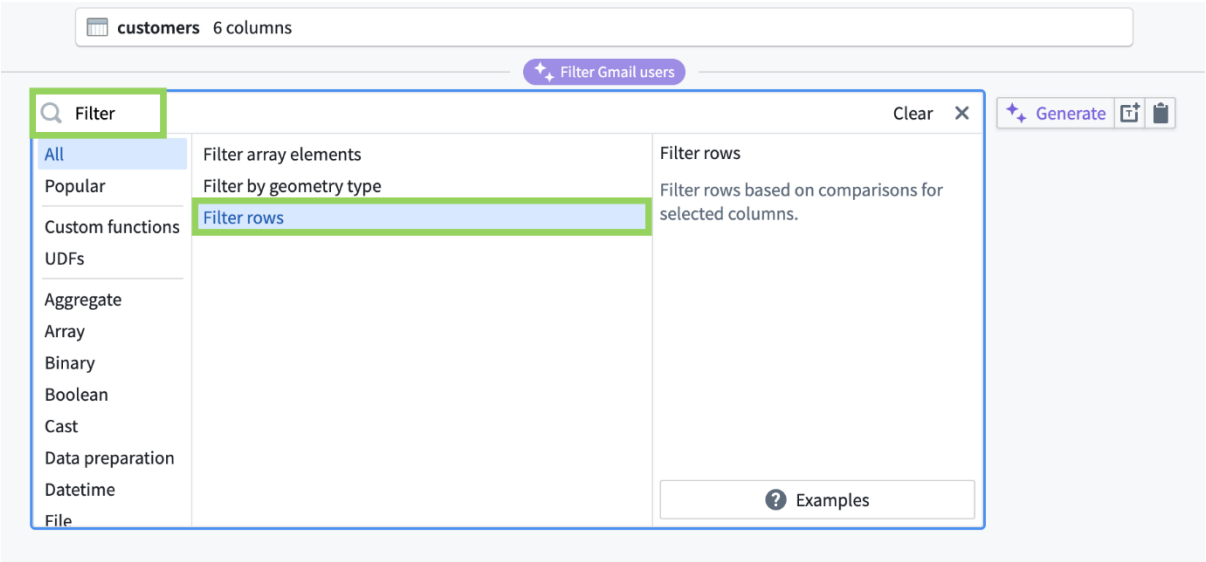
1. Similarly to the previous step, create a new transform to clean the raw **customers** dataset.



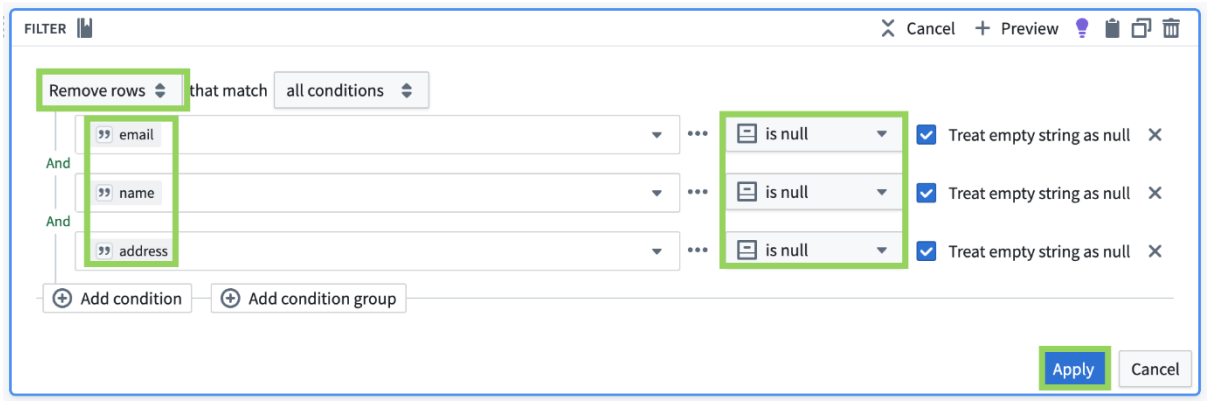
2. Give the transform a relevant name.



3. Search for the Filter rows transform.



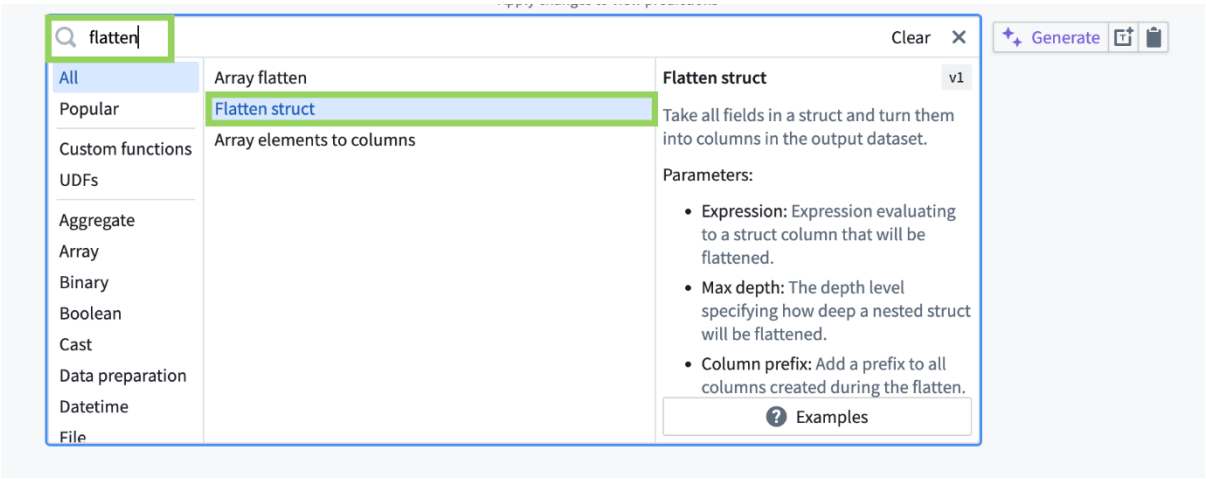
4. Filter the rows where where **email**, **name** and **address** are **null**. Click **Apply.**



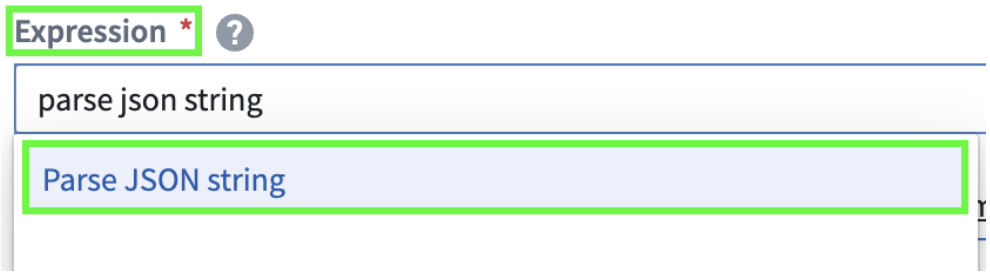
Step 3: Clean the address field

Notice that the **address** field is in JSON format and doesn't convey the information easily for the end user. We can use Pipeline Builder transforms to create a clean string with the street + city + state format. You will create a nested Transform, to flatten the struct defined by the address JSON.

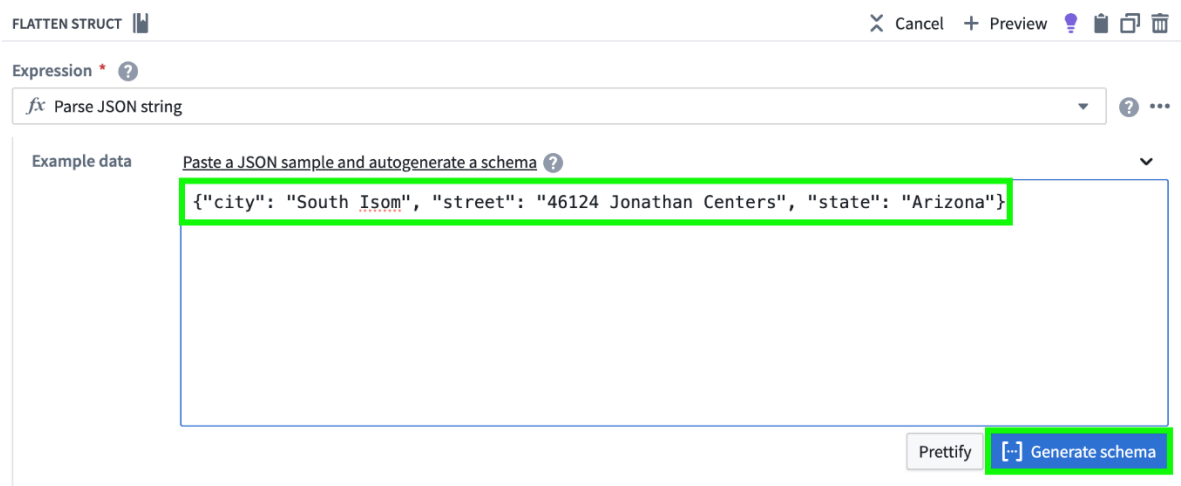
1. Search for the Flatten struct transform.

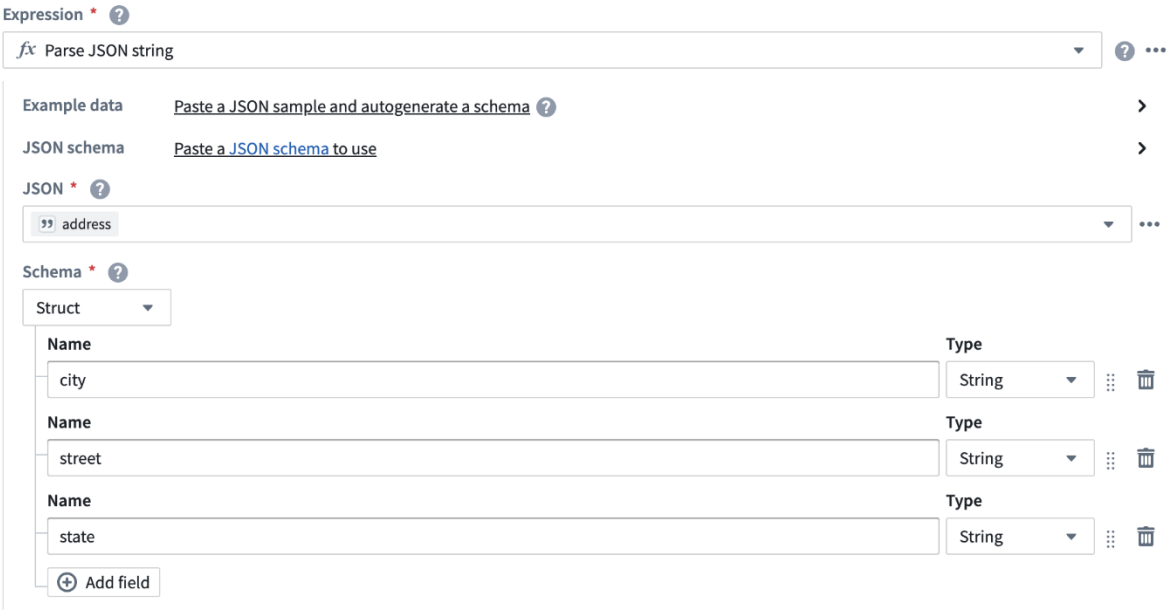


2. The **Flatten struct** transform requires a struct type input (a type of data format), so you need to parse the JSON string to a struct.
3. Select the **Parse JSON as Struct** expression.

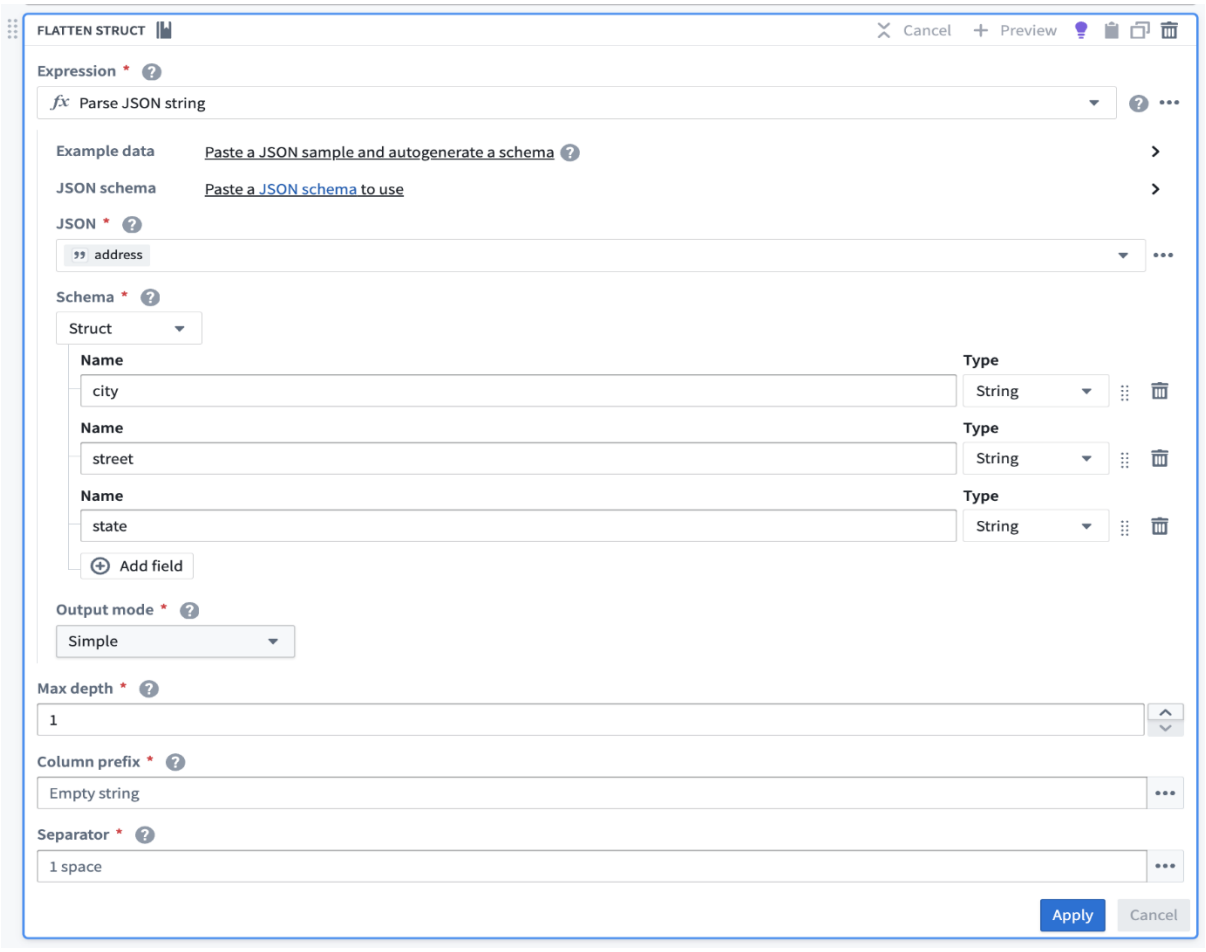


4. Paste in the below example address to infer the schema.
    - {"city": "South Isom", "street": "46124 Jonathan Centers", "state": "Arizona"}

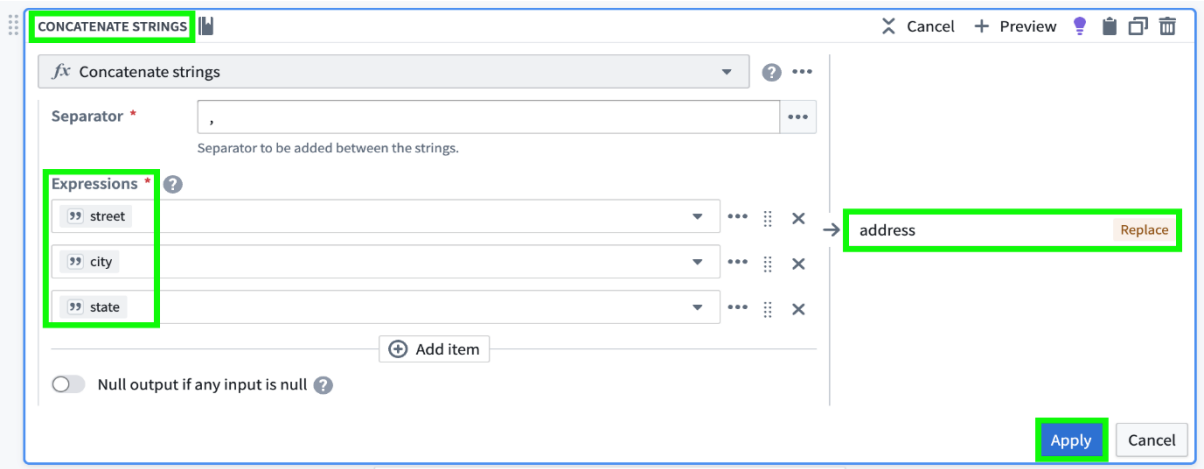




5. Your final Flatten struct transform should look similar to the below. You may need to change the underscore in **Separator** to a space:

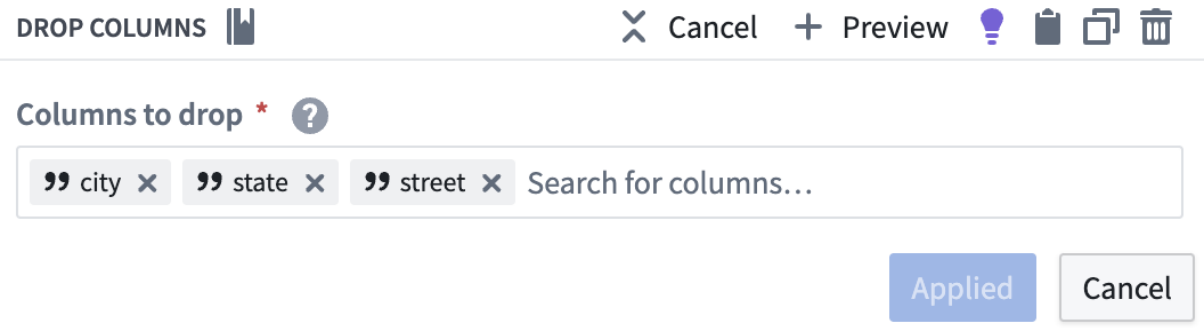


6. This board creates three columns, **city, state, and street.** Concatenating them into a new **address** column will result in a readable string column. For now, click **Apply.**
7. Search for the Concatenate Strings transform and select the **city, state, and street** columns. Add , as the Separator.
8. Name the output column **address.**
9. Click **Apply.**



Step 4: Drop unnecessary columns

1. Search for the **Drop Columns** transform.
2. Select **city, state, and street** columns.
3. Click **Apply.**
4. Make sure you apply all changes and click **Close** to take you back to the screen with the dataset nodes.



Work with Different File Types: Preprocess and Clean the Transactions Dataset

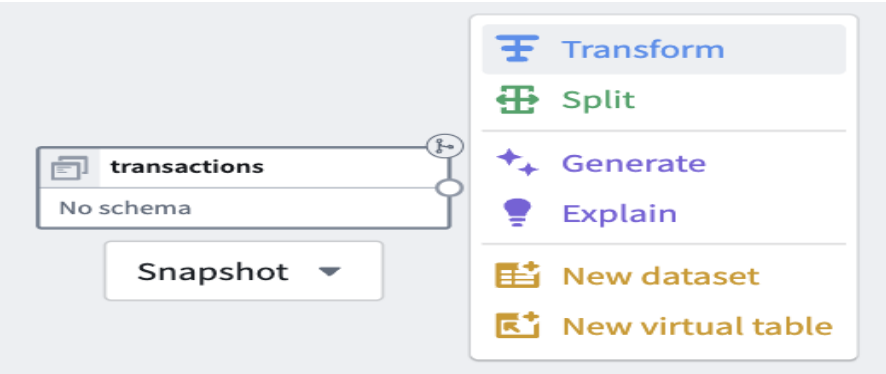
Datasets in Foundry are collections of files, generally Parquet. However, raw data may arrive in different formats, and your pipeline needs to be able to handle these formats and transform them into adequate Foundry datasets.

Throughout this section, you will walk through an example of handling a raw dataset in JSON format using readily available Pipeline Builder transforms. This will provide you with a practical understanding of how to manage various file formats in your pipeline.

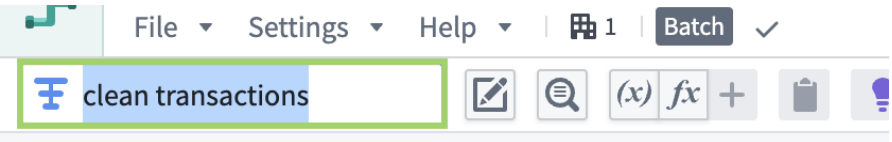
Additionally, you will be introduced to Foundry's AI assistant, which, if available on your Foundry installation, can be a valuable resource for simplifying the processing and transformation. By leveraging the AI assistant's capabilities, you can streamline the handling of various formats, further enhancing the efficiency and performance of your pipeline.

Step 1: Create a new transform for the transactions dataset

1. Similarly to the previous datasets, create a new transform.

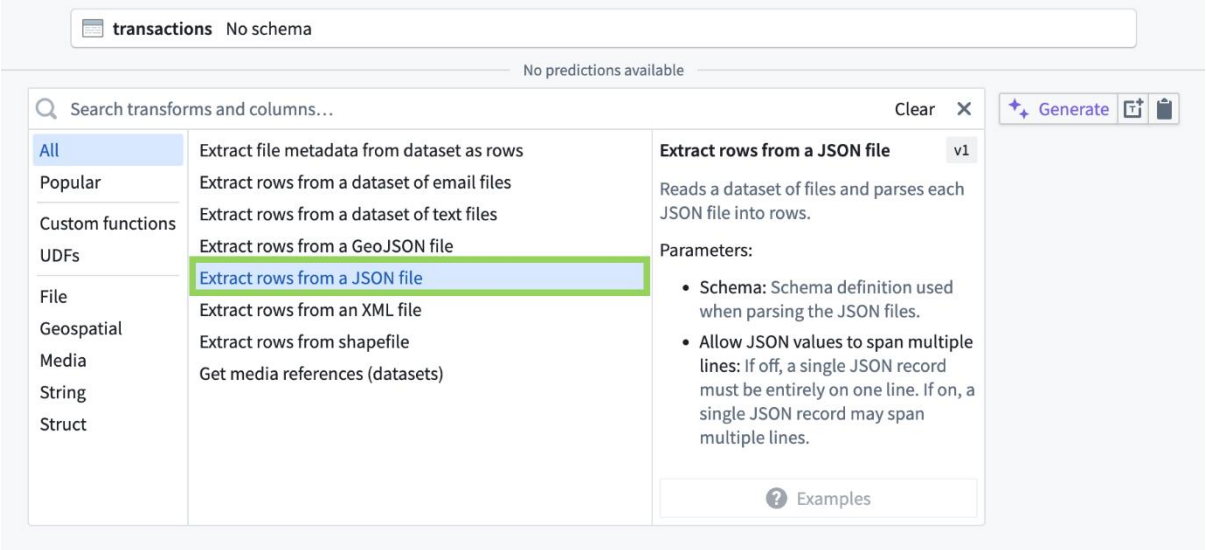


2. Give it a relevant name.

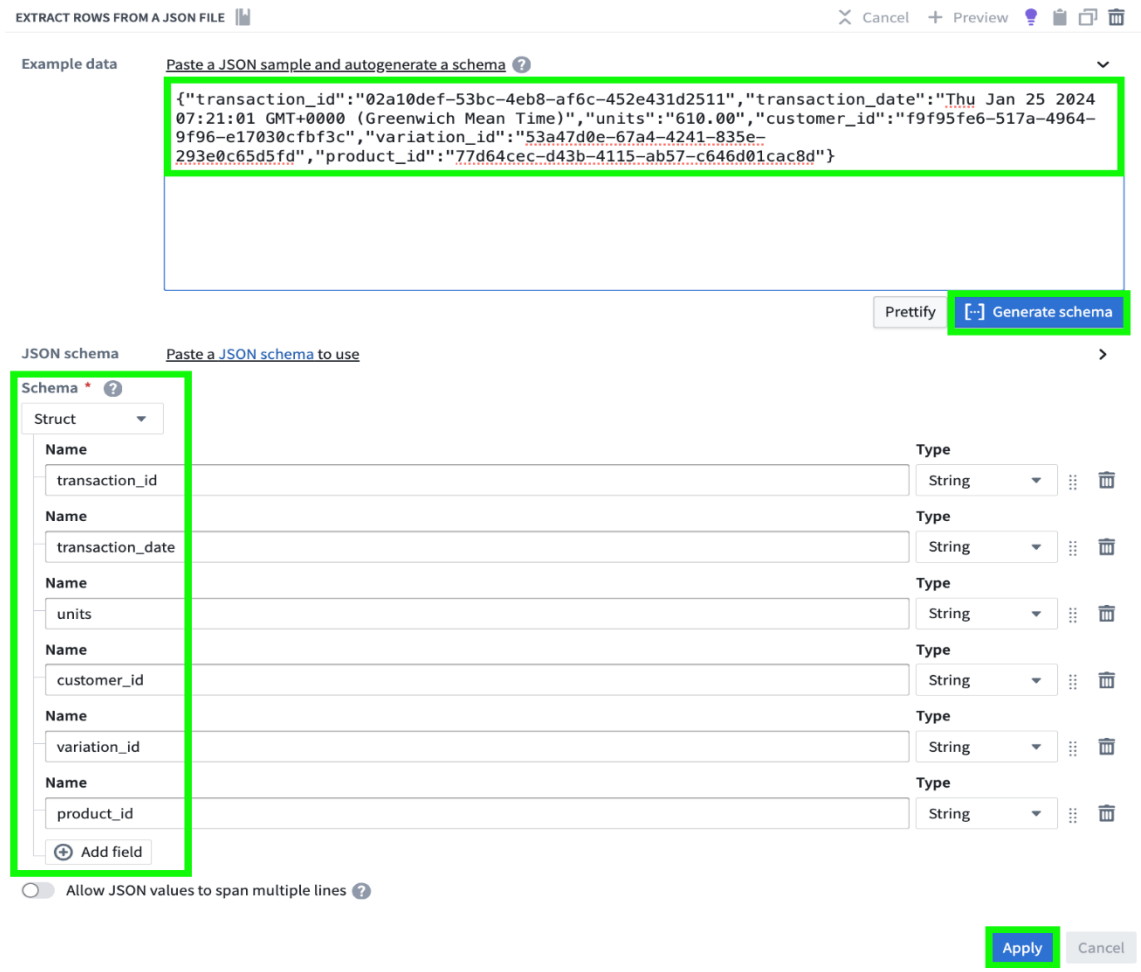


Step 2: Transform from JSON to dataset

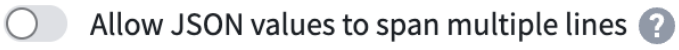
1. Select the **Extract rows from a JSON file** transform



2. Paste in a sample JSON to auto-generate the schema. You can get one by opening the transactions file locally.
    - For example, {"transaction_id":"02a10def-53bc-4eb8-af6c-452e431d2511","transaction_date":"Thu Jan 25 2024 07:21:01 GMT+0000 (Greenwich Mean Time)","units":"610.00","customer_id":"f9f95fe6-517a-4964-9f96-e17030cfbf3c","variation_id":"53a47d0e-67a4-4241-835e-293e0c65d5fd","product_id":"77d64cec-d43b-4115-ab57-c646d01cac8d"}

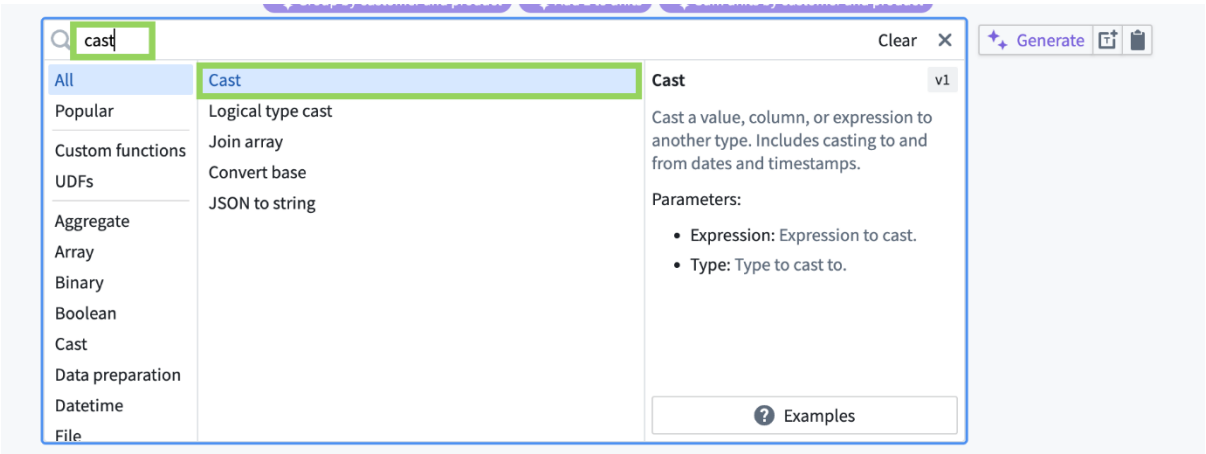


3. Toggle the “Allow JSON values to span multiple lines” option to False (because we have one JSON file per row).



Step 3: Cast transaction_date to Timestamp

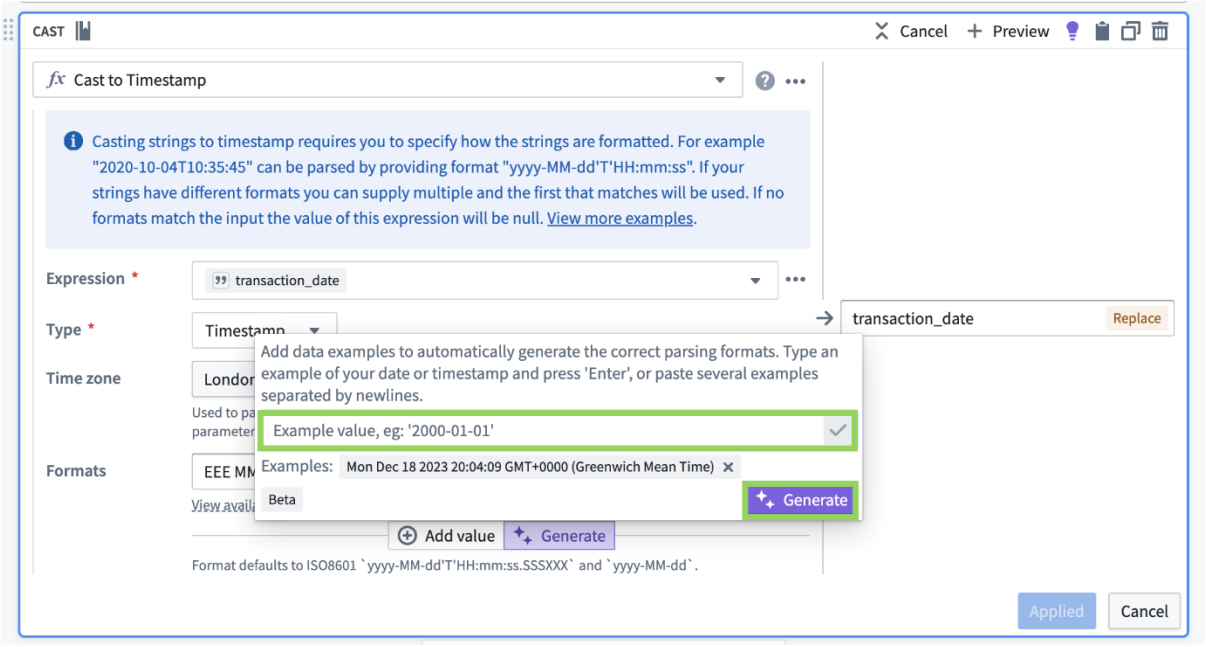
1. Search for the **Cast** transform.



2. Select 'transaction_date' as the **Expression.**

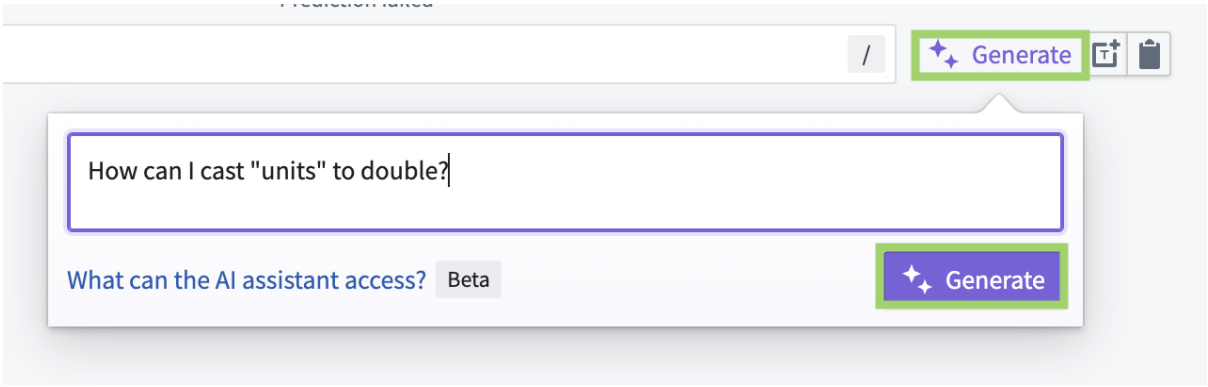
3. Select the type “Timestamp” and use the AI Assistant (Generate button) to infer the correct Timestamp format from an example string from the dataset. If AI Assistant is not available on your Foundry instance, just paste in the format: EEE MMM dd yyyy HH:mm:ss 'GMT'Z (zzzz).

4. Then click **Generate** and **Apply**.

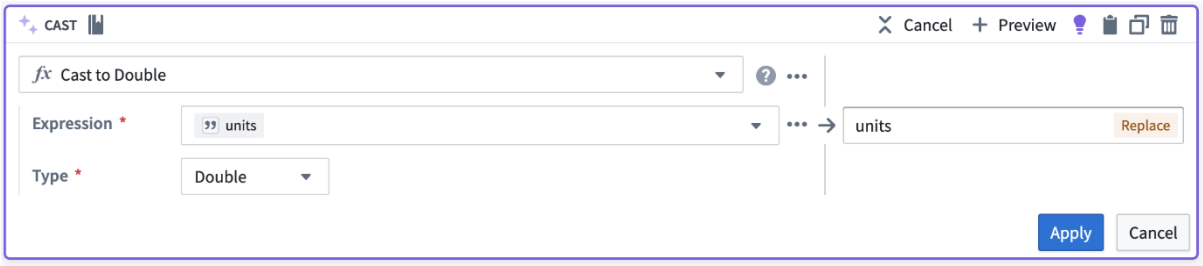


Step 4: Cast units to double

1. If available on your Foundry instance, you can use the AI Assistant again to generate a transform casting the units column to a double data type.If you don’t have the AI Assistant available, simply select the **Cast** transformation for a new board.



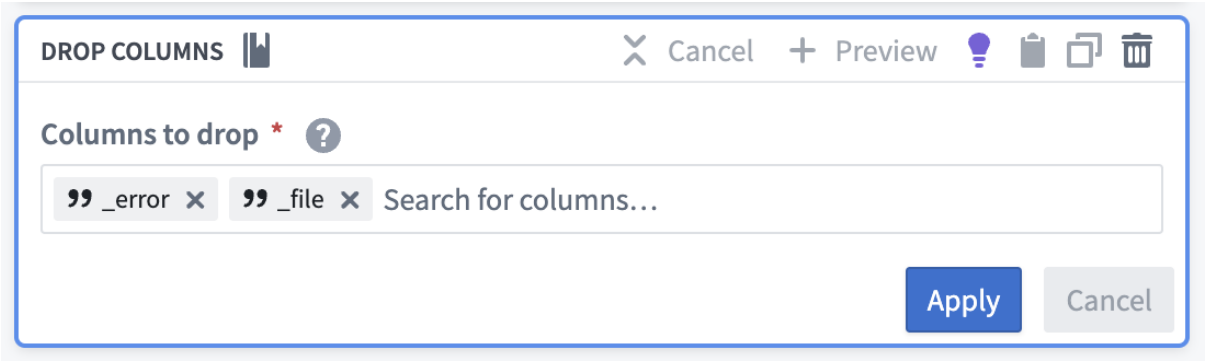
2. The AI assistant should generate the board similar to the one below. There may be missing pieces in it, so you might need to adjust it.



Step 5: Drop unused columns

The conversion from JSON generated two additional columns, **_error** and **file**. These are not needed for this specific pipeline, so it's safe to drop them. When processing many files, these could be valuable to catch any formatting errors in the raw files.

1. Search for the Drop Columns transform.
2. Select **_error** and **file.**
3. Click **Apply**.
4. Make sure you have applied all changes and click **Close** to return to the main screen with the dataset nodes.



Data Integration, Joining, and Aggregations

Introduction

In this module, you will build upon the clean data generated previously, in order to derive operationally valuable datasets, focusing on generating a Customer Lifetime Value (CLV) dataset. The cleaned data in your pipeline is a valuable foundation for key metrics and analyses, and while it is essential, it becomes even more valuable when integrated to break down data silos and provide comprehensive insights.

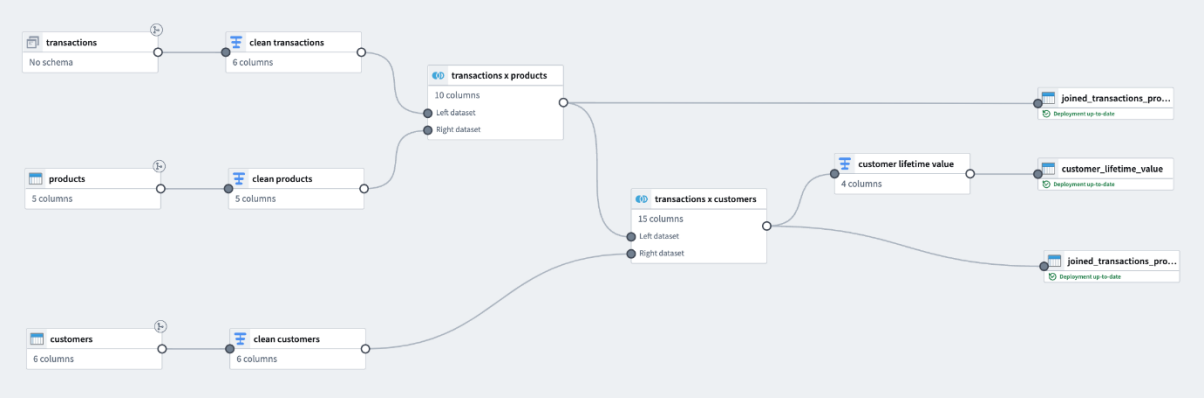
Throughout this section, you will gain hands-on experience with critical data processing techniques, such as joins and aggregations. Additionally, you will learn how to interpret the results of your pipeline to ensure that it delivers meaningful and accurate information.

In this module, you will explore the following concepts:

- **Joining and aggregation**: Learn how to combine and summarize datasets, while being mindful of potential pitfalls such as duplicates, incorrect join conditions, and the computational expense of joins.
- **Materializing outputs**: Understand the significance of outputs and why they matter.
- **Validating the output of the join**: Learn the importance of validating your joined dataset, as joins can sometimes introduce duplicates or errors if not executed correctly.

By the end of this section, you will have gained valuable insights into advanced pipeline building concepts and techniques. These skills will enable you to create more sophisticated, reliable, and efficient pipelines that effectively transform raw data into actionable intelligence.

At the end of this section, you will have created a pipeline similar to the one below:



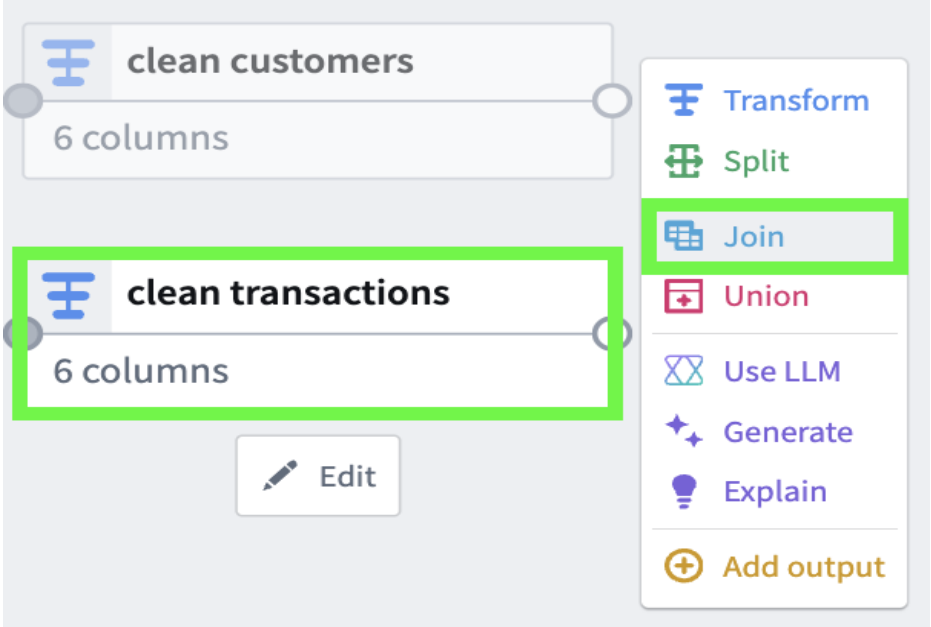
Simple Joins: Join Transactions and Products

To derive operationally valuable insights, it is often necessary to join different datasets, such as combining transactional data with product pricing, to paint a comprehensive picture of customer value and behavior. A Customer Lifetime Value dataset is one such example, showcasing the total revenue brought in by each customer since their registration.

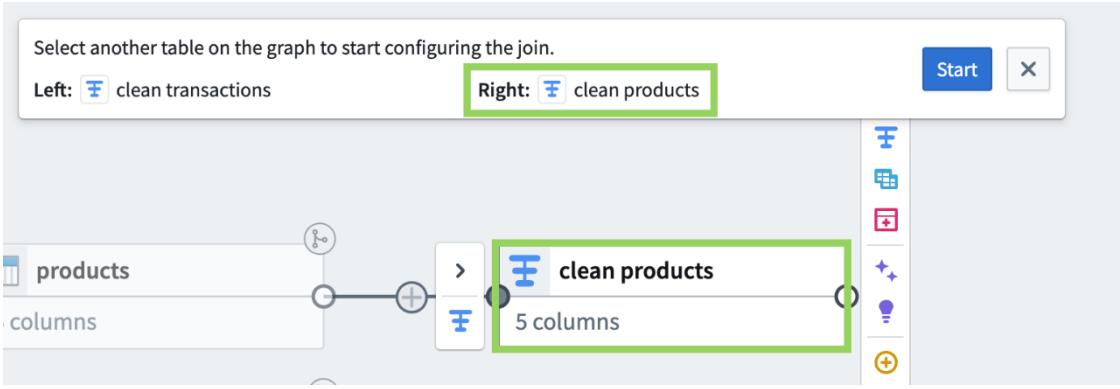
The transactions dataset holds information about all the purchases of all the customers, and the products dataset holds information about the price of each product. Joining those two will give a simple calculation for the revenue brought in by each transaction.

Step 1: Join clean transactions and clean products

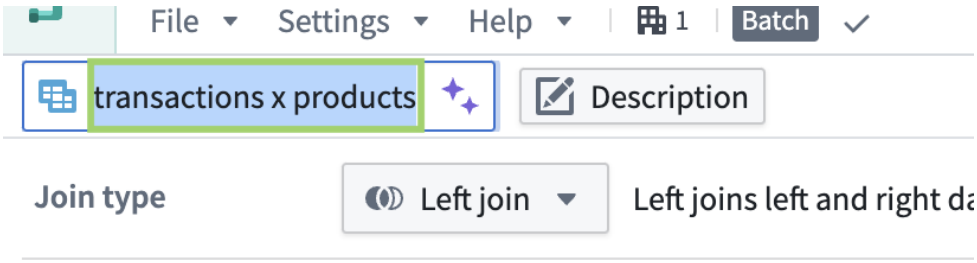
1. Select the **clean transactions** node and choose the **Join** option on the right hand side.



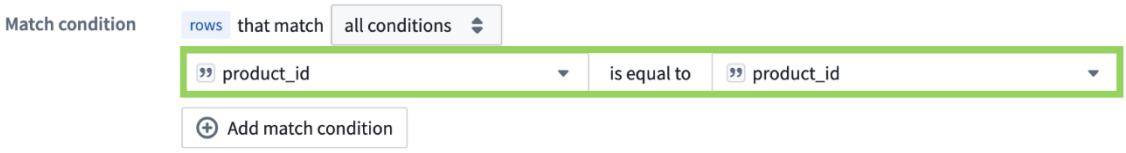
2. Choose the **clean products** node to join with. Click **Start.**



3. Give the join a relevant name.



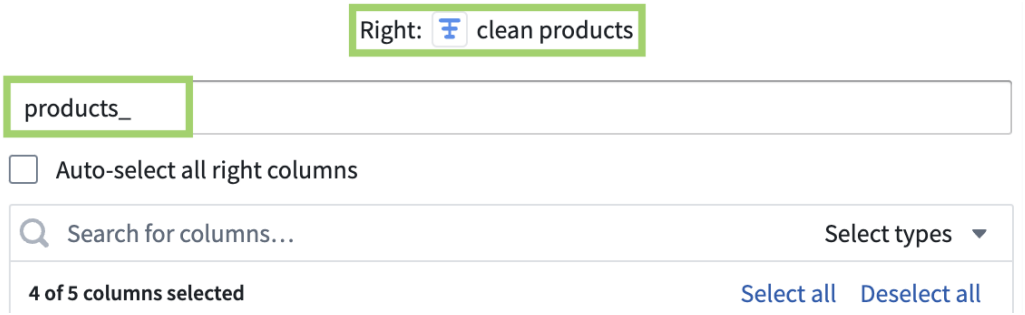
4. Set **product_id is equal to product_id** as the Join Condition.



Step 3: Prefix the product columns with products_

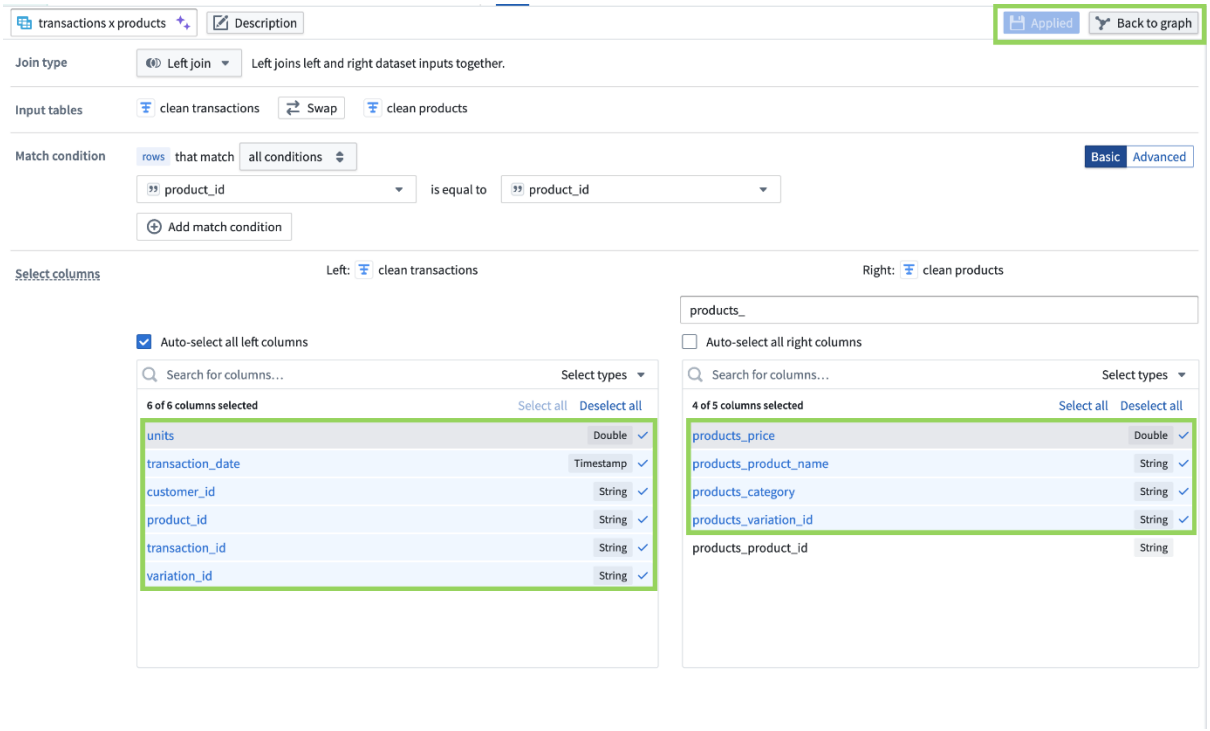
After we join the datasets, the resulting single dataset may have repeated column names. We need to add a prefix to some of the columns to distinguish which dataset they originated from.

1. On the right side of the screen, input products_ into the prefix input bar.



Step 4: Verify Updates

1. Ensure you have the same columns selected in your join as in the screenshot below.
2. Click **Apply**
3. Click **Close** or **Back to graph**.



Chaining Joins: Join Transactions x Products with Customers

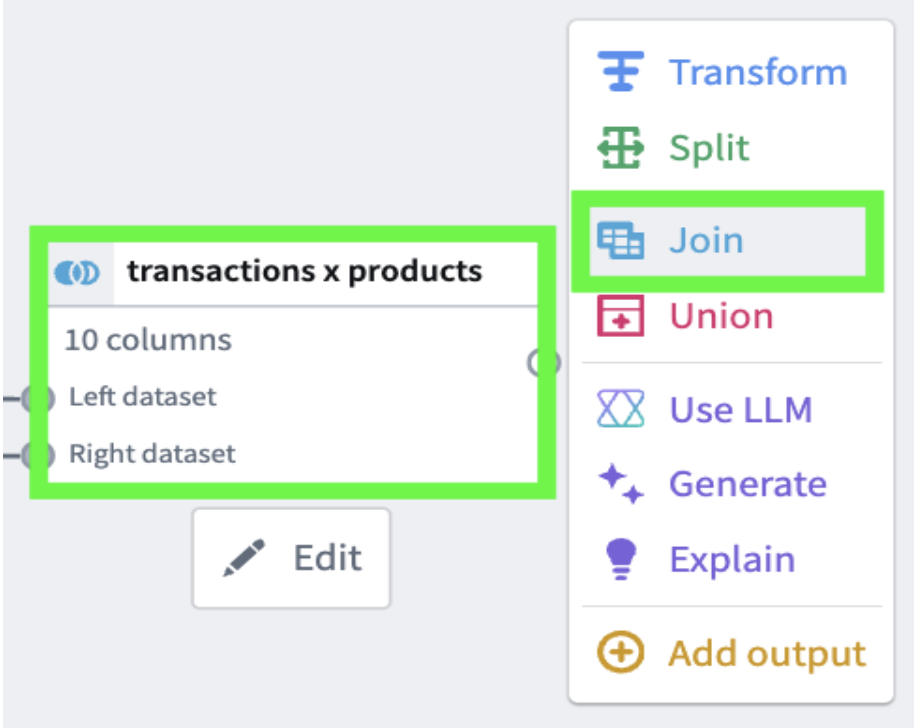
Chaining joins is the process of combining multiple datasets sequentially, with each join operation using the output of the previous join as an input. This technique is commonly used when you need to integrate data from several sources to create a comprehensive and operationally valuable dataset.

In your current example, you have created the **transactions x products** dataset by joining the

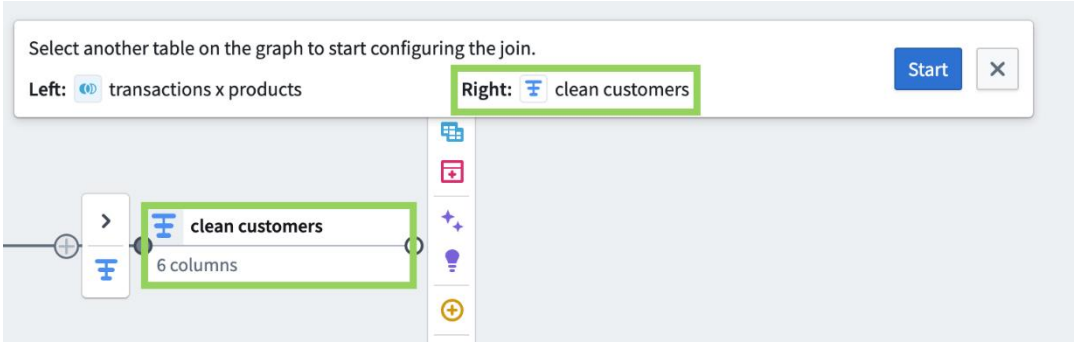
two, and you now need to integrate the **customers** data to create a complete view of the total revenue for each customer.

Step 1: Join transactions x products with clean customers

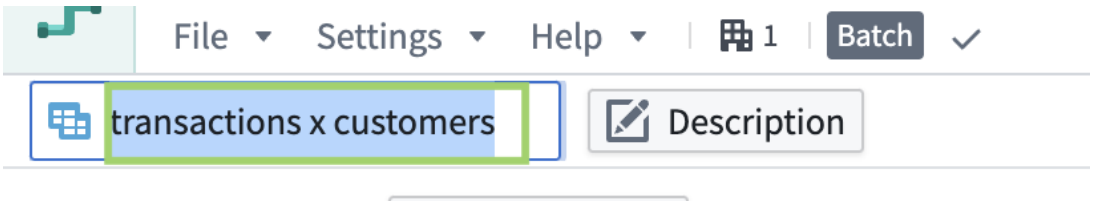
1. Select the transactions x products node and choose the **Join** option on the right hand side.



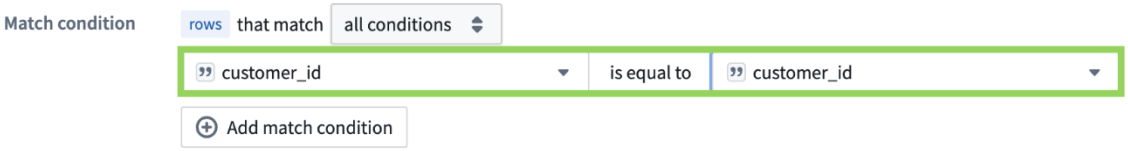
2. Choose the **clean customers** node to join with. Click **Start.**



3. Give the join a relevant name, for example **transactions x products x customers**, or just **transactions x customers.**

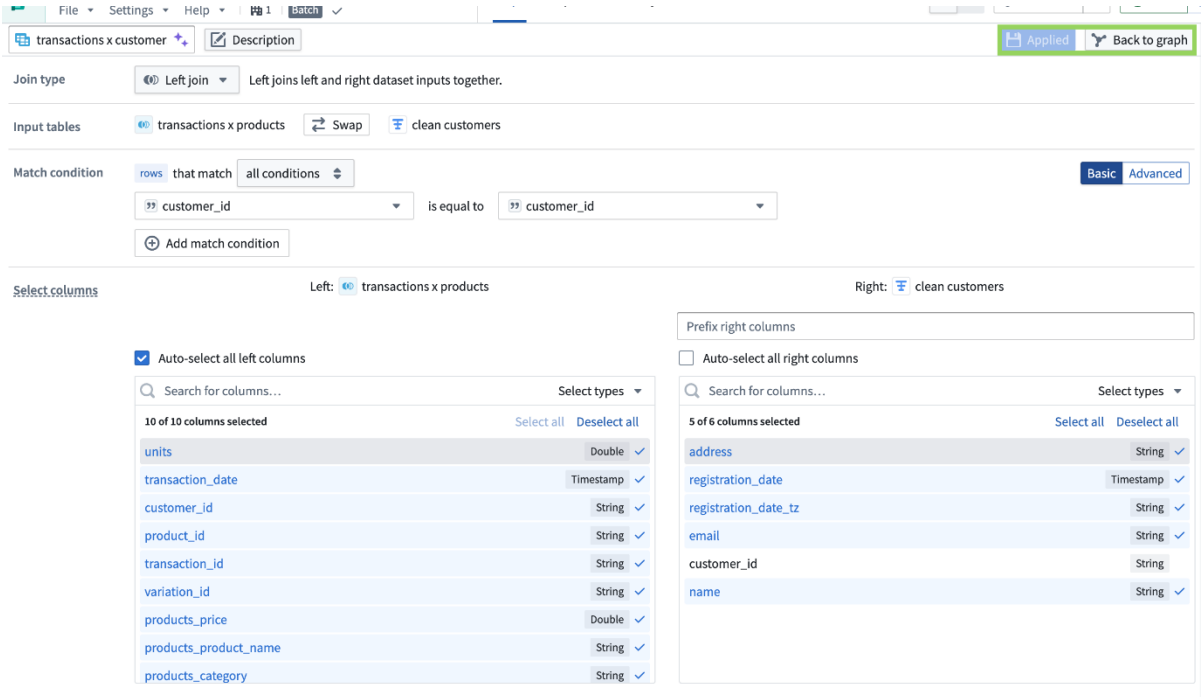


4. Select **customer_id is equal to customer_id** as the match condition.



Step 2: Verify updates and apply

1. Ensure your updates match the screenshot below.
2. Click **Apply** and **Back to Graph** or **Close**.



Add a Dataset Output

The nodes in your pipeline so far are each intermediate transformations. Materializing outputs refers to the process of storing the results of a data transformation or join operation as a new dataset, which can be reused for other use cases and analyses within Foundry.

Materializing outputs is particularly beneficial when working with large or complex data transformations, as it allows you to break down your pipelines to encapsulate the results.

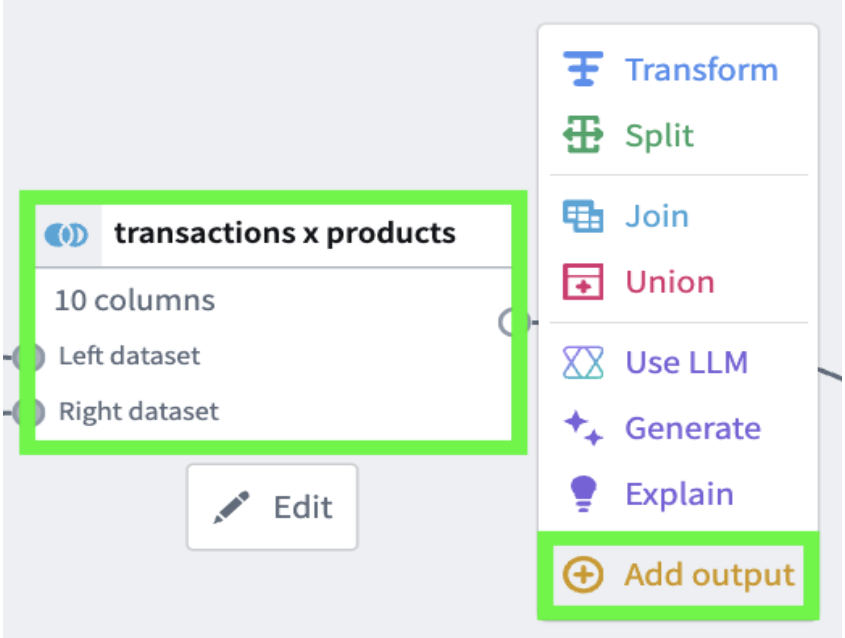
Materializing outputs can improve the maintainability and traceability of your data pipeline, so you can track the lineage of each dataset and understand how it was derived.

Materialized outputs can not be further used as inputs to other transformations, and they represent the outcome of your pipeline.

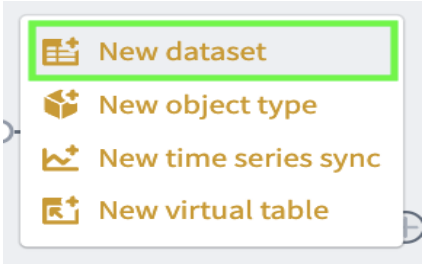
In your current example, you will materialize the outputs of both the **transactions x products** join and the **transactions x products x customers** join, since they will be operationally valuable for various use cases further down the line.

Step 1: Add a new dataset output to transactions x products

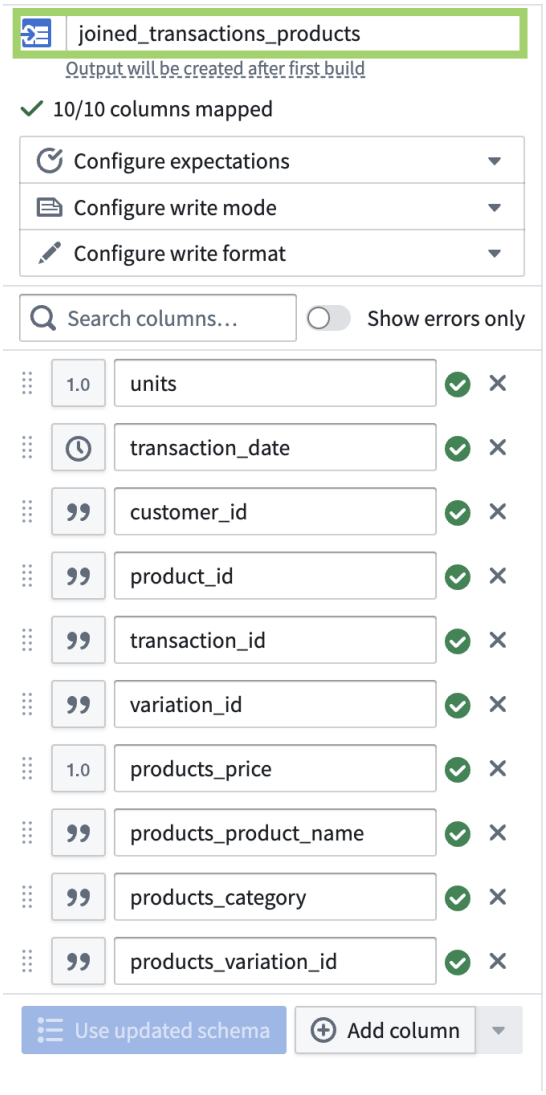
1. Navigate to the **transactions x products** node and select the **Add output** option in the list.



2. Select **New dataset.**

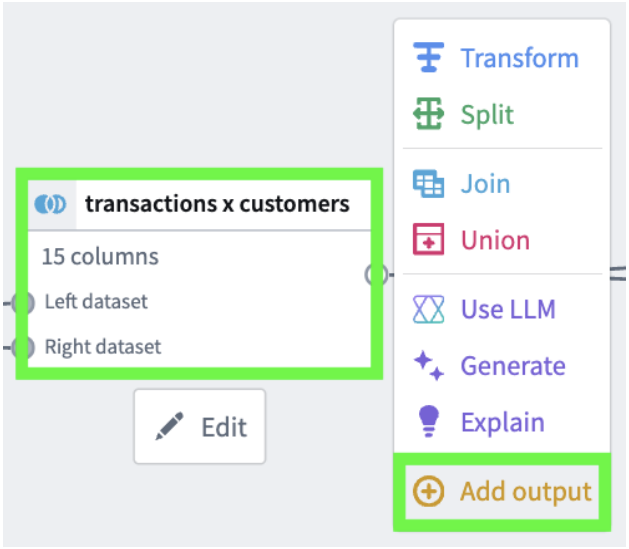


3. Give the generated dataset a relevant name on the right hand side.

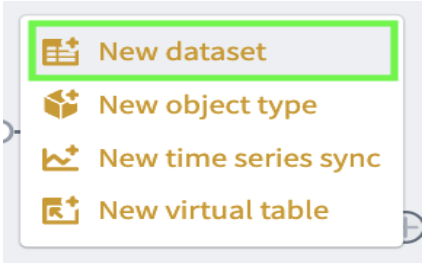


Step 2: Add a new dataset output to transactions x customers

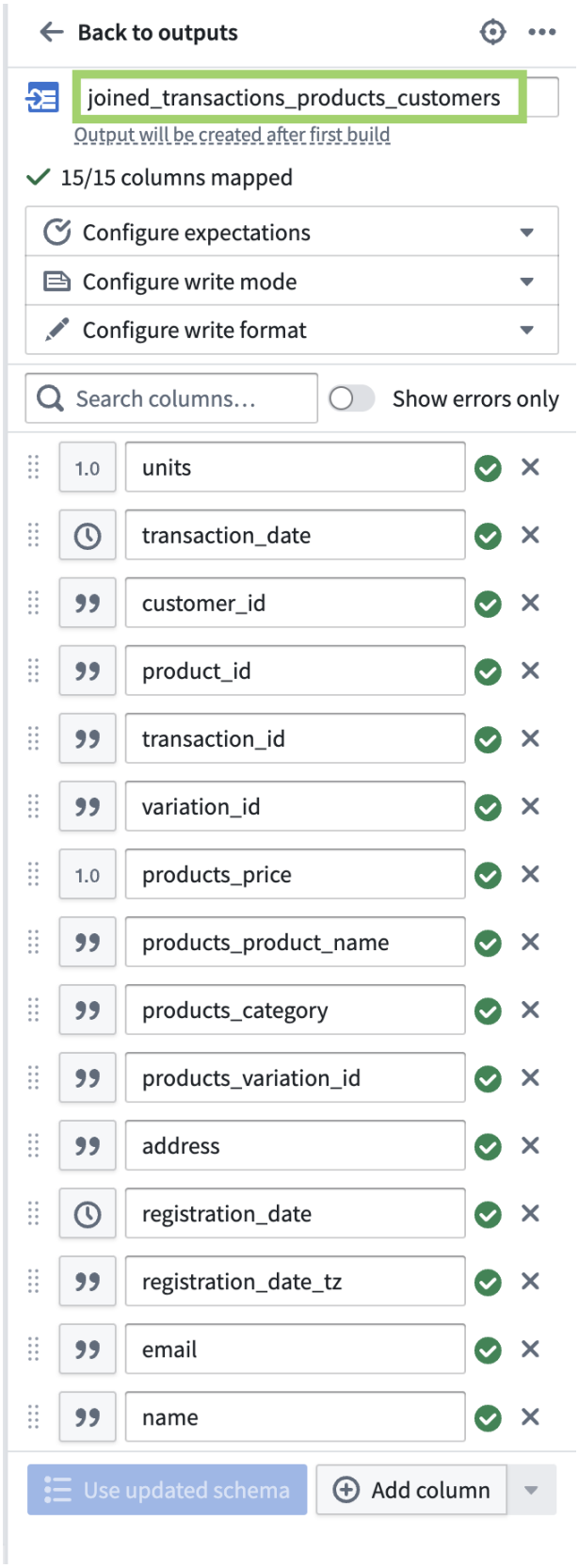
1. Navigate to the **transactions x customers** node and select the **Add output** option in the list.



2. Choose **New dataset**.



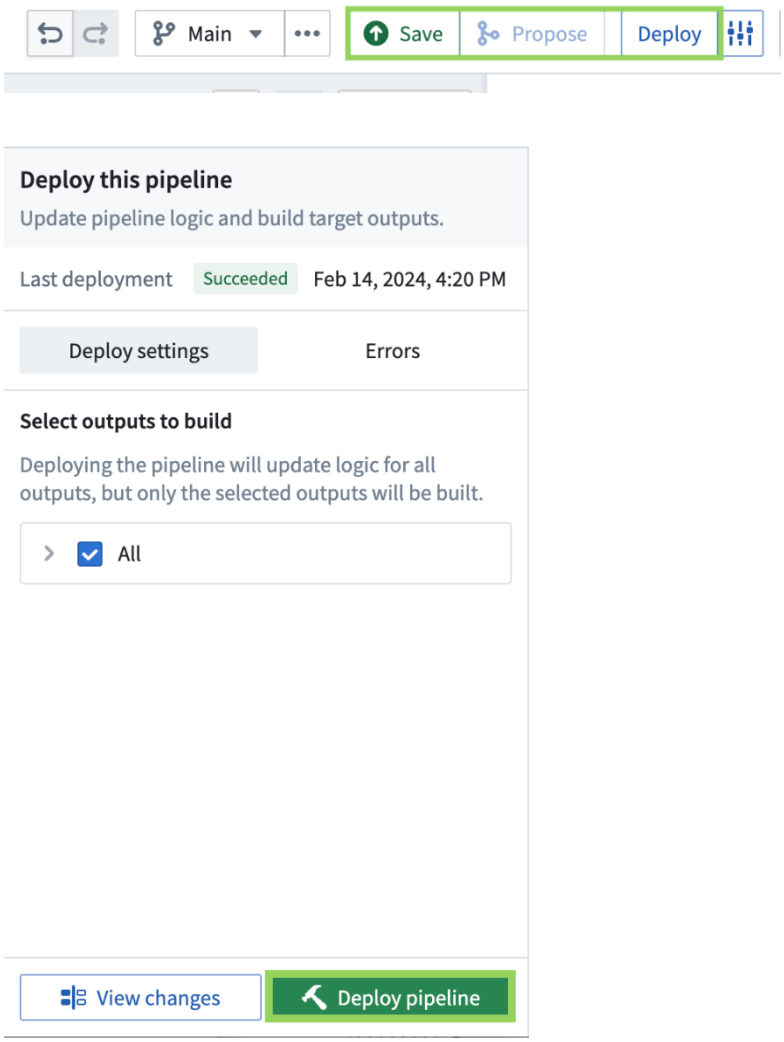
3. Give the dataset a relevant name.



Step 3: Save all changes and deploy the pipeline

1. Click **Save**
2. Click **Deploy** and note the pop-up stating that a build is starting.

Once the pipeline is deployed, your outputs will be materialized into Foundry resources and will be created in the folder of the pipeline.



Identify a Bad Join: Drill into Transactions x Products

In some cases, a bad join occurs when two datasets are combined incorrectly, leading to inaccurate, duplicate, or missing data in the resulting dataset. This can significantly impact the quality of the insights derived from the data and, ultimately, the decisions made based on those insights.

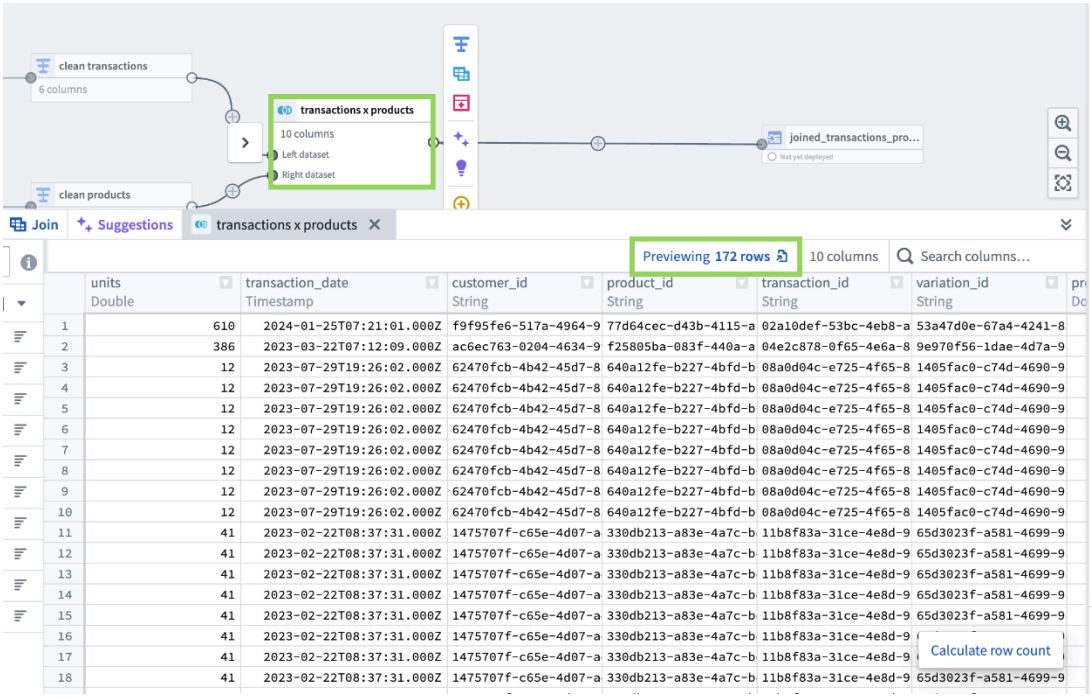
When working with data, it's crucial to understand the relationships between different datasets and the keys used to join them. In many cases, a bad join can be the result of using the wrong key or not considering all relevant keys in the join condition. This can lead to a

dataset that is either too large, due to duplicate entries, or too small, due to missing data.

It is a recommended practice to analyze the outputs of your joins and the techniques introduced in the sections before, such as the Preview functionality, are a great starting point for small datasets. In this section, however, you will leverage a different Foundry tool, Contour, to help analyze the joins that you created in the previous steps. In most cases, you'd be able to do your basic analyses in Preview, however Contour allows you to look through the entire data and discover bugs that would otherwise be impossible to find in Preview.

Step 1: Analyze the transactions x products result using Preview

1. Click on the **transactions x products** node and open the Preview tab at the bottom of the page. Notice that there are 172 rows in the output, whereas **cleaned transactions** only has 50 rows. The expectation would be that each transaction is enriched after the join, but not multiplied. This means that the join likely introduces multiplications.

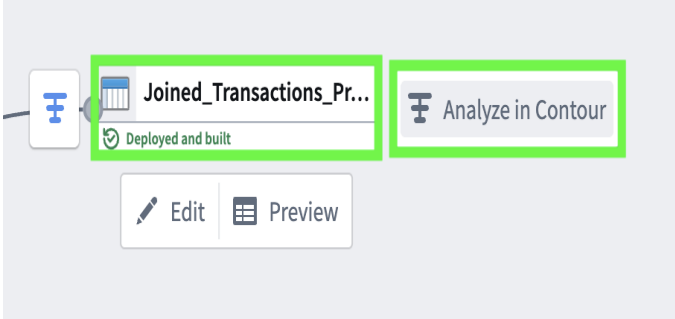


	units Double	transaction_date Timestamp	customer_id String	product_id String	transaction_id String	variation_id String
1	610	2024-01-25T07:21:01.000Z	f9f95fe6-517a-4964-9	77d64cec-d43b-4115-a	02a10def-53bc-4eb8-a	53a47d0e-67a4-4241-8
2	386	2023-03-22T07:12:09.000Z	ac6ec763-0204-4634-9	f25805ba-083f-440a-a	04e2c878-0f65-4e6a-8	9e970f56-1dae-4d7a-9
3	12	2023-07-29T19:26:02.000Z	62470fcb-4b42-45d7-8	640a12fe-b227-4bfd-b	08a0d04c-e725-4f65-8	1405fac0-c74d-4690-9
4	12	2023-07-29T19:26:02.000Z	62470fcb-4b42-45d7-8	640a12fe-b227-4bfd-b	08a0d04c-e725-4f65-8	1405fac0-c74d-4690-9
5	12	2023-07-29T19:26:02.000Z	62470fcb-4b42-45d7-8	640a12fe-b227-4bfd-b	08a0d04c-e725-4f65-8	1405fac0-c74d-4690-9
6	12	2023-07-29T19:26:02.000Z	62470fcb-4b42-45d7-8	640a12fe-b227-4bfd-b	08a0d04c-e725-4f65-8	1405fac0-c74d-4690-9
7	12	2023-07-29T19:26:02.000Z	62470fcb-4b42-45d7-8	640a12fe-b227-4bfd-b	08a0d04c-e725-4f65-8	1405fac0-c74d-4690-9
8	12	2023-07-29T19:26:02.000Z	62470fcb-4b42-45d7-8	640a12fe-b227-4bfd-b	08a0d04c-e725-4f65-8	1405fac0-c74d-4690-9
9	12	2023-07-29T19:26:02.000Z	62470fcb-4b42-45d7-8	640a12fe-b227-4bfd-b	08a0d04c-e725-4f65-8	1405fac0-c74d-4690-9
10	12	2023-07-29T19:26:02.000Z	62470fcb-4b42-45d7-8	640a12fe-b227-4bfd-b	08a0d04c-e725-4f65-8	1405fac0-c74d-4690-9
11	41	2023-02-22T08:37:31.000Z	1475707f-c65e-4d07-a	330db213-a83e-4a7c-b	11b8f83a-31ce-4e8d-9	65d3023f-a581-4699-9
12	41	2023-02-22T08:37:31.000Z	1475707f-c65e-4d07-a	330db213-a83e-4a7c-b	11b8f83a-31ce-4e8d-9	65d3023f-a581-4699-9
13	41	2023-02-22T08:37:31.000Z	1475707f-c65e-4d07-a	330db213-a83e-4a7c-b	11b8f83a-31ce-4e8d-9	65d3023f-a581-4699-9
14	41	2023-02-22T08:37:31.000Z	1475707f-c65e-4d07-a	330db213-a83e-4a7c-b	11b8f83a-31ce-4e8d-9	65d3023f-a581-4699-9
15	41	2023-02-22T08:37:31.000Z	1475707f-c65e-4d07-a	330db213-a83e-4a7c-b	11b8f83a-31ce-4e8d-9	65d3023f-a581-4699-9
16	41	2023-02-22T08:37:31.000Z	1475707f-c65e-4d07-a	330db213-a83e-4a7c-b	11b8f83a-31ce-4e8d-9	
17	41	2023-02-22T08:37:31.000Z	1475707f-c65e-4d07-a	330db213-a83e-4a7c-b	11b8f83a-31ce-4e8d-9	
18	41	2023-02-22T08:37:31.000Z	1475707f-c65e-4d07-a	330db213-a83e-4a7c-b	11b8f83a-31ce-4e8d-9	65d3023f-a581-4699-9

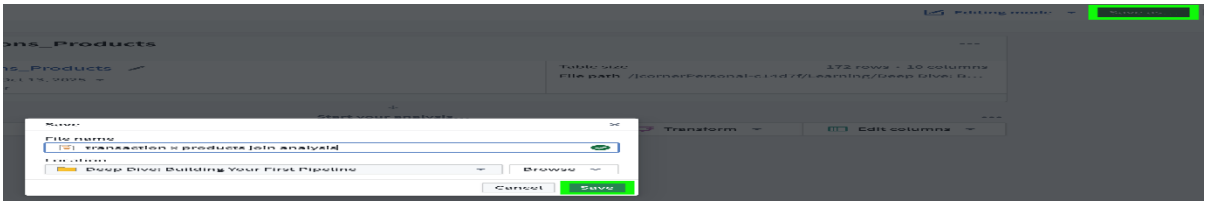
2. You can double check the number of rows in cleaned transactions using Preview as well.

Step 2: Open the materialized dataset using Contour

1. From the transactions x products dataset, click **Analyze in Contour.**

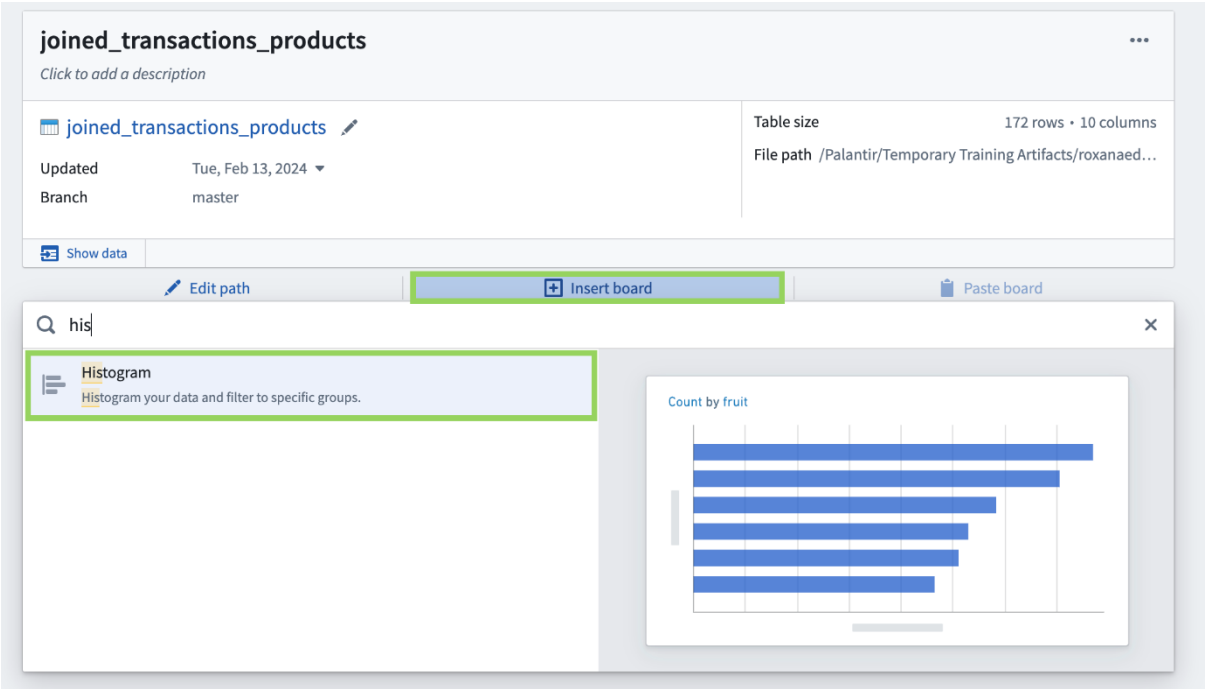


2. A new tab will open. Navigate to it and click **Save as** to save your Contour analysis in your working folder.

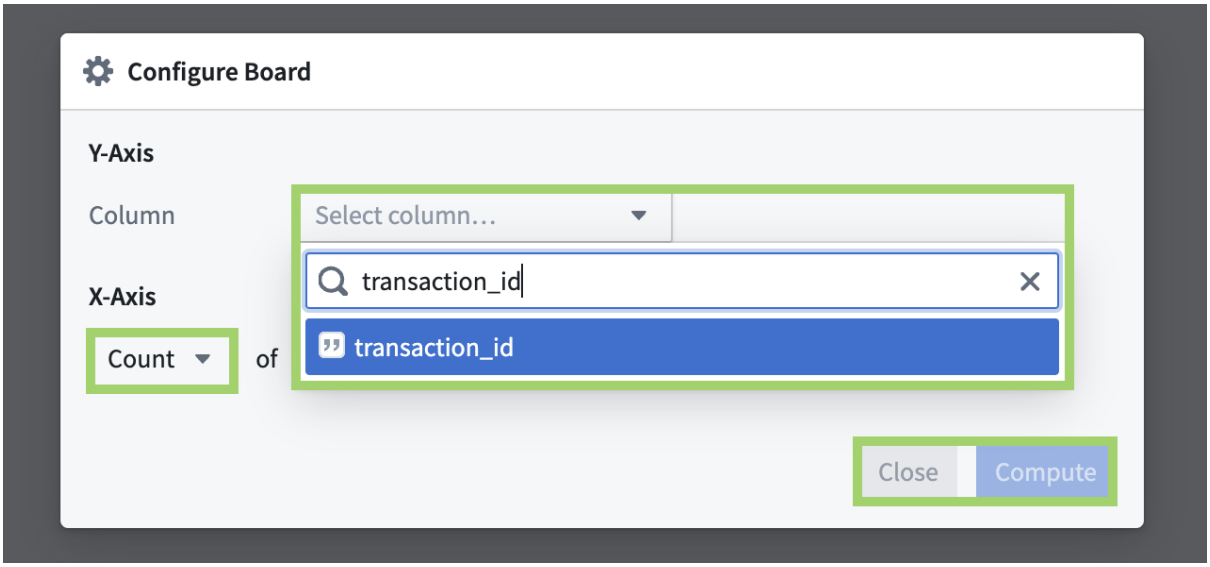


Step 3: Check for transaction_id primary key uniqueness

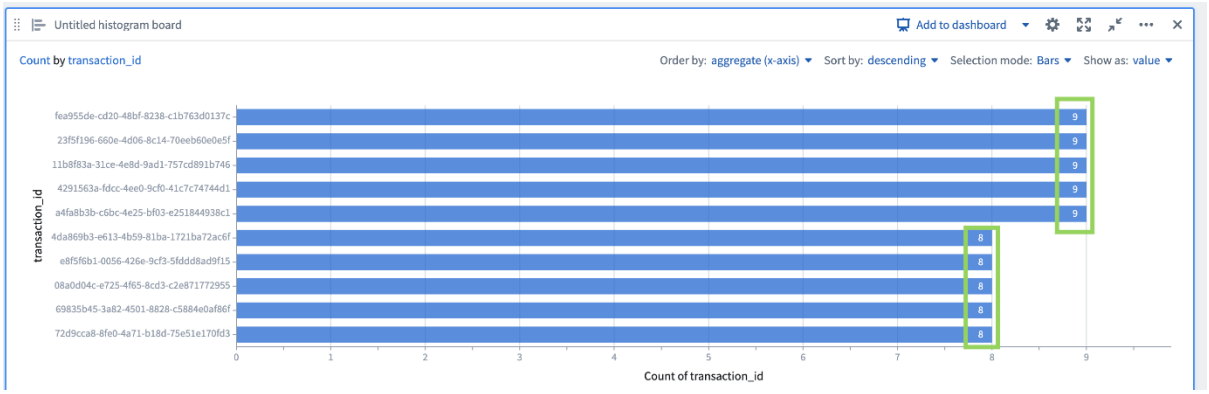
1. Click **Insert Board**.
2. Search and select **Histogram**.



3. Plot the count of the **transaction_id.** This board will then display how many times each primary key occurs in the dataset. Any count more than 1 denotes duplication.

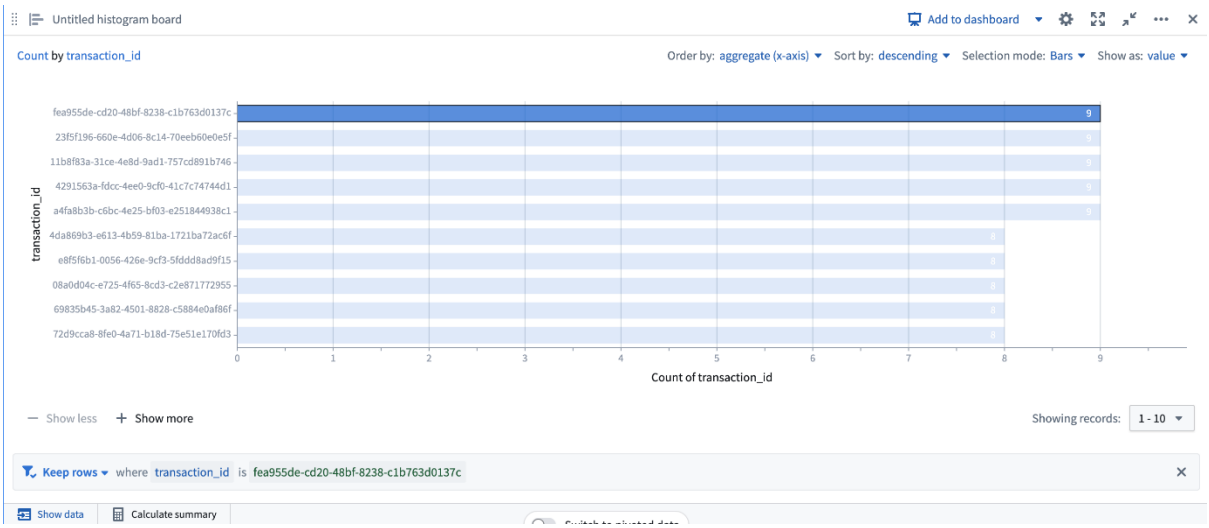


4. Notice that some **transaction_id**s appear up to 9 times, meaning that there was duplication introduced by the join.



Step 4: Drill into one of the duplicated ids.

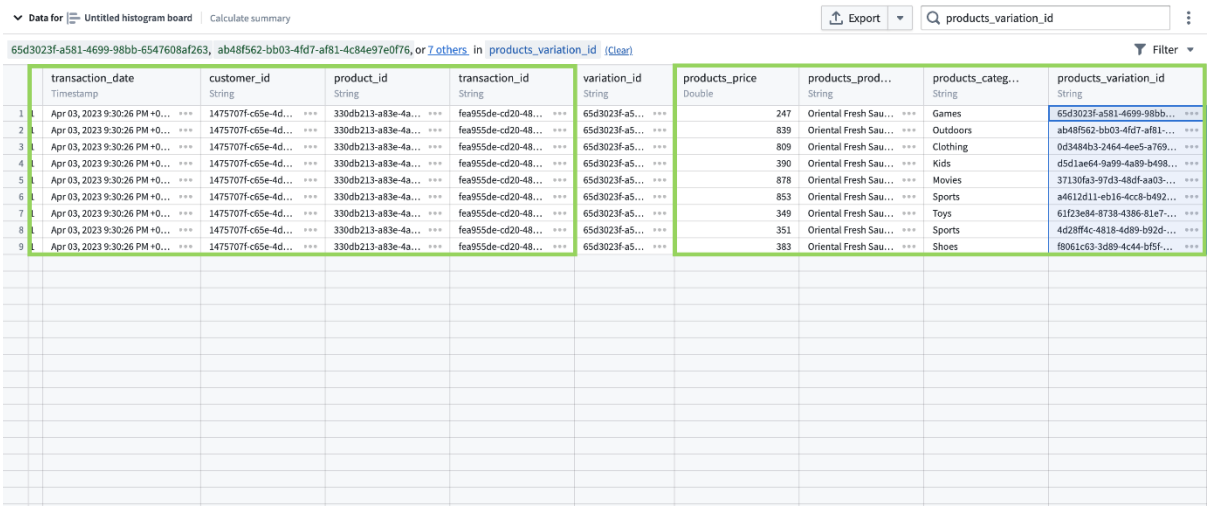
1. Click on the top result. This will automatically filter to the selected values.



Step 5: Open the data and notice the duplication on product_id but the uniqueness of products_variation_id

1. Open the filtered data, by clicking the **Show data** button on the bottom left.

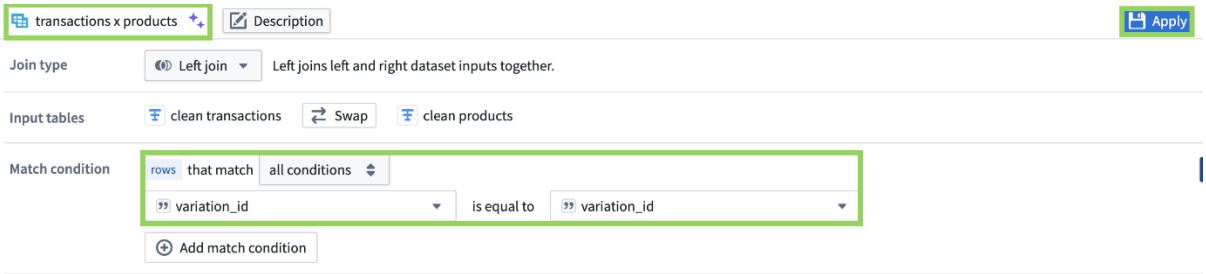
2. Notice that all columns have the same value, except for the ones prefixed with **products_.**



3. This means that the product_ids are not unique, and instead each product has a unique variation_id which is what introduced the duplication in the join.

Step 6: Return to the pipeline and fix the join

1. Return to your Pipeline Builder resource.
2. Change the match condition in the transactions x products join to be based off **variation_id.**
3. Apply the changes and return to the main screen with dataset nodes.



Create an Operationally Valuable Dataset

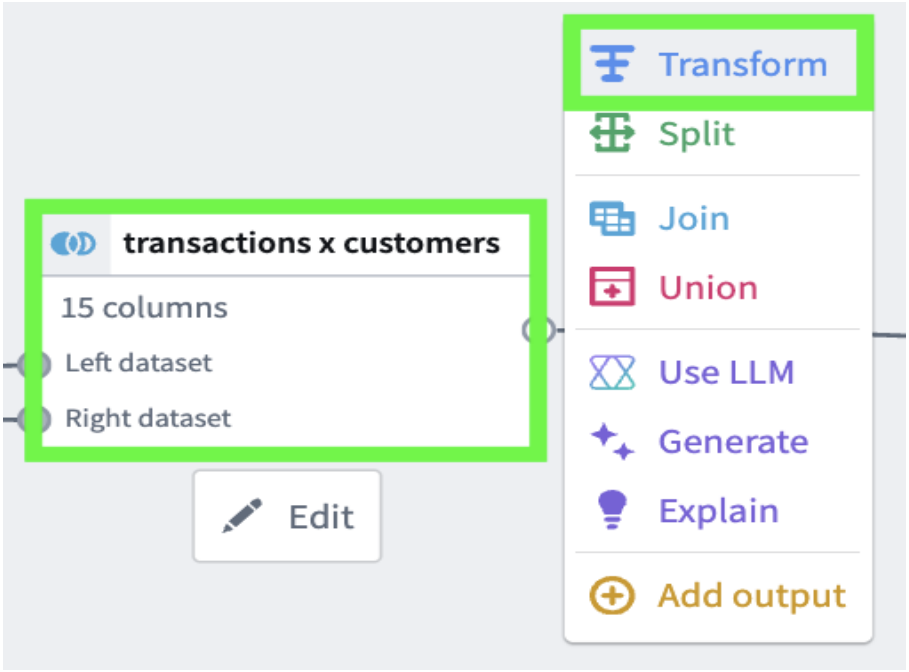
In this section, you will finally use the data to create an operationally valuable dataset, representing the customer lifetime value. Customer lifetime value is a metric that represents the total net profit a company can expect to earn from a customer over the entire duration of their relationship.

Using the combined dataset from the **transactions (x products) x customers** join, you will leverage Pipeline Builder transforms to aggregate the total revenue generated by each

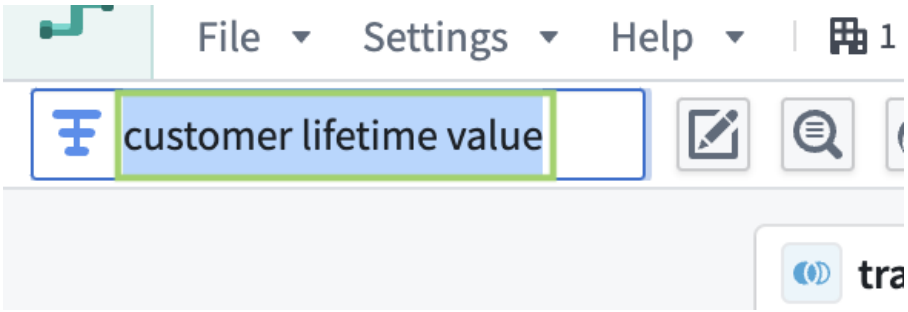
customer and derive their lifetime value. This is, of course, just one example of how you can transform the joined data into valuable insights that can drive business decisions.

Step 1: Transform the transactions x customers dataset

1. Select the **transactions x customers** node and choose the **Transform** option on the right hand side.

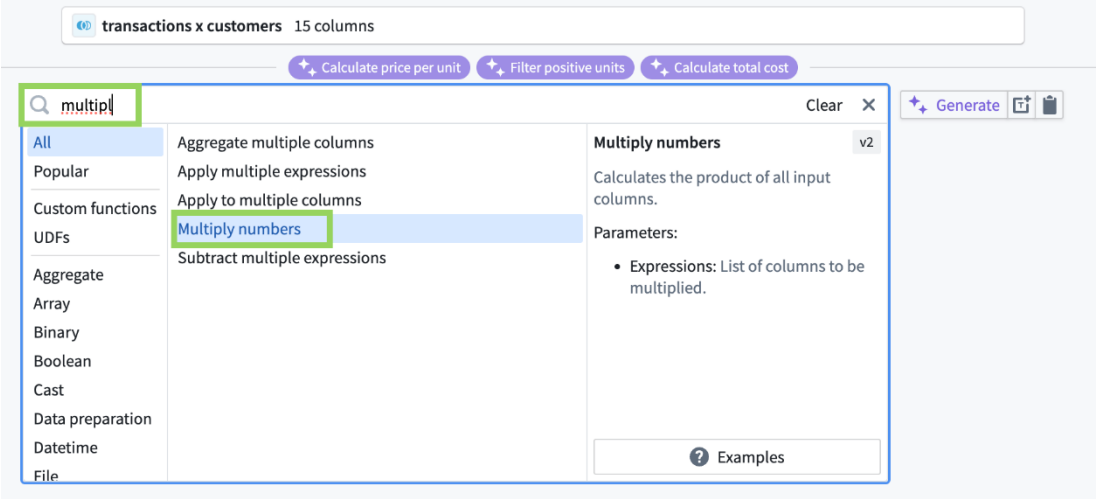


2. Give the transform a relevant name like customer lifetime value.

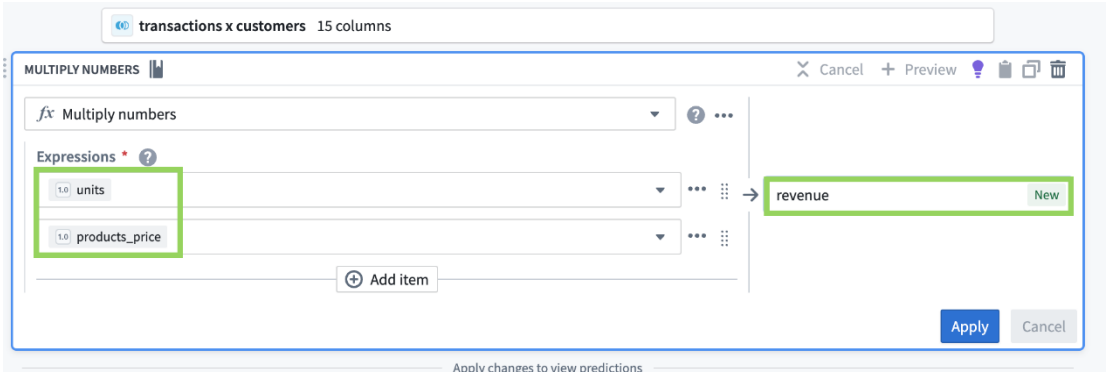


Step 2: Compute the revenue per transaction

1. Select the **Multiply numbers** transform. The revenue generated by each transaction is the number of units sold multiplied by the price of a unit.



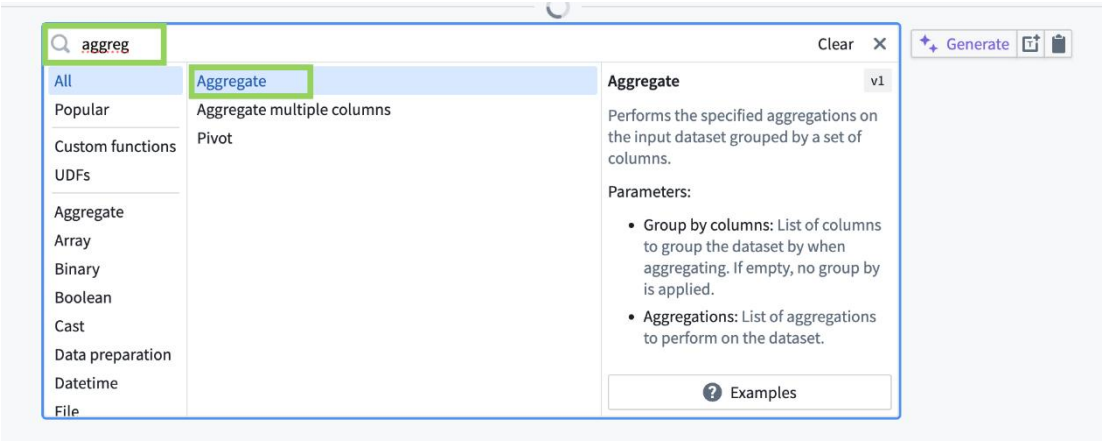
2. Select the **units** and **products_price** columns, and name the output column **revenue**.



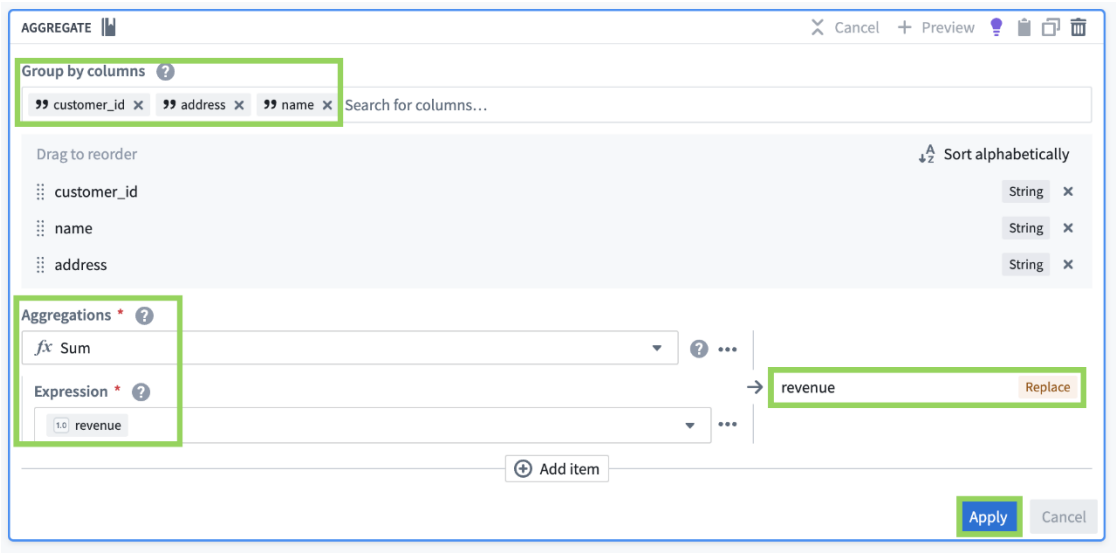
3. Click **Apply.**

Step 3: Aggregate per customer

1. Search for the **Aggregate** transform.



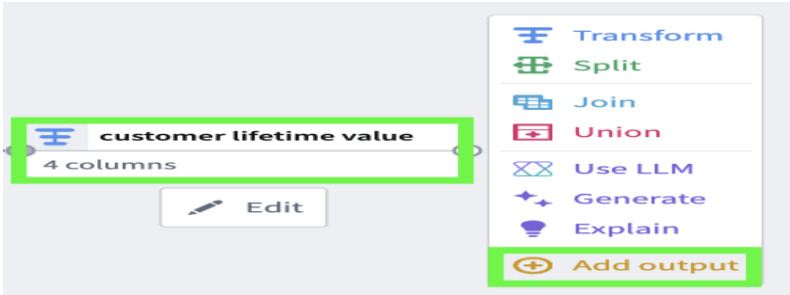
2. Sum the **revenue** column per **customer,** and keep some important columns containing customer details.



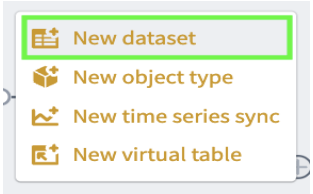
3. Apply all changes and return to the graph.

Step 4: Add a new dataset output to customer lifetime value

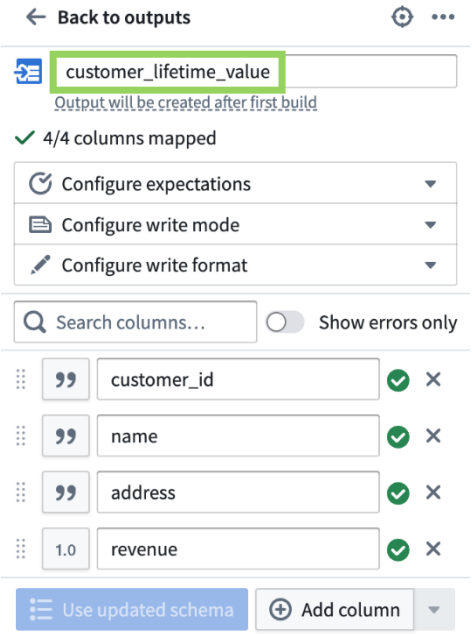
1. Navigate to the **customer lifetime value** node and select the **Add output** option from the list.



2. Choose **New dataset.**



3. Give the output dataset a relevant name.



4. Save and deploy the pipeline.

Additional Considerations and Best Practices

Introduction

The previous sections have laid the groundwork with the fundamental techniques on how to construct a data pipeline in Foundry. However, it is important to recognize that the reality of data processing and engineering often presents scenarios more intricate than the example in this training path. In this section, you will explore some advanced considerations and best practices that will help you handle real-life data complexities.

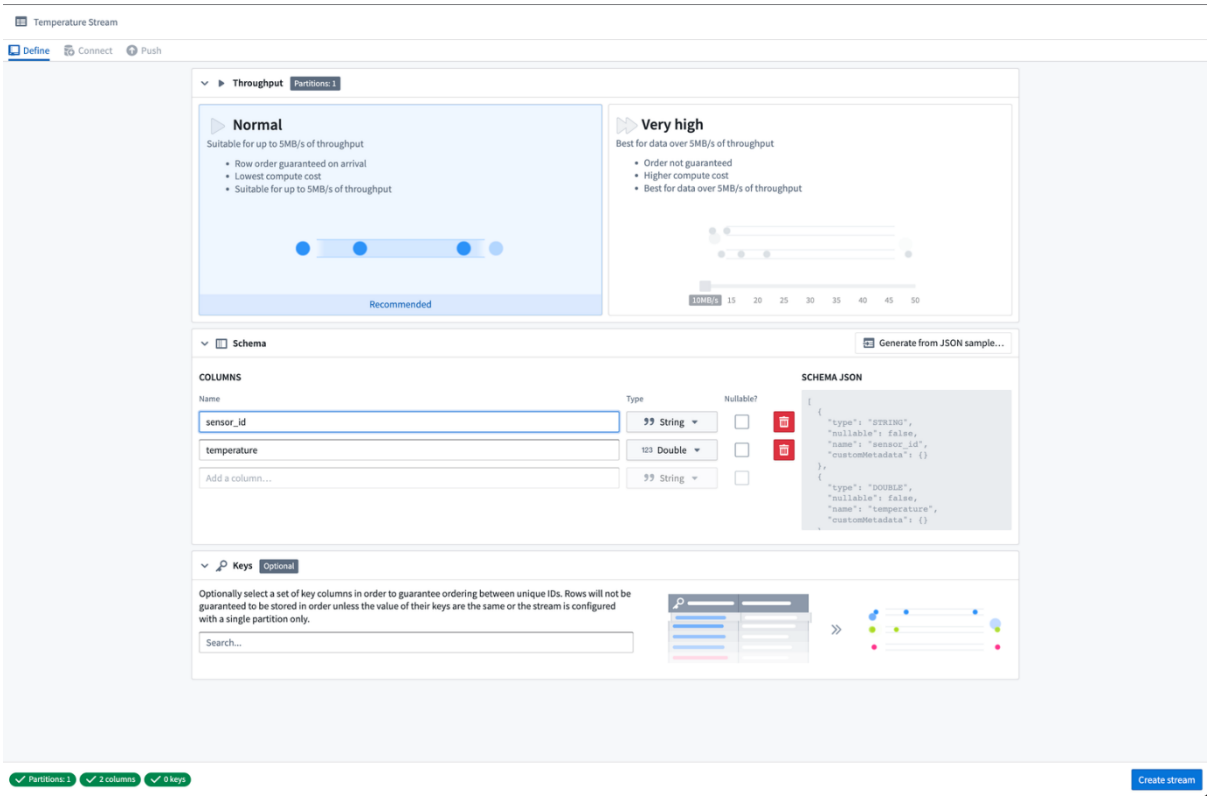
This section will highlight additional Pipeline Builder capabilities, such as streaming pipelines for real-time data processing and directly outputting to Foundry's ontology. Additionally, this section will cover key areas to consider as you advance your pipeline development, namely the use of custom User-Defined Functions (UDFs) for those instances when pre-built transforms don’t fit your requirements, strategies for segmenting your pipeline and determining when to materialize outputs, and ways of maintaining your pipelines long-term.

Other Pipeline Builder Capabilities

Pipeline Builder stands out for its versatility, allowing you to create sophisticated data flows beyond the basic tasks of data cleaning and preprocessing.

Streaming

One of the notable features of Pipeline Builder is its support for **streaming pipelines**, suitable for time-sensitive scenarios, where data needs to be processed with minimal delay. Streaming pipelines process data in (almost) real-time as it is ingested, being processed on average in under 15 seconds.



Outputting to ontology

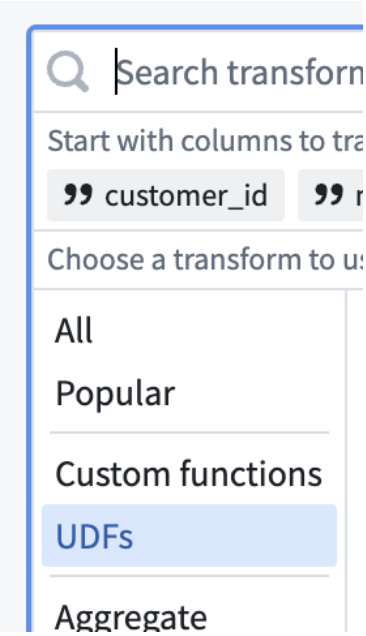
In addition to its data processing capabilities, Pipeline Builder also provides options for the destination of the output. While the creation of datasets is a common outcome, **Pipeline Builder can also direct the output straight into an ontology.** In Foundry, ontologies represent a comprehensive knowledge graph that encapsulates the relationships and connections between various data entities. Directing output to an ontology facilitates the immediate use of structured and relationally rich data for complex analyses and applications.

While this course primarily introduces the foundational elements of data pipeline creation, it's important to be aware that Pipeline Builder's functionality extends to advanced features such as streaming and ontology outputs. These features offer pathways to integrate your data pipeline more deeply with Foundry's extensive data management ecosystem.

User-Defined Functions (UDFs)

The pre-built data transformations in Pipeline Builder are powerful, but they may not meet all the specific requirements of every data engineering task. As an advanced capability of Pipeline Builder, User Defined Functions (UDFs) can be leveraged in your pipeline to define custom functionality written in a programming language supported by Foundry.

After publishing a UDF to Pipeline Builder, you can leverage it in your transforms through the Pipeline Builder boards. Once published, a UDF cannot be unpublished.



Best Practices for UDFs

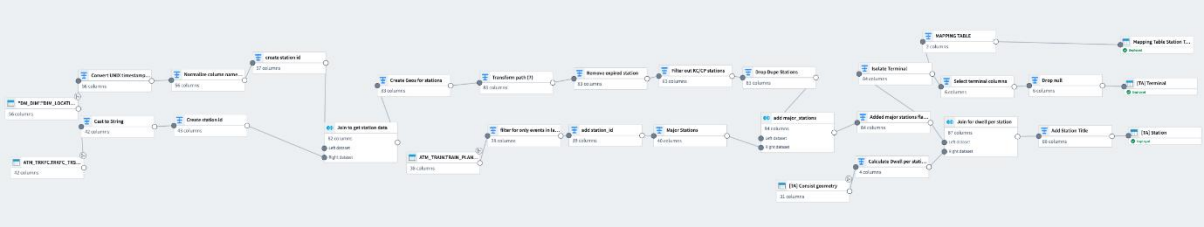
- **Identify the need for UDFs**. Before creating a UDF, ensure that the operation you need cannot be accomplished with the existing built-in transformations. UDFs should be a last resort due to their potential impact on performance and maintainability.
- **Minimize usage.** Only use UDFs when necessary. Leverage built-in functions whenever possible as they are typically optimized for performance.
- **Keep it simple**. Write UDFs that are simple, perform a single task, and minimize overlap with other built-in transformations. Complex UDFs are harder to maintain and can be less efficient.

Best Practices: Segmenting your Pipeline and Materializing Outputs

In practice, your data pipeline may require a vast number of inputs, multiple intermediate steps, or tight connectedness between siloed data points. As the number of nodes grows in your pipeline, the graphical representation in Pipeline Builder can swiftly become overwhelming. In scenarios where your pipeline exceeds 100 nodes, you may struggle with navigation and even encounter difficulties in loading the graph. The key to maintaining manageability lies in strategic segmentation of your pipeline and which outputs you decide to materialize.

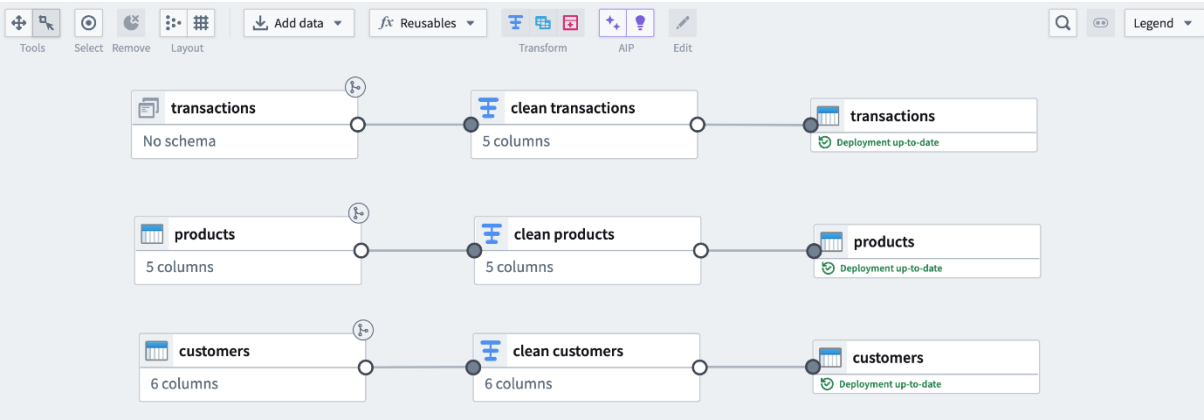
A pipeline that defers the materialization of outputs until the final stages is optimized for

latency. Each output is effectively deployed and built into one single step from the pipeline inputs, so if you have a pipeline that goes through the typical stages of “raw” to “clean” to “joined” to “ontology'', the ontology outputs require only one step of chained intermediary transforms to be deployed and materialized into a Foundry resource. A common approach for organizing your pipeline when optimizing for latency is to split it by use case, where each pipeline generates end to end transformations powering each independent workflows in Foundry.

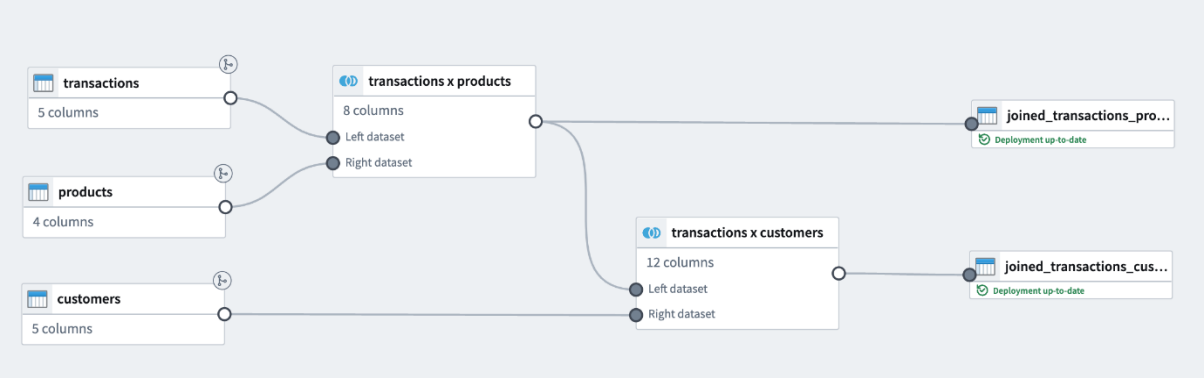


end-to-end use case pipeline

Alternatively, in the context of a large-scale use case with intertwined data points, you may prefer to optimize for extensibility and readability, as opposed to latency. In this case, consider dividing your project into distinct segments, each with its own Pipeline Builder instance, such as one for "cleaning", one for "joining", one for "ontology", and so forth. By materializing the outputs at the conclusion of each segment and then using those outputs as inputs for subsequent stages, you maximize reusability and clarity, at the cost of build latency.



clean pipeline



joined pipeline

Regardless of your approach, you can think of segmenting your pipeline similarly to software development, where code is modularized rather than amalgamated into a single file. By applying similar principles to your data pipeline, you can ensure that each segment is focused, manageable, and tailored to a particular aspect of your data processing.

As a rule of thumb, you can refer to the below best practices:

- **Modularize your pipeline** just as you would with code, dividing your pipelines into logical sections that can be developed, tested, and debugged independently.
- **Materialize judiciously** at key points to create natural breakpoints and to capture intermediary states that might be reusable in other contexts.
- **Test segments** at key points to identify bad joins, wrong aggregations, or incomplete assumptions about the data, to catch errors quickly before they trickle through your pipeline.

By implementing these best practices, you can create a pipeline that is not only optimized for performance but also for development and maintenance, ensuring that your data processing workflows remain robust, scalable, and understandable.

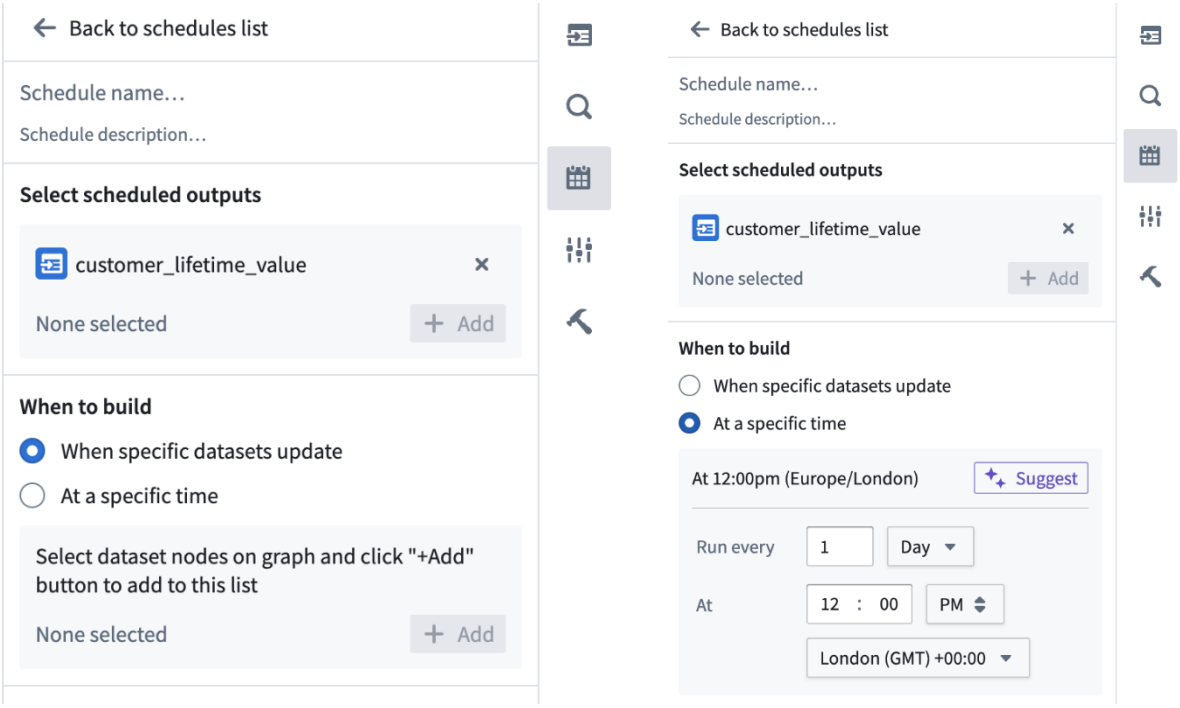
Pipeline Maintainability

As you grow your data infrastructure in Foundry, maintaining the performance and accuracy of your pipelines is essential. Proper maintenance ensures that the pipeline remains reliable, efficient, and up-to-date with evolving data structures and business requirements.

Each of these areas plays a vital role in maintaining a robust and effective data pipeline. By leveraging Pipeline Builder out-of-the-box functionality and adhering to best practices, you can ensure that your pipeline remains a reliable asset for data-driven decision-making.

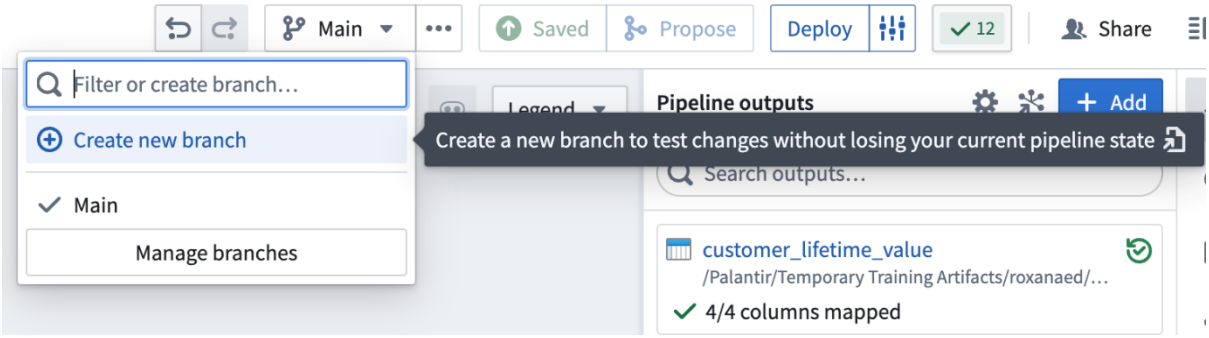
Scheduling

A key next step after implementing the logic of your data processing is implementing a scheduling strategy, to ensure data is refreshed at regular intervals, keeping it current with the source and relevant for the workflows. Pipeline Builder offers out-of-the-box scheduling capabilities that can be configured directly in your pipeline to meet your specific needs.



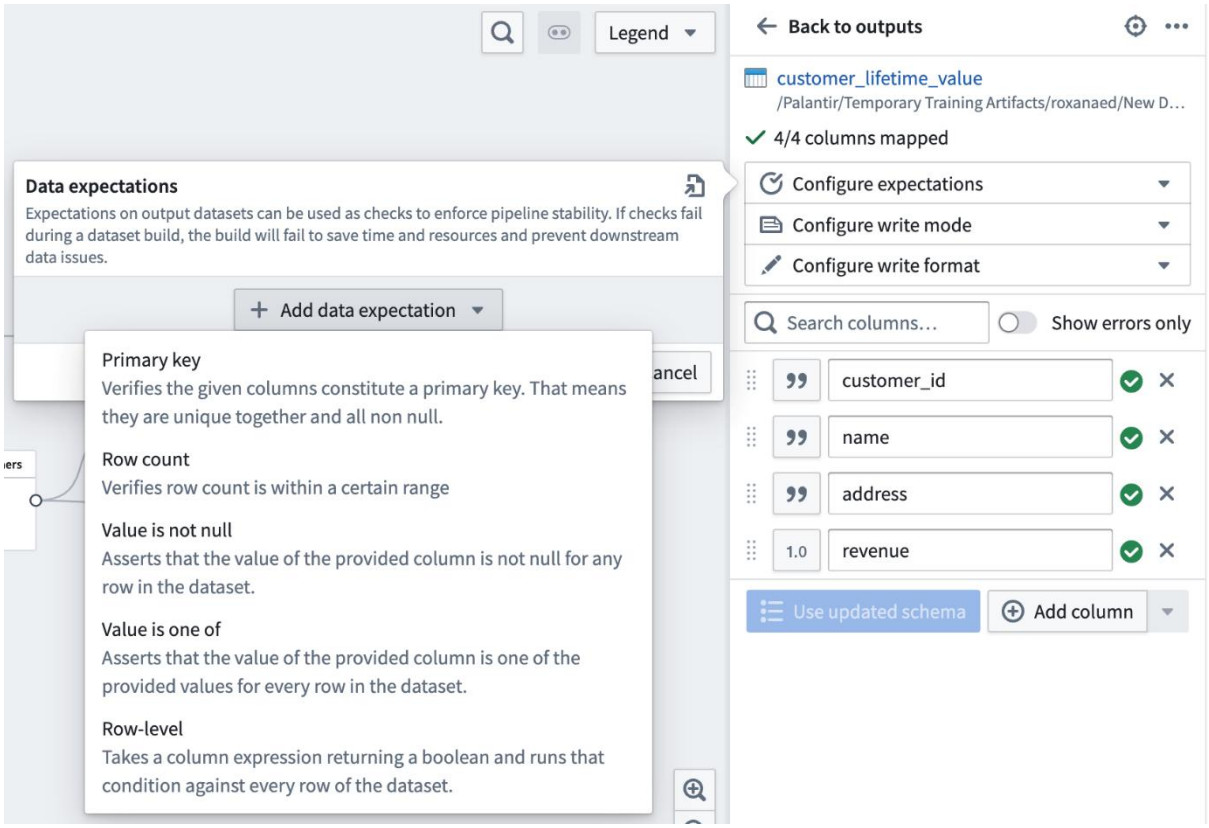
Branching

Once your pipeline is used across multiple workflows, enforcing branching is an important practice for maintaining a stable data pipeline while iterating on new features or updates.



Data Expectations

Setting data expectations, or "data quality checks," helps in proactively monitoring the health of your data. These checks can validate data against predefined criteria, such as data type constraints, value ranges, or uniqueness, to catch anomalies early.



Incremental Transforms

For efficiency, especially with large volumes of data, incremental transforms process only the data that has changed since the last pipeline run, rather than reprocessing the entire dataset. This can significantly reduce the computational load and speed up the time-to-insight.