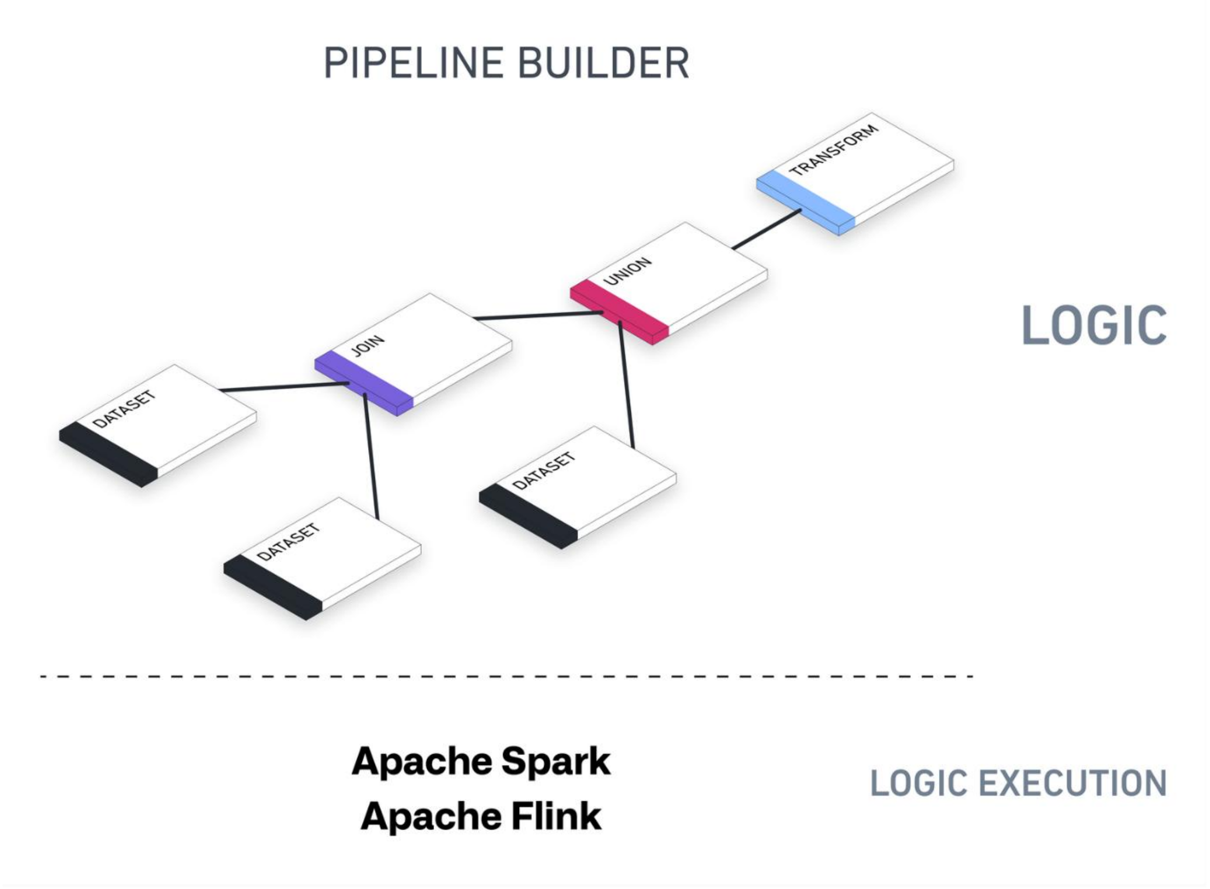
Transforms

Pipeline Builder provides a flexible, powerful, and easy-to-use interface for transforming your data in Foundry. Writing data transformations in existing tooling (for example, in Spark or SQL) can be challenging and error-prone, both for non-coders and experienced software developers. In addition, existing tooling is often coupled to one specific execution engine and requires using a code library to express data transformations.

Pipeline Builder uses a general model for describing data transformations. This backend is an intermediate layer between the tools used to write transformations and the execution of said transformations.

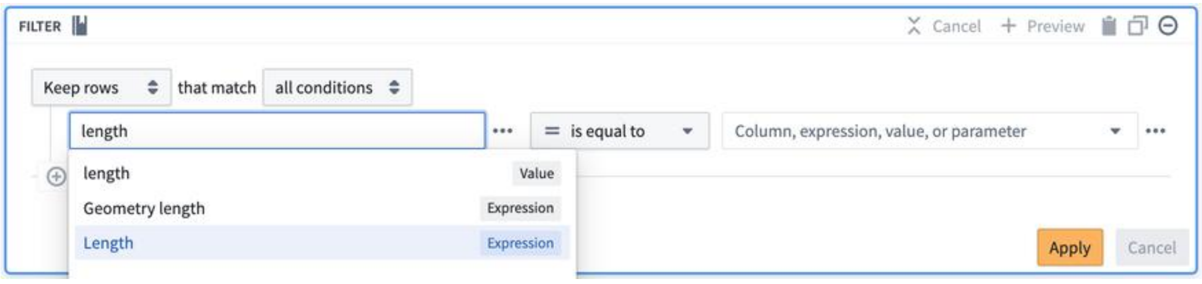


Pipeline Builder's underlying architecture is designed to support all kinds of outputs - datasets, ontological objects, streams, time-series, and exports to external systems. You can run batch pipelines for datasets, object types, link types, or streaming pipelines that correspond to streaming datasets.

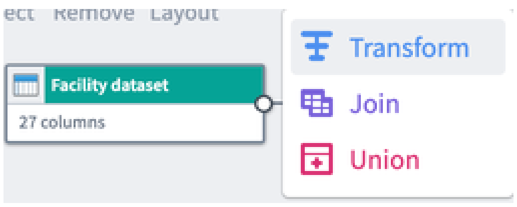
Using transforms in Pipeline Builder

In Pipeline Builder, you can use two types of data transformations: expressions and transforms. Expressions take columns from a table as input and output a single column (for example Split string), while transforms take an entire table as input and return an entire table (for example, Pivot or Filter).

We group expressions and transforms together in the same configuration interface. For example, you can find the Drop columns transform alongside expressions like Cast and Concatenate strings. This allows you to use expressions and transforms together in the same path, and embed expressions within transforms in one configuration form, as shown by inserting the Length expression into the Filter transform below.



Other data structuring transforms, namely **Join** and **Union**, have their own configuration panes and are marked with unique icons in the Pipeline Builder interface.



For simplicity, we typically refer to all types of data transformations as transforms.

Join

A join combines two datasets that have at least one matching column. Depending on the type of join you configure, your join output can combine matching rows and exclude non-matching rows.

Union

A union combines two datasets to include all rows.

The union transform requires all inputs have the same schema. If input schemas do not all match, the union will display an error message with a list of missing columns.

User-defined functions

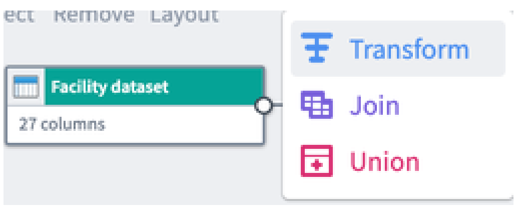
If you cannot manipulate your data with existing transformation options, or have complex logic that you want to reuse across pipelines, you can create a user-defined function (UDF). User-defined functions let you run custom code in Pipeline Builder that can be versioned and upgraded.

Transform data

You can start transforming and structuring your data in Pipeline Builder after adding datasets to your workspace.

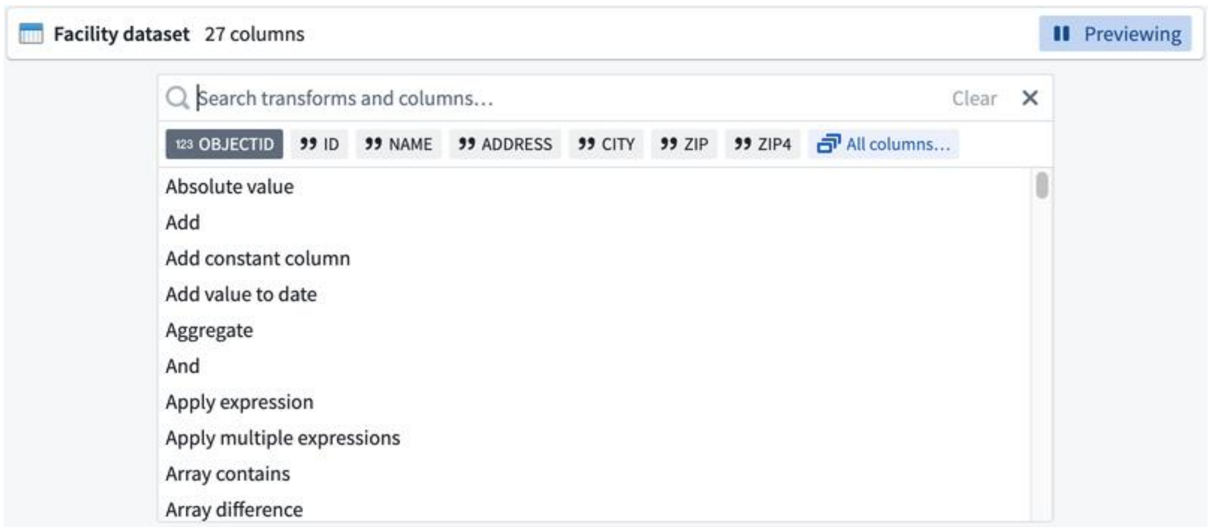
Select a dataset

To apply a transform to a dataset, select a dataset node in your workspace and click **Transform**.



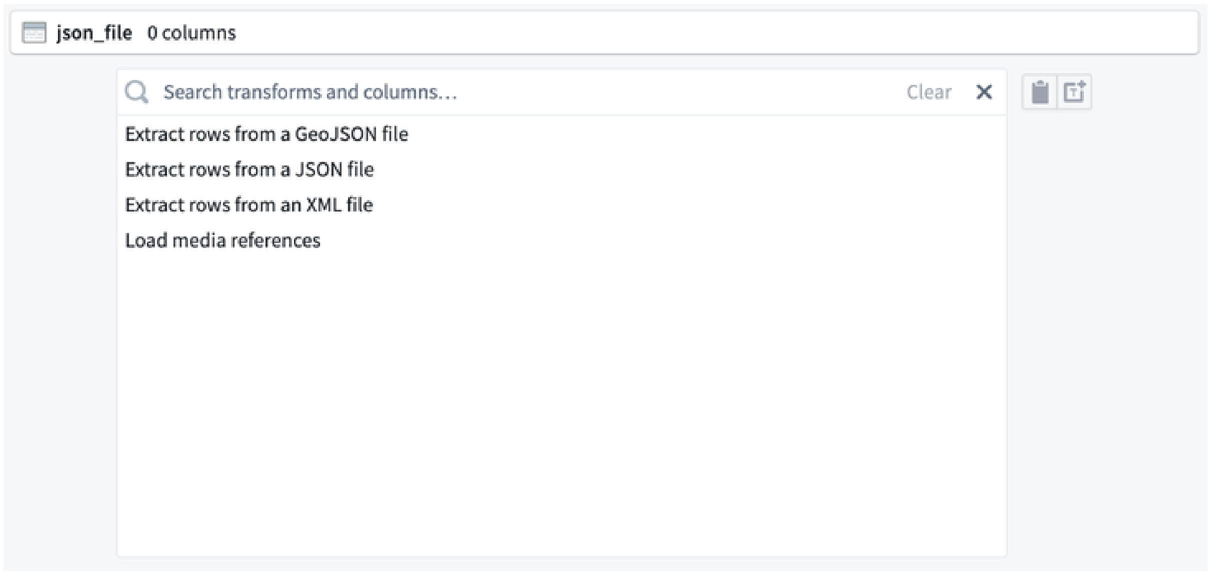
Search for a transform

In the transform page, search for a transform type by name or browse from a list of available transforms. If you are using a structured (tabular) dataset, this field shows a comprehensive list of table transforms.



Absolute value
Add
Add constant column
Add value to date
Aggregate
And
Apply expression
Apply multiple expressions
Array contains
Array difference

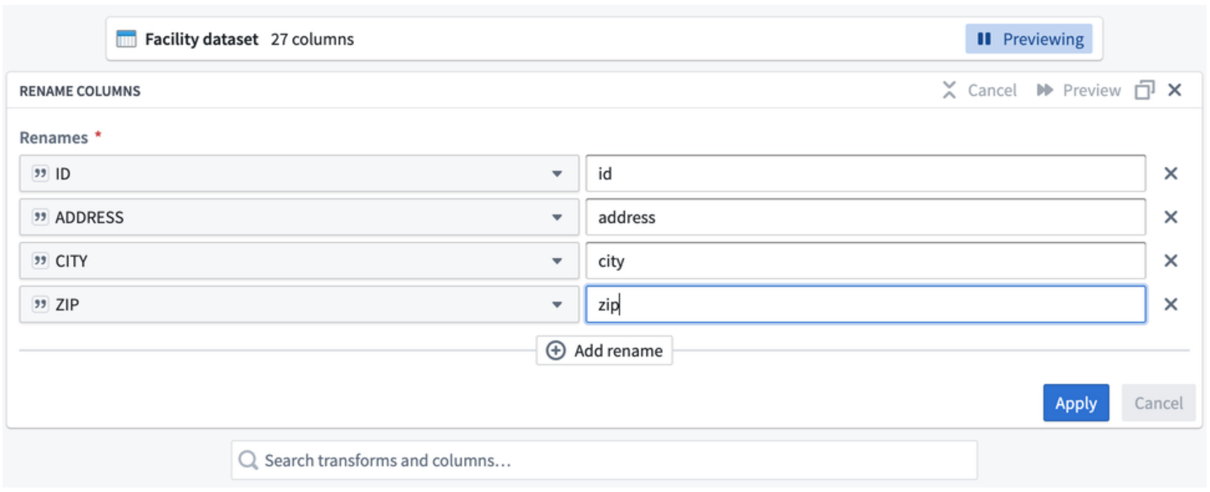
For semi-structured datasets like JSON files, the search field includes file transforms that allow you to parse your dataset into table format.



Extract rows from a GeoJSON file
Extract rows from a JSON file
Extract rows from an XML file
Load media references

Configure a transform

Complete the transform configuration board with required information, including columns, expressions, or values. In the example below, we chose the Rename columns transform, selected columns to rename, and entered new name values for the columns.

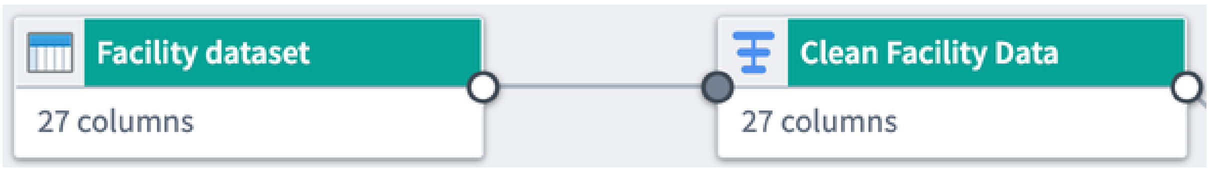


Renames *

ID	id
ADDRESS	address
CITY	city
ZIP	zip

Apply a transform

After completing the transform form, click **Apply** to add the transform to your workflow. You will see the transform node connected to the origin dataset in your graph. We named our new transform Clean Facility Data, and it is a direct output of the original Facility dataset.



You can rename or edit the transform by clicking the transform node and selecting **Edit.**

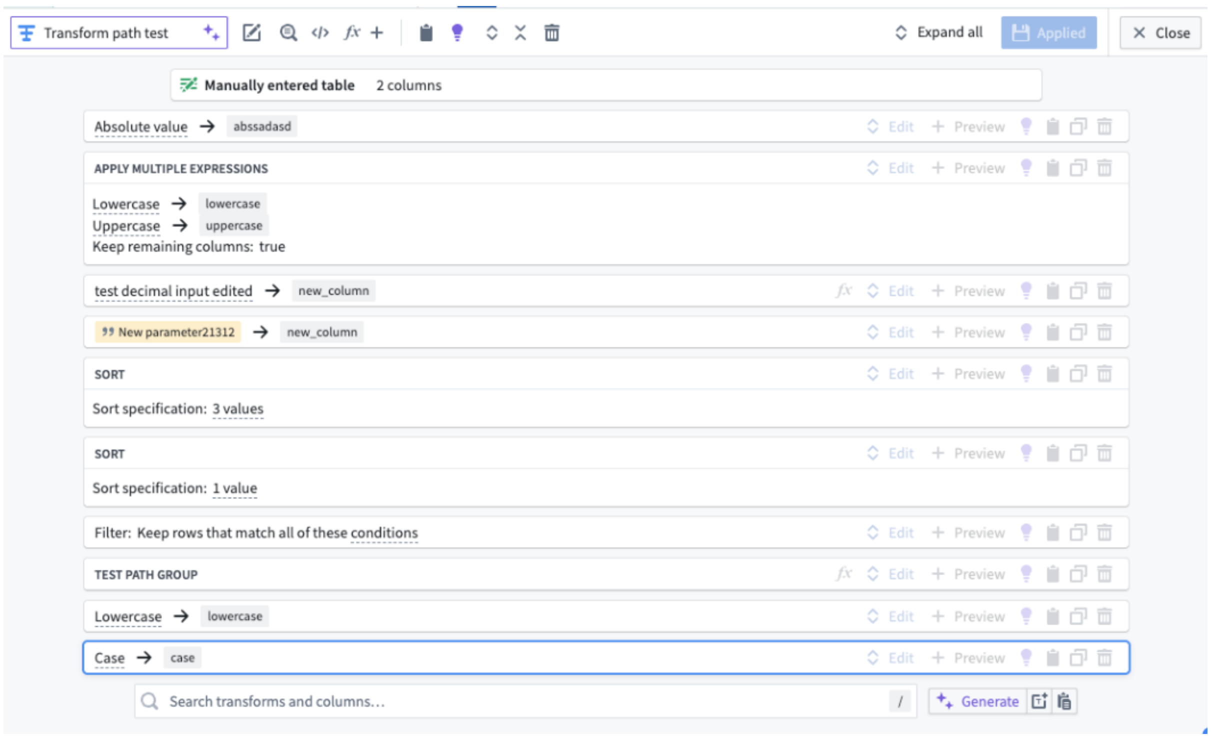
Drag the white output circles on nodes to change connections on the graph.

Transform view

Pipeline Builder offers two ways to view transforms: the traditional collapsed board rendering and pseudocode rendering. Setting your view preference will update your personal view globally across the Palantir platform.

Collapsed board rendering

The collapsed board rendering format displays transforms in a compact, board-like structure. This view is the traditional format and may be preferred by users who are accustomed to this layout.



Absolute value → abssadasd

APPLY MULTIPLE EXPRESSIONS

Lowercase → lowercase
Uppercase → uppercase
Keep remaining columns: true

test decimal input edited → new_column

New parameter21312 → new_column

SORT

Sort specification: 3 values

SORT

Sort specification: 1 value

Filter: Keep rows that match all of these conditions

TEST PATH GROUP

Lowercase → lowercase

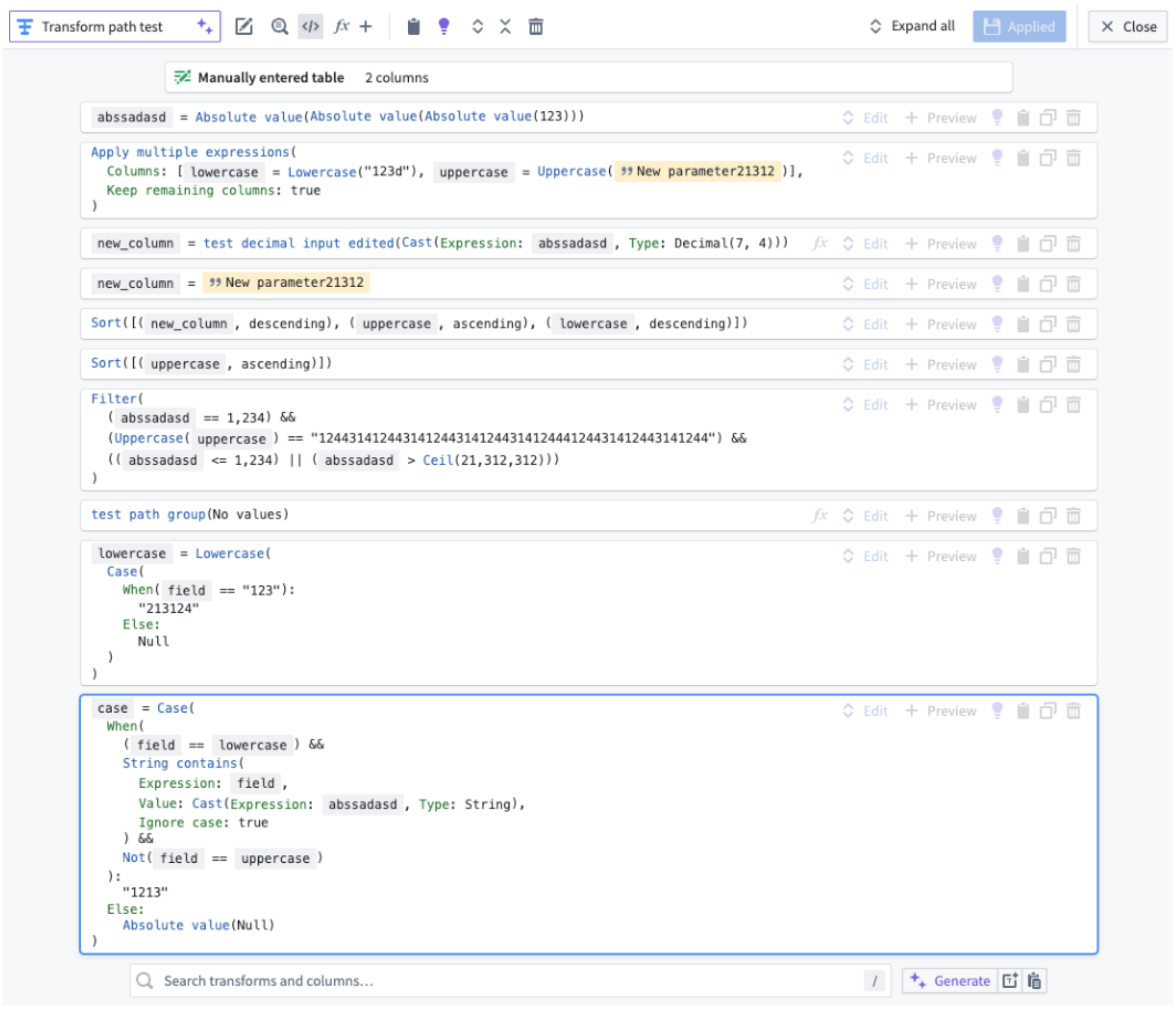
Case → case

Pseudocode rendering

The pseudocode rendering format displays transforms in a cleaner format resembling code but does not adhere to any specific programming language's syntax.

The joins, unions, and LLM nodes are not affected by the pseudocode rendering option.

This option is particularly beneficial for users familiar with coding or those who prefer a more textual representation of their pipeline logic. The pseudocode automatically adjusts to fit your screen, reducing the need for scrolling as well.



```
abssadasd = Absolute value(Absolute value(Absolute value(123)))
```

```
Apply multiple expressions(
  Columns: [ lowercase = Lowercase("123d"), uppercase = Uppercase( ” New parameter21312 )],
  Keep remaining columns: true
)
```

```
new_column = test decimal input edited(Cast(Expression: abssadasd , Type: Decimal(7, 4)))
```

```
new_column = ” New parameter21312
```

```
Sort([( new_column , descending), ( uppercase , ascending), ( lowercase , descending)])
```

```
Sort([( uppercase , ascending)])
```

```
Filter(
  ( abssadasd == 1,234) &&
  (Uppercase( uppercase ) == "1244314124431412443141244314124441244314124431412443141244") &&
  (( abssadasd <= 1,234) || ( abssadasd > Ceil(21,312,312)))
)
```

```
test path group(No values)
```

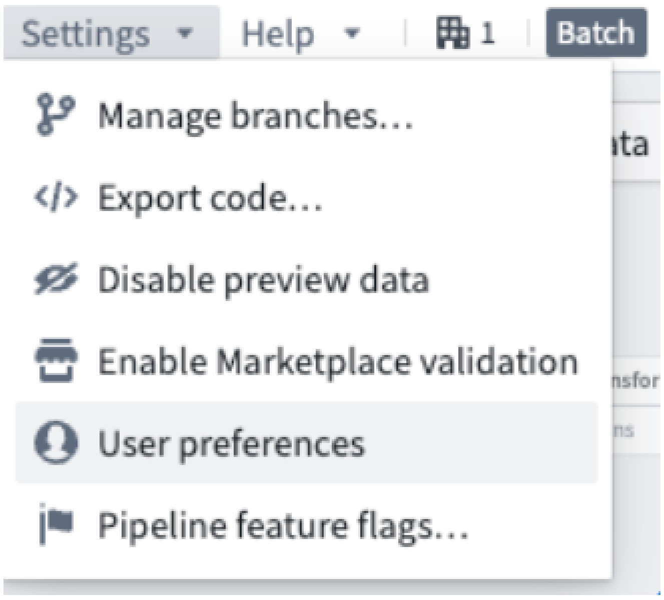
```
lowercase = Lowercase(
  Case(
    When( field == "123"):
      "213124"
    Else:
      Null
  )
)
```

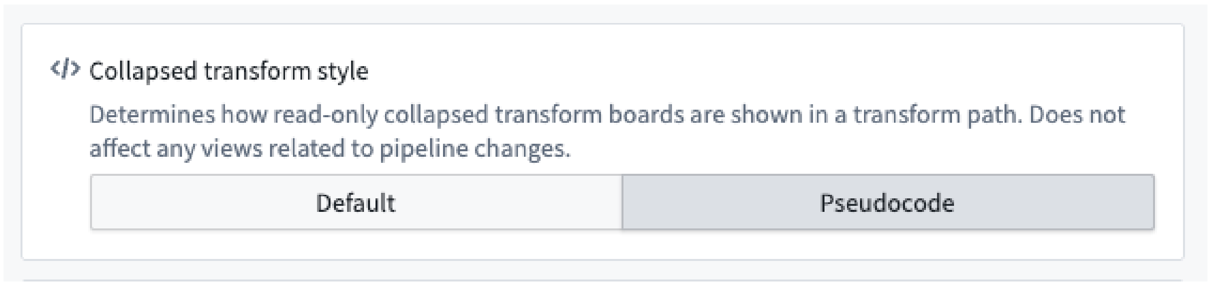
```
case = Case(
  When(
    ( field == lowercase ) &&
    String contains(
      Expression: field ,
      Value: Cast(Expression: abssadasd , Type: String),
      Ignore case: true
    ) &&
    Not( field == uppercase )
  ):
    "1213"
  Else:
    Absolute value(Null)
)
```

Enabling pseudocode rendering does not allow code editing within Pipeline Builder. The format is solely to create a more familiar view.

You can enable the pseudocode rendering option via the following methods:

- **Settings menu:** Navigate to the **Settings** menu. Then, select **User preferences** and toggle the option for **Pseudocode** under **Collapsed transform style**.

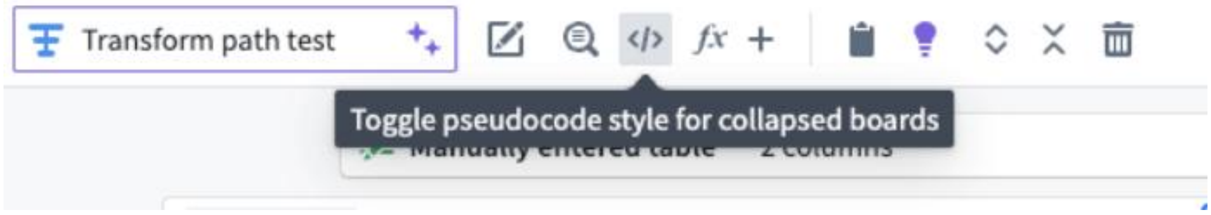



Collapsed transform style

Determines how read-only collapsed transform boards are shown in a transform path. Does not affect any views related to pipeline changes.

Default | Pseudocode

- **Within a transform path:** Within any transform path, select the **</>** icon.



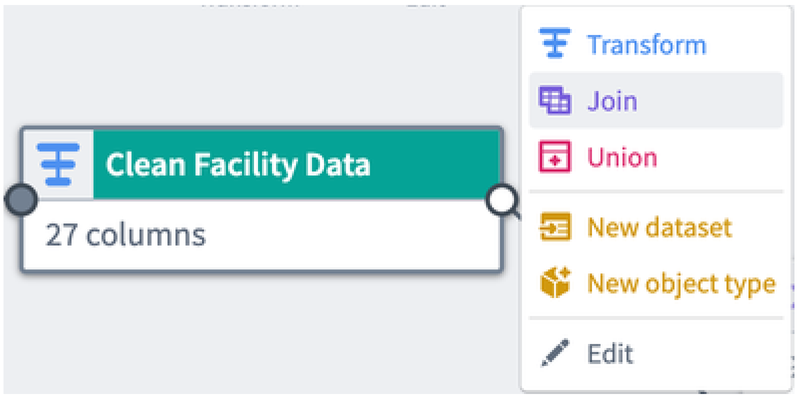
Join data

In addition to transforming single datasets, Pipeline Builder allows you to bring datasets together with joins and unions.

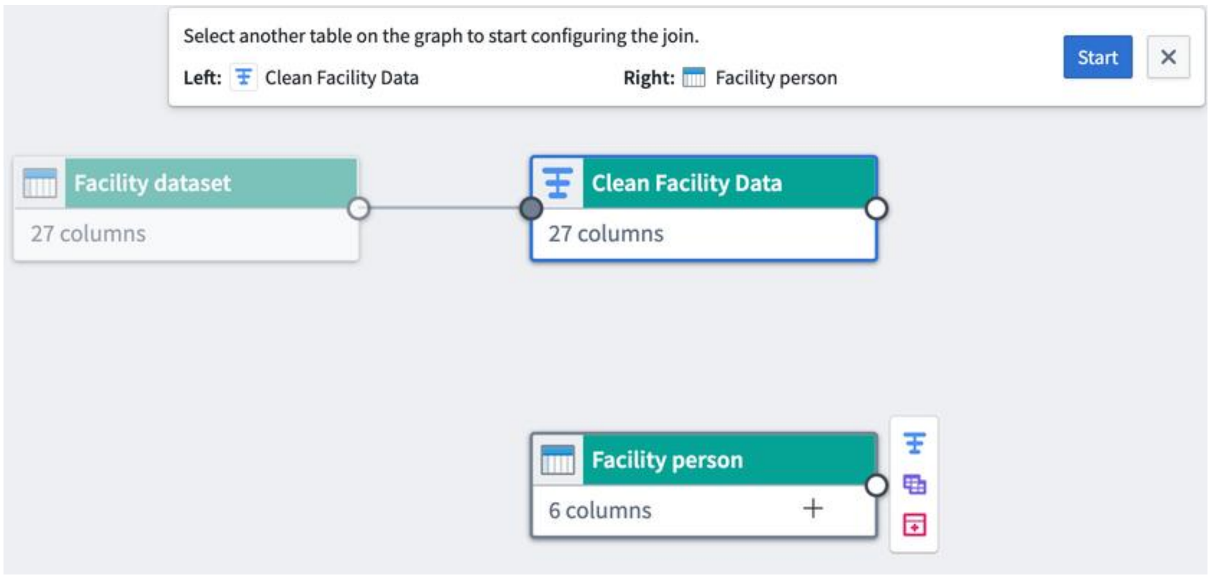
A join combines two datasets that have at least one matching column. Depending on the type of join you configure, your join output can combine matching rows and exclude non-matching rows.

Select datasets

To join two datasets together, select the first dataset node in your graph and click **Join**.



The first selected dataset is the **Left** side dataset. Select another dataset node to be the **Right** side dataset. Click **Start** to configure the join.

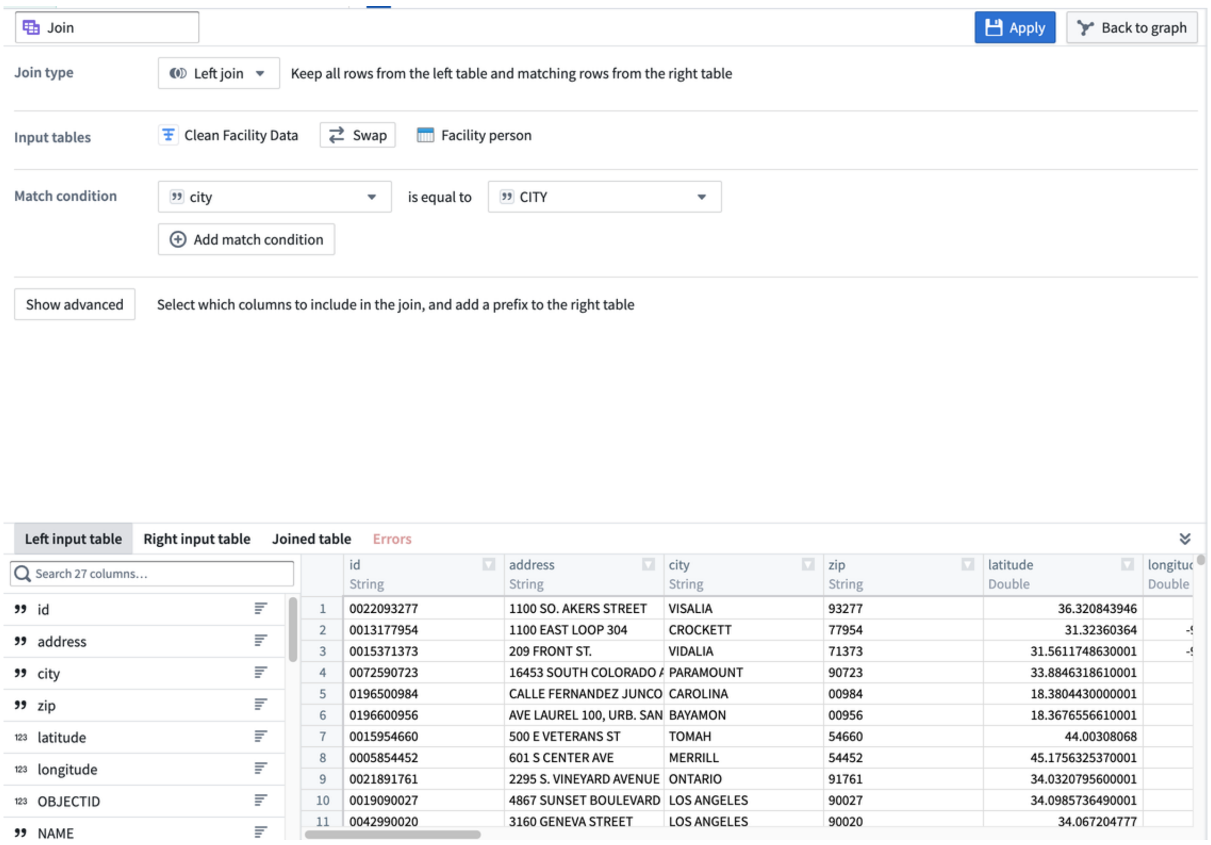

Select another table on the graph to start configuring the join.

Left: Clean Facility Data   Right: Facility person

Configure a join

In the join form, you can edit the join type, select match conditions, and preview the output table.

- **Join type:** Choose whether to create a left, right, inner, or outer join.
    - **Left:** Keep all rows from the left table and matching rows from the right table.
    - **Right:** Keep all rows from the right table and matching rows from the left table.
    - **Inner:** Only keep matching rows between both tables.
    - **Outer:** Keep all rows from both tables, with null filled in columns for non-matching rows.
- **Match condition:** Select a column from the left dataset to mark it as equal to a column from the right dataset. For example, the city column in the left Clean Facility Data dataset is equal to the CITY column in the right Facility Person dataset.
- **Preview:** View preview data from both the right and left input datasets. After applying a join, view preview data from the output table. If any errors occur while applying a join, view them in the **Errors** tab.



Join

Apply | Back to graph

Join type: Left join — Keep all rows from the left table and matching rows from the right table

Input tables: Clean Facility Data | Swap | Facility person

Match condition: city is equal to CITY

Add match condition

Show advanced — Select which columns to include in the join, and add a prefix to the right table

Left input table | Right input table | Joined table | Errors

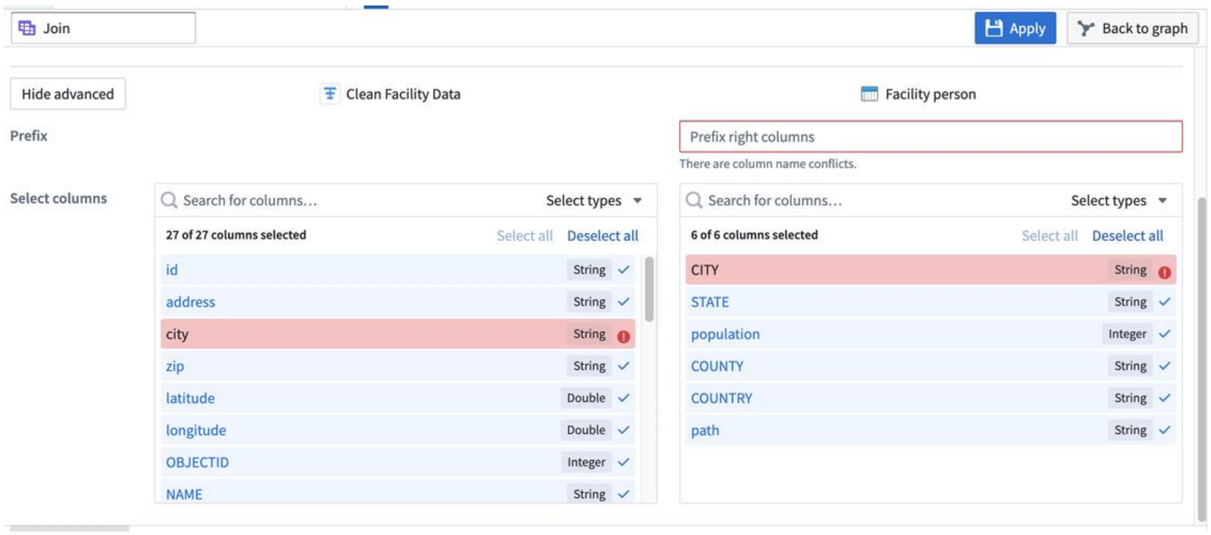
Search 27 columns...

- id
- address
- city
- zip
- latitude
- longitude
- OBJECTID
- NAME

	id String	address String	city String	zip String	latitude Double	longitu Double
1	0022093277	1100 SO. AKERS STREET	VISALIA	93277	36.320843946	
2	0013177954	1100 EAST LOOP 304	CROCKETT	77954	31.32360364	-
3	0015371373	209 FRONT ST.	VIDALIA	71373	31.5611748630001	-
4	0072590723	16453 SOUTH COLORADO A	PARAMOUNT	90723	33.8846318610001	
5	0196500984	CALLE FERNANDEZ JUNCO	CAROLINA	00984	18.3804430000001	
6	0196600956	AVE LAUREL 100, URB. SAN	BAYAMON	00956	18.3676556610001	
7	0015954660	500 E VETERANS ST	TOMAH	54660	44.00308068	
8	0005854452	601 S CENTER AVE	MERRILL	54452	45.1756325370001	
9	0021891761	2295 S. VINEYARD AVENUE	ONTARIO	91761	34.0320795600001	
10	0019090027	4867 SUNSET BOULEVARD	LOS ANGELES	90027	34.0985736490001	
11	0042990020	3160 GENEVA STREET	LOS ANGELES	90020	34.067204777	

All data in the above and following examples was randomly generated and is non-representational.

You can decide to include specific columns in the join and add a prefix to the right table. Select **Show advanced** to expand the prefix and column fields, enter a prefix for the right table, and select the columns to include in the join. In the example below, we are keeping all columns from the left dataset and only including the STATE and population columns from the right dataset.



Join

Apply | Back to graph

Hide advanced

Clean Facility Data | Facility person

Prefix: Prefix right columns
There are column name conflicts.

Select columns

Clean Facility Data — Search for columns... | Select types
27 of 27 columns selected — Select all | Deselect all

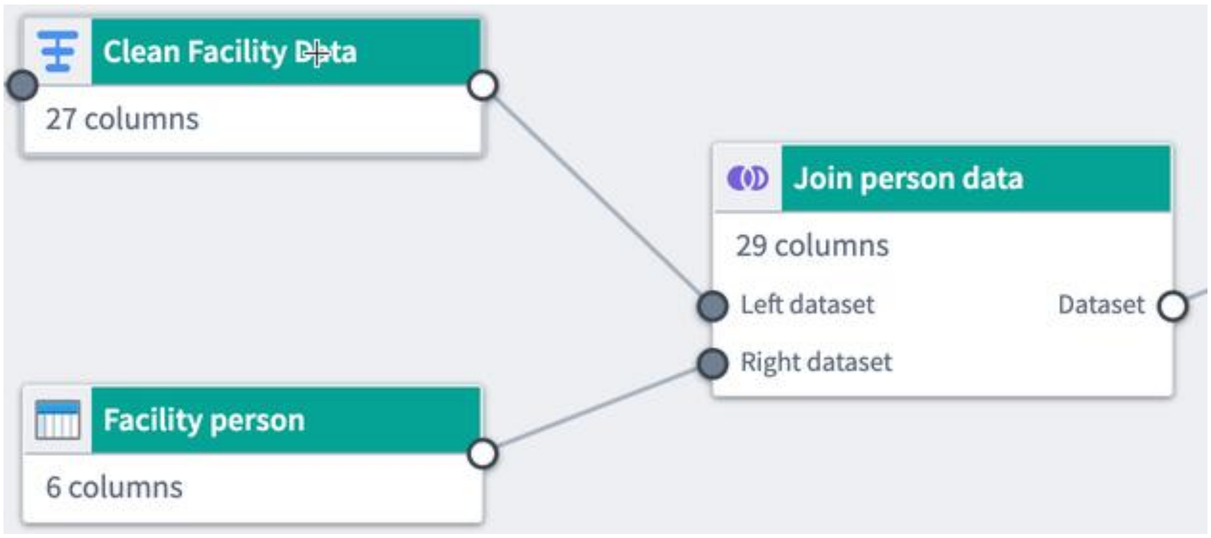
Column	Type	
id	String	✓
address	String	✓
city	String	!
zip	String	✓
latitude	Double	✓
longitude	Double	✓
OBJECTID	Integer	✓
NAME	String	✓

Facility person — Search for columns... | Select types
6 of 6 columns selected — Select all | Deselect all

Column	Type	
CITY	String	!
STATE	String	✓
population	Integer	✓
COUNTY	String	✓
COUNTRY	String	✓
path	String	✓

Apply a join

Once you finish configuring your join, click **Apply** to add the join to your workflow. You will see the join node connected to the two joined datasets in your graph. We named our new join Join person data, and it is a direct output of the original Clean Facility Data and Facility person datasets.



Rename or edit the join by clicking the join node and selecting **Edit**.

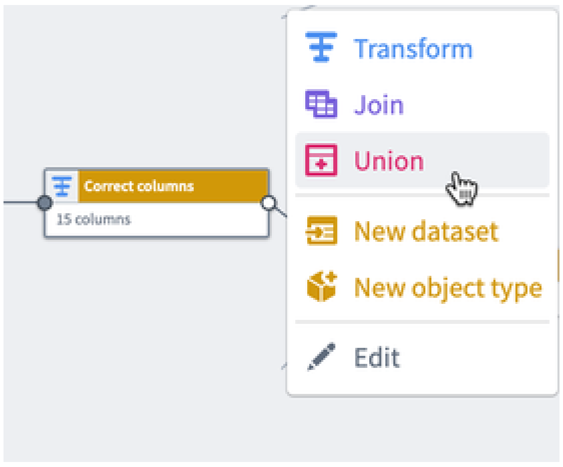
Drag the white or gray circles on nodes to change connections and remove links on the graph.

Union data

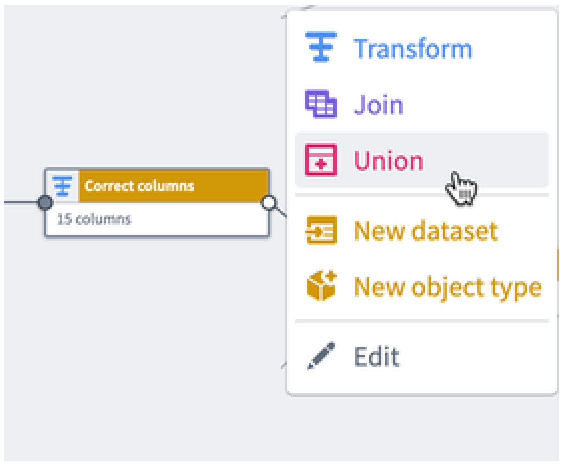
Another way to transform and structure your data in Pipeline Builder is to apply a union. A union combines two datasets to include all rows from each dataset. In Pipeline Builder, a union retains all rows, including duplicates.

Select datasets

To union two datasets together, select the first dataset node in your workspace and click **Union**.

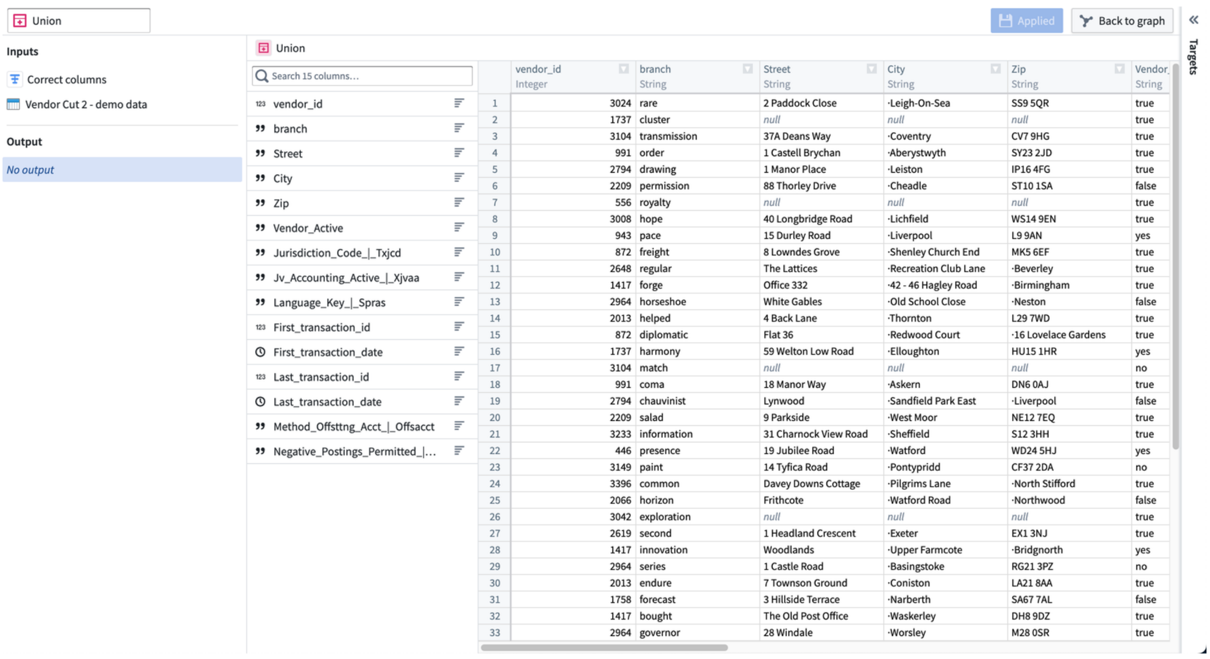


The first selected dataset is the **Left** side dataset. Select another dataset node to be the **Right** side dataset. Click **Start** to navigate to the union output preview page.



Preview a union

In the preview pane, click **Create union**, then view the output dataset preview.



Union

Inputs

- Correct columns
- Vendor Cut 2 - demo data

Output

No output

Union

- vendor_id
- branch
- Street
- City
- Zip
- Vendor_Active
- Jurisdiction_Code_|_Txjcd
- Jv_Accounting_Active_|_Xjvaa
- Language_Key_|_Spras
- First_transaction_id
- First_transaction_date
- Last_transaction_id
- Last_transaction_date
- Method_Offsttng_Acct_|_Offsacct
- Negative_Postings_Permitted_|...

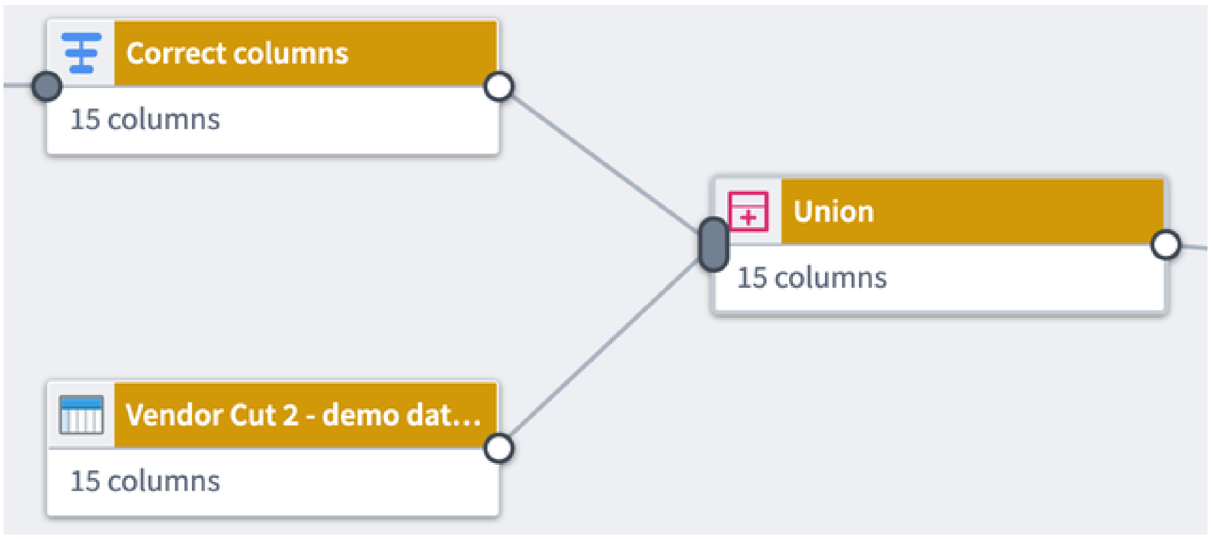
	vendor_id Integer	branch String	Street String	City String	Zip String	Vendor String
1	3024	rare	2 Paddock Close	·Leigh-On-Sea	SS9 5QR	true
2	1737	cluster	*null*	*null*	*null*	true
3	3104	transmission	37A Deans Way	·Coventry	CV7 9HG	true
4	991	order	1 Castell Brychan	·Aberystwyth	SY23 2JD	true
5	2794	drawing	1 Manor Place	·Leiston	IP16 4FG	true
6	2209	permission	88 Thorley Drive	·Cheadle	ST10 1SA	false
7	556	royalty	*null*	*null*	*null*	true
8	3008	hope	40 Longbridge Road	·Lichfield	WS14 9EN	true
9	943	pace	15 Durley Road	·Liverpool	L9 9AN	yes
10	872	freight	8 Lowndes Grove	·Shenley Church End	MK5 6EF	true
11	2648	regular	The Lattices	·Recreation Club Lane	·Beverley	true
12	1417	forge	Office 332	·42 - 46 Hagley Road	·Birmingham	true
13	2964	horseshoe	White Gables	·Old School Close	·Neston	false
14	2013	helped	4 Back Lane	·Thornton	L29 7WD	true
15	872	diplomatic	Flat 36	·Redwood Court	·16 Lovelace Gardens	true
16	1737	harmony	59 Welton Low Road	·Elloughton	HU15 1HR	yes
17	3104	match	*null*	*null*	*null*	no
18	991	coma	18 Manor Way	·Askern	DN6 0AJ	true
19	2794	chauvinist	Lynwood	·Sandfield Park East	·Liverpool	false
20	2209	salad	9 Parkside	·West Moor	NE12 7EQ	true
21	3233	information	31 Charnock View Road	·Sheffield	S12 3HH	true
22	446	presence	19 Jubilee Road	·Watford	WD24 5HJ	yes
23	3149	paint	14 Tyfica Road	·Pontypridd	CF37 2DA	no
24	3396	common	Davey Downs Cottage	·Pilgrims Lane	·North Stifford	true
25	2066	horizon	Frithcote	·Watford Road	·Northwood	false
26	3042	exploration	*null*	*null*	*null*	true
27	2619	second	1 Headland Crescent	·Exeter	EX1 3NJ	true
28	1417	innovation	Woodlands	·Upper Farmcote	·Bridgnorth	yes
29	2964	series	1 Castle Road	·Basingstoke	RG21 3PZ	no
30	2013	endure	7 Townson Ground	·Coniston	LA21 8AA	true
31	1758	forecast	3 Hillside Terrace	·Narberth	SA67 7AL	false
32	1417	bought	The Old Post Office	·Waskerley	DH8 9DZ	true
33	2964	governor	28 Windale	·Worsley	M28 0SR	true

A union requires that all inputs have the same schema. If input schemas do not all match, the union will display an error message with a list of missing columns.

To resolve, remove the references to the missing columns or review your input.

Apply a union

Once you finish creating your union, click **Apply** to add the union to your workflow. You will see the union node connected to the two unioned datasets in your graph. We named our new union Union, and it is a direct output of the original Correct columns and Vendor Cut 2 - demo data datasets.



You can rename or edit the union by clicking the union node and selecting **Edit**.

Drag the white or gray circles on nodes to change connections and remove links on the graph. Click the gray oval on a union node to remove multiple connections.

Remember, a union keeps all rows from both the right and left datasets, including duplicate rows. To remove duplicate rows, add a Drop duplicates transform to your union output.

Create geospatial transforms

With Pipeline Builder, you can load, transform, and wield geospatial data.

Modeling geospatial data

Logical types

Pipeline Builder models geospatial data internally using the concept of a logical type, which is a base type (string, integer, boolean, array, struct) with additional constraints on the data represented. For example, the **Geometry** type is defined as a string which must be valid GeoJSON, while a **GeoPoint** must be a struct of longitude between -180 and 180 and latitude between -90 and 90, both inclusive. A full list of supported types can be found below.

All logical types in Pipeline Builder are inheritors of their base types; for instance, a geometry can be used as input to an expression which expects an input of type string, but not vice versa. To cast from a base type to a particular logical type which extends that base type, you can use the “Logical Type Cast” expression, which will apply the constraints associated with that logical type to the data and null any values which fail this validation. The ability for expressions to specify logical types as input and output ensures that when a geospatial-specific expression expects a GeoJSON string, a GeoJSON string will be received.

Supported geospatial types

Pipeline Builder currently supports the following geospatial types:

- **GeoPoint:** A struct of longitude and latitude, where longitude is a double between -180 and 180, and latitude is a double between -90 and 90, both inclusive. A GeoPoint must be a valid (x, y) coordinate according to the WGS:84 or EPSG:4326 coordinate reference system (CRS).

- **Geometry:** A stringified JSON blob adhering to the GeoJSON specification. Individual coordinates are expected to be in WGS:84/EPSG:4326 format, like the GeoPoint type.
- **H3Index:** A string which represents a valid H3 hexagonal index.
- **LatLonBoundingBox:** A bounding box, represented by a struct of minLat, minLon, maxLat, maxLon, where each entry is a valid GeoPoint and where maxLat > minLat and maxLon > minLon.
- **Ontology GeoPoint:** A string compatible with the Ontology's GeoPoint property type, fulfilling the format {lat},{lon}, where -90 <= lat <= 90 and -180 <= lon <= 180.
- **MGRS:** A string which represents a valid MGRS (Military Grid Reference System) coordinate.

Loading geospatial data

Pipeline Builder supports a variety of different transforms and expressions for geospatial data.

- GeoPoint:
    - **Construct GeoPoint column:** Takes a lat,lon pair, validates the bounds outlined above, and converts it into GeoPoint representation.
    - **Create GeoPoint from Coordinate System (CRS):** Takes an x,y pair and a coordinate reference system, projects that (x,y) into WGS:84, then constructs a GeoPoint representation. Supports conversion from most coordinate systems in the EPSG database, including all UTM zones.
- Geometry:
    - **Parse well-known text (WKT):** Converts well-known text (WKT) string to the geometry logical type. Optionally, supply a source coordinate system identifier to convert from the source CRS to WGS:84 if the WKT is not in WGS:84 already.
    - **Normalize Geometry:** Given a GeoJSON string in WGS:84 format, normalizes the following attributes: proper order (right-hand rule), closed rings, duplicate removal, and constant dimensions of points.
    - **Extract rows from Shapefile:** Given a dataset of raw shapefiles, parses each shapefile into rows containing each entry’s geometry and properties. The output dataset will have a geometry column as well as a column for each property listed by the user. Coordinate reference systems can be specified for non-WGS:84 datasets.
    - **Extract rows from GeoJSON:** Given a dataset of raw GeoJSON files, parses each shapefile into rows containing each entry’s geometry and properties. The output dataset will have a geometry column as well as a column for each property listed by the user. Coordinate reference systems can be specified for non-WGS:84 datasets.

Additional expressions exist to translate between the above two types, as well as to convert them to H3 indices, MGRS, bounding boxes, and the Ontology GeoPoint format.

Transforming geospatial data

Once you have populated your columns of Pipeline Builder's geospatial types, you can take advantage of transforms that operate specifically on geospatial data. Most transforms (except for geo-joins) are currently supported in both streaming and batch workflows. Some highlights are listed below.

Geometry comparisons

- Intersection
- Difference
- Symmetric difference
- Union (column-wise and aggregate)

Spherical geometry

- Haversine/great circle distance between two points
- Inverse haversine distance (given a starting point, distance, and bearing angle, calculate the end point)
- Area/centroid/length of a geometry

H3

- Get neighbors of H3 hexagon at a certain resolution
- Cover a polygon with H3 hexagons at a certain resolution

Complex shape approximation

- Ellipse/Circle
- Range fan (annulus sector)
- Convex Hull of a given geometry

Geospatial joins

Pipeline Builder supports the following geospatial joins:

- Geometry intersection joins
- Geometry distance joins
- Geometry nearest neighbor joins

Geometry intersection joins

Pipeline Builder's geometry intersection join requires two datasets, each of which must have a geometry typed column. The geometry intersection join does not accept Ontology GeoPoint or GeoPoint as an input type. Before applying the join, we recommend normalizing the geometry column and explicitly filtering out null values if they are not needed in the output. If there is non-determinism or another join in the pipeline, we recommend adding a checkpoint prior to the geojoin.

Pipeline Builder can join datasets of medium-sized geometries (approximately up to 34 points) with a scale of up to 1 million rows on either side, assuming a twofold increase in the number of output rows. For skewed data, the join can support up to 250 million rows on one side against 1.6 thousand rows on the other. Stability may degrade as the size of the geometries increases. The join can consistently support joining a dataset with one massive geometry (on the order of 40k points) against up to 500k rows. Any larger scale may succeed intermittently but is not officially supported.

Geometry intersection joins that have a number of rows in the output comparable to that of a cross join can cause stability degradation in the join.

As an alternative to the geometry intersection join, the cross join configured with the "Geometries have intersection" filter may provide more stable memory usage. However, this approach could lead to a sharp increase in build times.

Geometry distance joins

Pipeline Builder's geometry distance join requires two datasets, each of which must have a geometry typed column, a value for distance greater than zero, and a coordinate reference system string which will determine the units of the distance provided. For example, if "epsg:4326" is provided for the coordinate reference system, then the distance will be assumed to be in units of degrees. Similar to the intersection join, we recommend normalizing the geometry column, and explicitly filtering out null values if they are not needed in the output. If there is another join or non-determinism in the pipeline, add a checkpoint prior to the join.

Pipeline Builder can join datasets of small geometries (approximately up to 8 points each) with a scale of up to 1 million rows on either side, assuming a 2x increase in the number of rows as a result of the join. When the number of rows output is comparable to that of a cross join, stability may degrade.

An alternative to the geometry distance join, a cross join configured with a geometry buffer and "Geometries have intersection" filter may provide more stable memory usage when the increase in row count is large. However, this approach could sharply increase build times in most cases.

Geometry k-nearest neighbors (KNN) joins

Pipeline Builder's geometry nearest neighbors join requires two datasets: a base dataset of geometries and a neighbors dataset of points. The k integer parameter configures the number of nearest neighbors to find for each base geometry. A coordinate reference system is required to determine how distances between base geometries and neighbor points are calculated and compared. The result will be the set of combined rows, each of which contains a GeoPoint that is one of the k closest points to the base geometry. Ties are broken arbitrarily, and results are returned in no particular order.

Note that this join has two requirements:

- All GeoPoints in the neighbors dataset must be able to fit inside executor and driver memory. This is currently a hard requirement and limits the scalability of the join. Contact your Palantir representative if your use case requires distributing the neighbors dataset.

- Foundry currently only accepts the GeoPoint logical type in the neighbors dataset to limit memory consumption. Contact your Palantir representative if non-point geometries are required on the neighbors side of the join.

In practice, Pipeline Builder supports modest values of k (< 5) with up to a few hundred thousand rows in the neighbors dataset and 1 million geometries in the base dataset. When both datasets have a few hundred thousand rows, Pipeline Builder can support much larger values of k. Finding up to several hundred nearest neighbors should finish quickly in such cases. Increasing the scale of the inputs beyond this point may succeed intermittently, but is not currently supported in general.

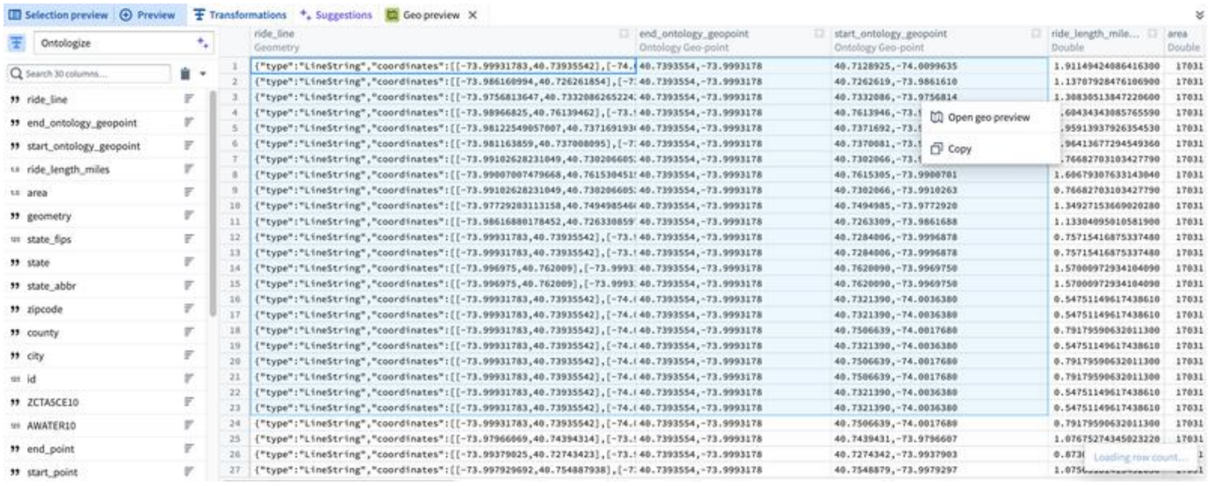
Troubleshooting

If your join is encountering stability issues, use the following steps to remediate:

1. Drop unnecessary columns prior to the join.
2. Simplify the input geometries (for example, can you use a coarser grain for large geometries?)
3. Scale vertically; manually select a compute profile with more memory for the driver and executors.
4. Split the largest input dataset into sets of about 25 million rows, then union the results together in a separate build.
5. Reduce the number of rows in the output (that is, the number of intersections between left and right geometries).

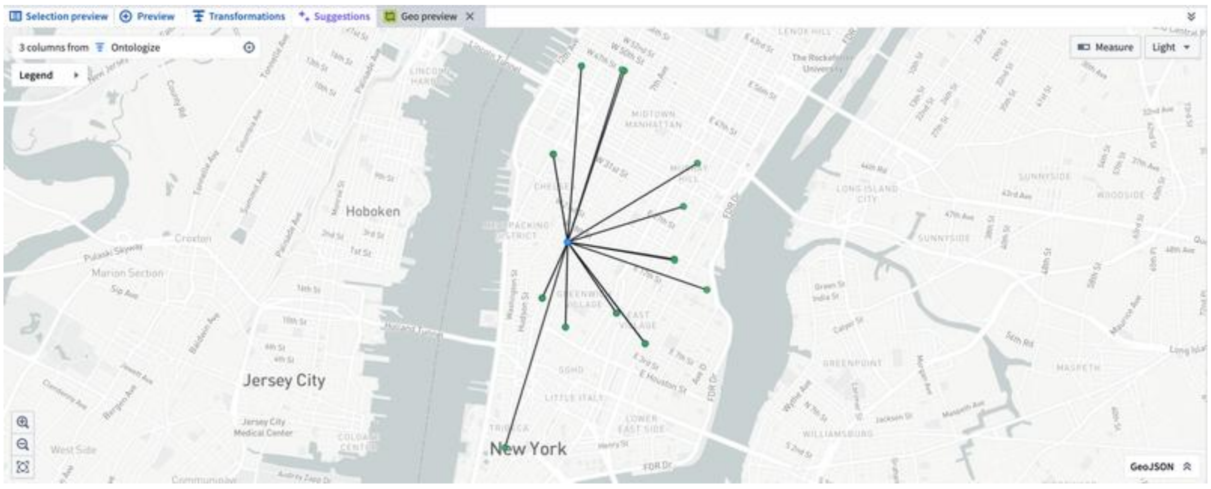
Preview transform results

Once you have finished transforming your data in Pipeline Builder, you can validate the results of these transforms visually on a map. In the regular preview pane, select the cells you would like to preview on a map (the cells must be from columns of one of the geospatial types mentioned above). Right-click and select **Open Geo Preview**.



	ride_line Geometry	end_ontology_geopoint Ontology Geo-point	start_ontology_geopoint Ontology Geo-point	ride_length_mile... Double	area Double
1	{"type":"LineString","coordinates":[[-73.99931783,40.73935542],[-74.	40.7393554,-73.9993178	40.7128925,-74.0099635	1.91149424086416300	17031
2	{"type":"LineString","coordinates":[[-73.986160994,40.726261854],[-7	40.7393554,-73.9993178	40.7262619,-73.9861610	1.13707928476106900	17031
3	{"type":"LineString","coordinates":[[-73.9756813647,40.7332086265224	40.7393554,-73.9993178	40.7332086,-73.9756814	1.30830513847220600	17031
4	{"type":"LineString","coordinates":[[-73.98966825,40.76139462],[-73.	40.7393554,-73.9993178	40.7613946,-73.9	.60434343085765590	17031
5	{"type":"LineString","coordinates":[[-73.98122549057007,40.737169193	40.7393554,-73.9993178	40.7371692,-73.9	.95913937926354530	17031
6	{"type":"LineString","coordinates":[[-73.981163859,40.737008095],[-7	40.7393554,-73.9993178	40.7370081,-73.9	.96413677294549360	17031
7	{"type":"LineString","coordinates":[[-73.99102628231049,40.730206605	40.7393554,-73.9993178	40.7302066,-73.9	.76682703103427790	17031
8	{"type":"LineString","coordinates":[[-73.99007007479668,40.761530451	40.7393554,-73.9993178	40.7615305,-73.9900701	1.60679307633143040	17031
9	{"type":"LineString","coordinates":[[-73.99102628231049,40.730206605	40.7393554,-73.9993178	40.7302066,-73.9910263	0.76682703103427790	17031
10	{"type":"LineString","coordinates":[[-73.97729203113158,40.749498546	40.7393554,-73.9993178	40.7494985,-73.9772920	1.34927153669020280	17031
11	{"type":"LineString","coordinates":[[-73.98616880178452,40.726330859	40.7393554,-73.9993178	40.7263309,-73.9861688	1.13304095010581900	17031
12	{"type":"LineString","coordinates":[[-73.99931783,40.73935542],[-73.	40.7393554,-73.9993178	40.7284006,-73.9996878	0.75715416875337480	17031
13	{"type":"LineString","coordinates":[[-73.99931783,40.73935542],[-73.	40.7393554,-73.9993178	40.7284006,-73.9996878	0.75715416875337480	17031
14	{"type":"LineString","coordinates":[[-73.996975,40.762009],[-73.9993	40.7393554,-73.9993178	40.7620090,-73.9969750	1.57000972934104090	17031
15	{"type":"LineString","coordinates":[[-73.996975,40.762009],[-73.9993	40.7393554,-73.9993178	40.7620090,-73.9969750	1.57000972934104090	17031
16	{"type":"LineString","coordinates":[[-73.99931783,40.73935542],[-74.	40.7393554,-73.9993178	40.7321390,-74.0036380	0.54751149617438610	17031
17	{"type":"LineString","coordinates":[[-73.99931783,40.73935542],[-74.	40.7393554,-73.9993178	40.7321390,-74.0036380	0.54751149617438610	17031
18	{"type":"LineString","coordinates":[[-73.99931783,40.73935542],[-74.	40.7393554,-73.9993178	40.7506639,-74.0017680	0.79179590632011300	17031
19	{"type":"LineString","coordinates":[[-73.99931783,40.73935542],[-74.	40.7393554,-73.9993178	40.7321390,-74.0036380	0.54751149617438610	17031
20	{"type":"LineString","coordinates":[[-73.99931783,40.73935542],[-74.	40.7393554,-73.9993178	40.7506639,-74.0017680	0.79179590632011300	17031
21	{"type":"LineString","coordinates":[[-73.99931783,40.73935542],[-74.	40.7393554,-73.9993178	40.7506639,-74.0017680	0.79179590632011300	17031
22	{"type":"LineString","coordinates":[[-73.99931783,40.73935542],[-74.	40.7393554,-73.9993178	40.7321390,-74.0036380	0.54751149617438610	17031
23	{"type":"LineString","coordinates":[[-73.99931783,40.73935542],[-74.	40.7393554,-73.9993178	40.7321390,-74.0036380	0.54751149617438610	17031
24	{"type":"LineString","coordinates":[[-73.99931783,40.73935542],[-74.	40.7393554,-73.9993178	40.7506639,-74.0017680	0.79179590632011300	17031
25	{"type":"LineString","coordinates":[[-73.97966069,40.74394314],[-73.	40.7393554,-73.9993178	40.7439431,-73.9796607	1.07675274345023220	17031
26	{"type":"LineString","coordinates":[[-73.99379025,40.72743423],[-73.	40.7393554,-73.9993178	40.7274342,-73.9937903	0.873	1
27	{"type":"LineString","coordinates":[[-73.997929692,40.754887938],[-7	40.7393554,-73.9993178	40.7548879,-73.9979297	1.075	[illegible]

A new preview tab will appear, displaying the selected cells plotted on a map.



Using geospatial data with the Ontology

Pipeline Builder's geospatial capabilities are designed to integrate seamlessly with downstream data across the platform.

- Ontology
  - Builder's geometry column type is compatible with the ontology's geoshape type, but make sure to apply the "Normalize geometry" expression before mapping your column into an object in Builder. This ensures that the geoshape data will pass validations performed while indexing data into the ontology.
  - While the current GeoPoint logical type cannot be used directly in the ontology, points can be easily converted to an "Ontology GeoPoint" type (the equivalent of the ontology's geohash type) prior to indexing.
- Datasets
  - Geospatial type data is persisted on output datasets on Builder pipelines, so that if you create a downstream Builder pipeline from that dataset, your data will still preserve its correct logical/geospatial types.
- Geotemporal series syncs
  - You can map GeoPoint and geometry columns to a geotemporal series sync output for rendering points and geometries in downstream applications, such as the Map application. All geotemporal series sync outputs must have a GeoPoint column indicating the position of the observation corresponding to each row.

Create unique IDs in Pipeline Builder

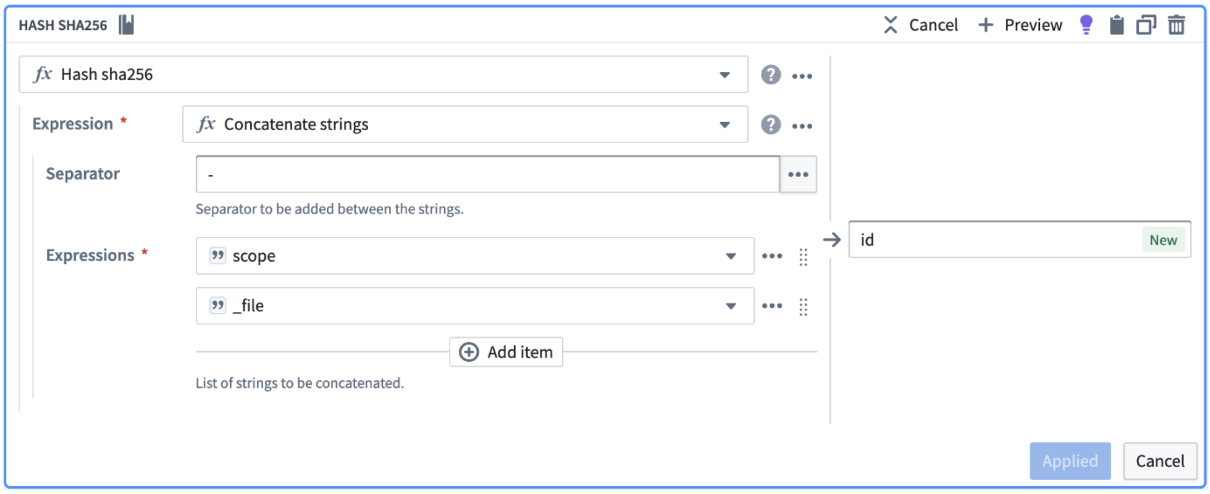
In Pipeline Builder, unique IDs facilitate tracking, processing, and analysis of the data, ensuring that each record can be individually identified and properly handled. For this reason, it is often necessary to create unique identifiers (IDs) for records. This section explains why using a monotonically increasing ID is not the best approach and why the preferred method for generating unique IDs is the concatenation of string columns followed by a SHA256 hash.

Using Concatenation of String Columns and SHA256 Hash

The best approach to generate unique IDs is to concatenate string columns from the input data and then create a SHA256 hash of the concatenated string.

To generate unique IDs using this method in Pipeline Builder, follow these steps within the Pipeline Builder transform path:

1. Identify the string columns that, when combined, can uniquely identify each record in your dataset.
2. Concatenate the selected string columns to form a single string for each record.
3. Use “Hash sha256” to compute the SHA256 hash of the concatenated string. The resulting 256-bit hash can be represented as a 64-character hexadecimal string, which will serve as the unique ID for each record.


HASH SHA256

Hash sha256

Expression * Concatenate strings

Separator: -

Separator to be added between the strings.

Expressions *: scope, _file

List of strings to be concatenated.

id (New)

This method has several advantages:

- **Consistency**: The same input data will always result in the same unique ID, ensuring consistency across different runs of the data pipeline. This makes it easier to track records, identify duplicates, and perform data reconciliation. Notably, if the IDs are used as primary keys for objects, you do not want those primary keys to change as a result of rebuilding the pipeline. Additionally, consider whether someone working on the data downstream at any point might depend on the IDs to be stable.
- **Distributed Generation**: Since the unique ID is derived from the data itself, multiple processes can generate unique IDs concurrently without the need for synchronization or centralized coordination. This improves scalability and performance in a distributed data processing environment.

By using the concatenation of string columns followed by a SHA256 hash, you can generate unique IDs that are scalable, secure, and consistent, making it an ideal choice for your data pipeline application.

Disadvantages of monotonically increasing IDs

While monotonically increasing IDs are not supported in Pipeline Builder, they are often used by data engineers who are familiar with Spark. Monotonically increasing IDs are generated sequentially, such as 1, 2, 3, and so on. While this approach has an inherent simplicity, it has several disadvantages:

- **Inconsistency between builds**: When using monotonically increasing IDs in Spark, the generated IDs can change between different runs of the same application. This is because the way Spark assigns tasks to its executors can vary, leading to different ID assignment orders. Consequently, this inconsistency can make it difficult to reproduce results, compare different runs, or perform incremental updates, making it a less reliable choice for an ID column. If used as a primary key to an ontology object, this will force a full re-index on every build.
- **Reliance on State**: Generating monotonically increasing IDs requires maintaining state between rows.

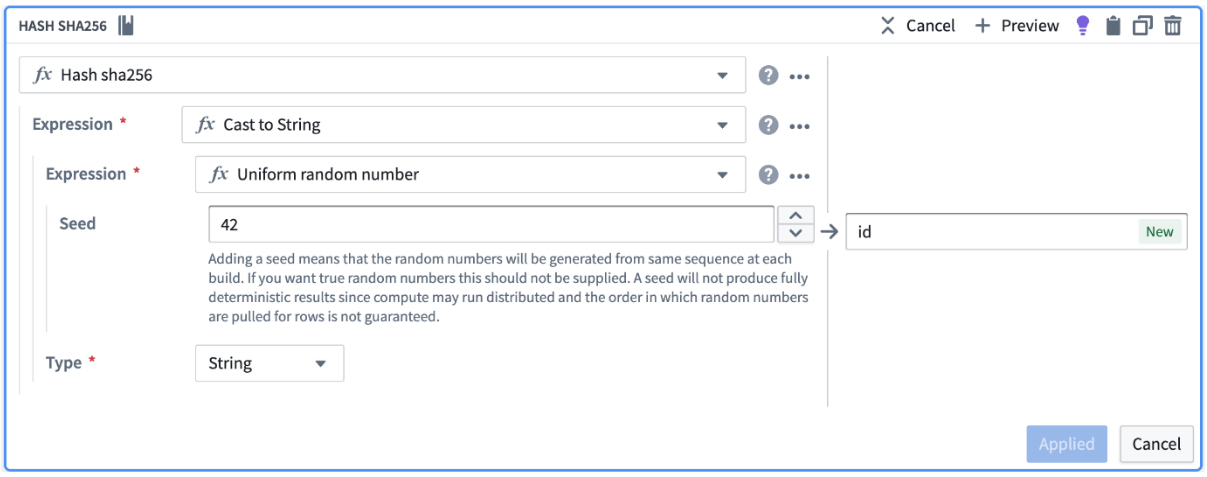
These disadvantages indicate that using monotonically increasing IDs is not the best approach for generating unique identifiers in a data pipeline application. Instead, as detailed in the previous section, we recommend using the concatenation of string columns followed by a SHA256 hash.

If a set of unique columns to hash is not available

Be aware that this will not be consistent across builds or previews. This method should be an absolute last resort if a unique set of columns can not be identified.

If you do not have a set of columns that define a unique row in your data, you can use the hash of a random number to create the ID. To create an ID in this way, follow the steps below within the Pipeline Builder transform path:

1. Create a random number using "Uniform random number".
2. Cast the column to string.
3. Use "Hash sha256" to hash that column.



Hash sha256

Expression * Cast to String

Expression * Uniform random number

Seed 42

Adding a seed means that the random numbers will be generated from same sequence at each build. If you want true random numbers this should not be supplied. A seed will not produce fully deterministic results since compute may run distributed and the order in which random numbers are pulled for rows is not guaranteed.

Type * String

id

Joins in streaming Pipeline Builder pipelines

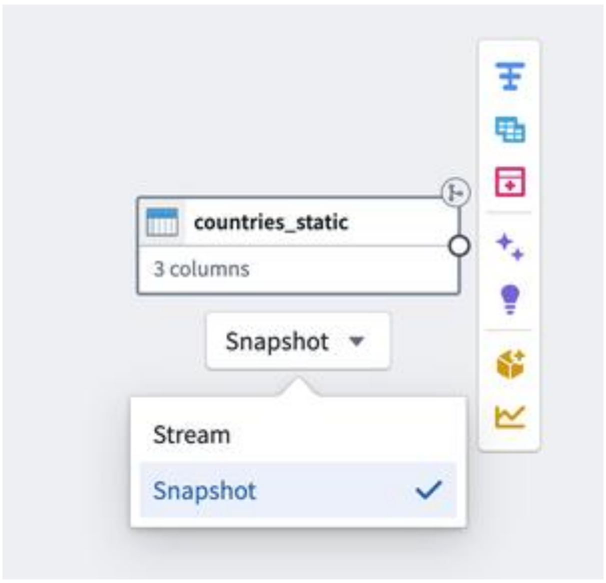
With Pipeline Builder for streaming, you can join your streams against both batch datasets and other streams. Given the low latency nature of streaming, the way joins are implemented differs from how they work in standard batch pipelines. This page explains how joins work and how best to leverage them in your pipelines.

Join streams with batch datasets

Foundry allows you to combine low latency streams with batch datasets in a manner similar to how you can join two batch datasets in a batch Pipeline Builder pipeline.

Complete the following steps to join a stream with a batch dataset in Pipeline Builder:

1. Add the stream and the batch dataset to your Pipeline Builder graph.
2. Under the batch dataset, select the dropdown menu and change the type to **Snapshot**.



3. Select the stream against which you want to join.
4. Select **Join**, and select batch dataset for the right side of the join.
5. Under **Join Type**, select **Left Lookup Join**.
6. Enter the match conditions.

Architecture

Streaming joins against batch datasets work by initially downloading the batch dataset and indexing it in the streaming cluster to allow for low latency lookups. To make the join low-latency, transforms on the batch dataset are not permitted in the same Pipeline Builder pipeline before the join with the stream.

The batch dataset is updated when new transactions are written to the dataset. When a new transaction is added to the batch dataset, a background process will download the new view of the data and convert it into a queryable format. Once that process is complete, the stream will start joining against that new view of the batch dataset.

Limitations

You cannot transform the batch dataset before joining it against a stream. If you need to transform the batch dataset, you can do so in an upstream Pipeline Builder pipeline.

Consider the following limitations for streaming joins:

- The left side of the join must be either a stream or a batch dataset with "Stream" read mode when joining against a batch dataset.

- Performance may degrade if you join against batch datasets with more than 8-10GB of data.
- The batch dataset will update at most once every five minutes if a new append transaction is detected.
- Joining against large static datasets can slow down cluster startup time.

Join streams with other streams

Foundry allows you to combine multiple low latency streams, similar to how you can join multiple batch datasets in a batch Pipeline Builder pipeline.

Complete the following steps to join two streams in Pipeline Builder:

1. Add the two streams to your Pipeline Builder graph.
2. Select **Join**, and select the two streams.
3. Under **Join Type**, select **Outer Caching Join**.
4. Enter the match conditions.
5. Specify the cache time values and units. The cache time values and units control how long data is stored in the cache we use to join the two streams.

If you want a left or right join instead of an outer join, you can filter out records that have null values downstream of the join. For a right join, filter where the right side values are null; for a left join, filter where the right side values are null.

Architecture

Since streams run indefinitely and new records are constantly flowing into both sides of the join, joins between two streams operate on caches of data from each side of the join instead of joining against the entire stream.

Joins between multiple streams are limited to operate on a cache of data to prevent unbounded state growth, which would cause the streaming cluster to eventually run out of memory and crash. By setting expiration times for the caches on the left and right side of the join, the state required to store the records for the join is bounded; this prevents the streaming cluster from running out of memory.

Data is stored on a per-key basis and distributed across task managers to allow for larger joins. This means that to join against larger streams, you can increase the memory per task manager or increase the number of task managers the cluster is running with.

Records from the left side of the stream will always be joined against the most recent record of the right side, based on the key column specified in the join. Only the most recent record for a particular key will be joined.

Limitations

Consider the following limitations when joining streams with other streams:

- A cache expiration time is required for both the left and right side of the joins to prevent unbounded state growth.

- Only the most recent value per join key is stored for each side of the join. This means the join behaves like an "outer" join.
- If a record arrives in either the left or right stream before the other side of the join has a match, a record will be emitted with null values for the other side of the join.

Use LLM node in Pipeline Builder

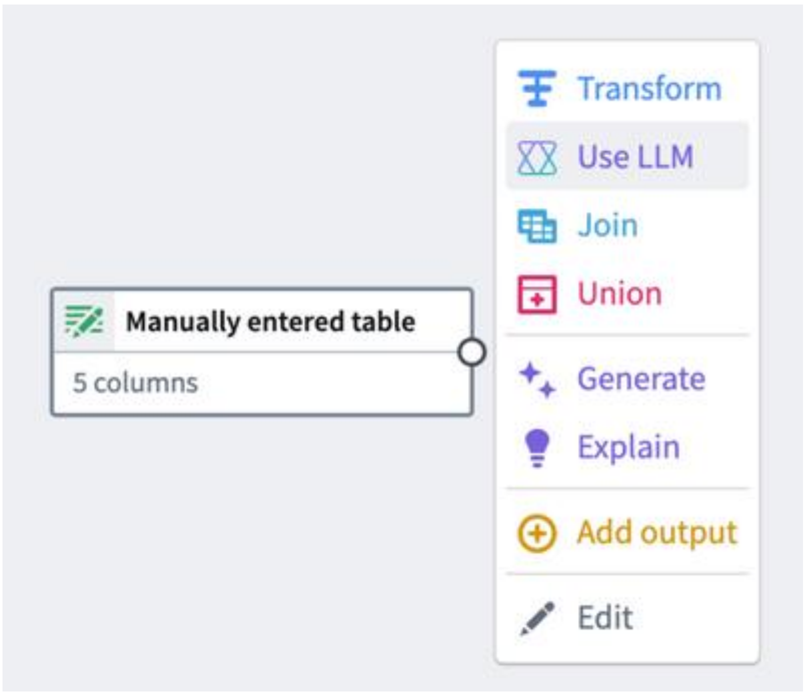
The Use LLM node in Pipeline Builder offers a convenient method for executing Large Language Models (LLMs) on your data at scale. The integration of this node within Pipeline Builder allows you to seamlessly incorporate LLM processing logic between various data transformations, simplifying the integration of LLMs into your pipeline with no coding required.

The Use LLM node includes pre-engineered prompt templates. These templates provide a beginner-friendly start to using LLMs that leverages the expertise of experienced prompt engineers. You can also run trials over a few rows of your input dataset to iterate on your prompt before running your model on an entire dataset. This preview functionality computes in seconds, speeding up the feedback loop and enhancing the overall development process.

To use, users must be granted permission for AIP capabilities for custom workflows by a platform administrator.

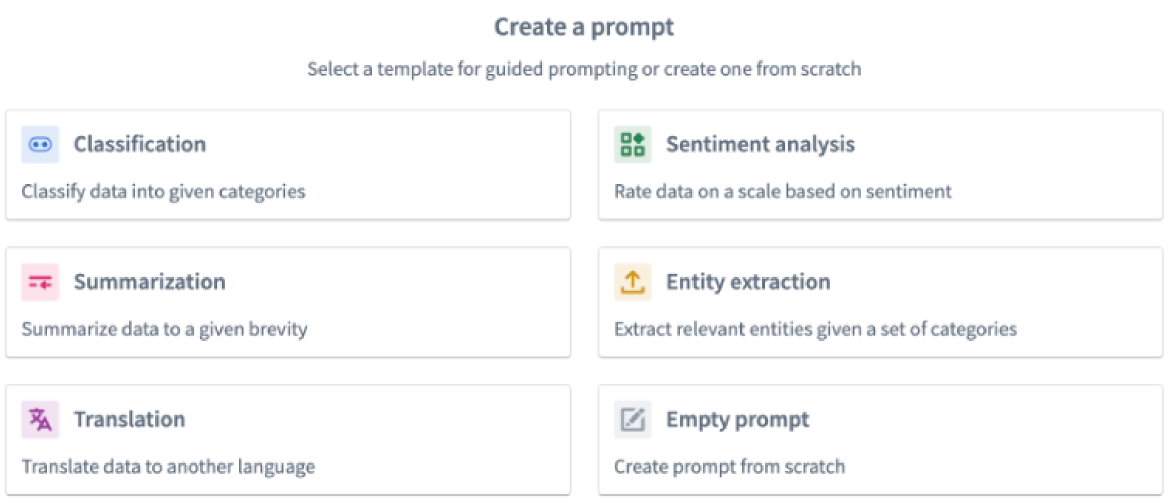
Select a dataset

To apply an LLM to a dataset, select a dataset node in your workspace and select **Use LLM**.



Select prompt

Below are different examples of the available template prompts. To create your own, select **Empty prompt**.


Create a prompt

Select a template for guided prompting or create one from scratch

Classification
Classify data into given categories

Sentiment analysis
Rate data on a scale based on sentiment

Summarization
Summarize data to a given brevity

Entity extraction
Extract relevant entities given a set of categories

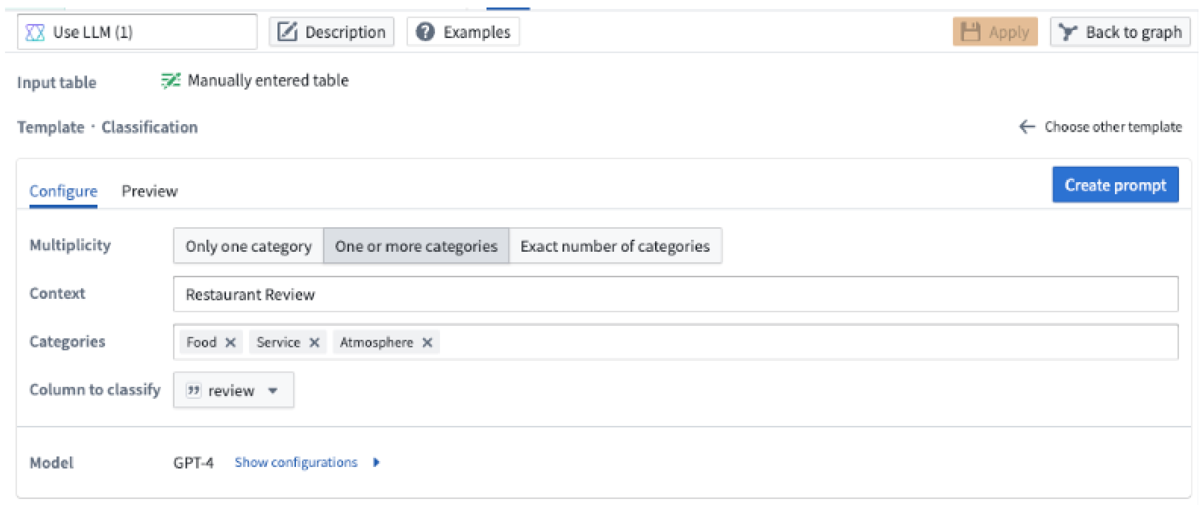
Translation
Translate data to another language

Empty prompt
Create prompt from scratch

Classification

You should use the classification prompt when you want to categorize data into different categories.

The example below demonstrates how the prompt would be filled out for our notional objective of classifying restaurant reviews into three categories: Service, Food, and Atmosphere.



Input table: Manually entered table

Template · Classification

Multiplicity: Only one category | [✓ One or more categories] | Exact number of categories

Context: Restaurant Review

Categories: Food, Service, Atmosphere

Column to classify: review

Model: GPT-4

The **Multiplicity** field allows you to choose whether you want the output column to have one category, multiple categories, or an exact number of categories. In our example, we want to include all the categories a review could fall in to, so we will choose the **One or more categories** option.

In the **Context** field, enter a description for your data. In our example, we will input Restaurant Review.

In the **Categories** field, input the distinct categories to which you want to assign your data. In our example we specify the three categories: Food, Service, and Atmosphere because we want to categorize our restaurant reviews into any of these three categories.

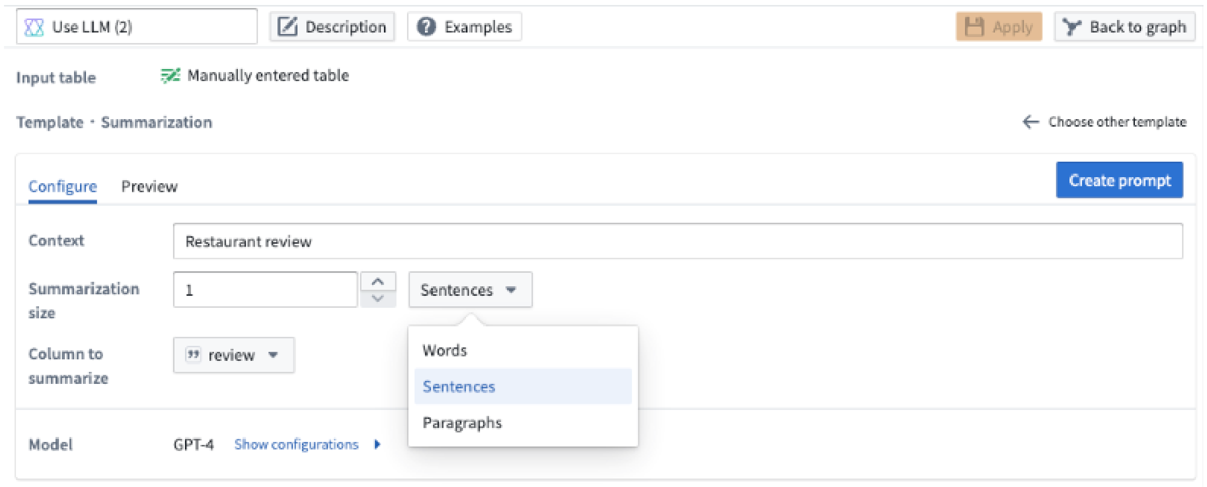
In the **Column to classify** field, choose the column that contains the data you want to classify. In our example, we choose the review column because that is the column containing our restaurant reviews.

Summarization

You can use the summarization template to summarize your data to a given length.

In this template, you can specify the length of the summarization. You can choose the number of words, sentences, or paragraphs and specify the size in the **Summarization size** field.

In our example, we want a one sentence summary of the restaurant review, so we specify 1 as the summarization size, and we choose **Sentences** from the dropdown.



Input table: Manually entered table

Template · Summarization

Context: Restaurant review

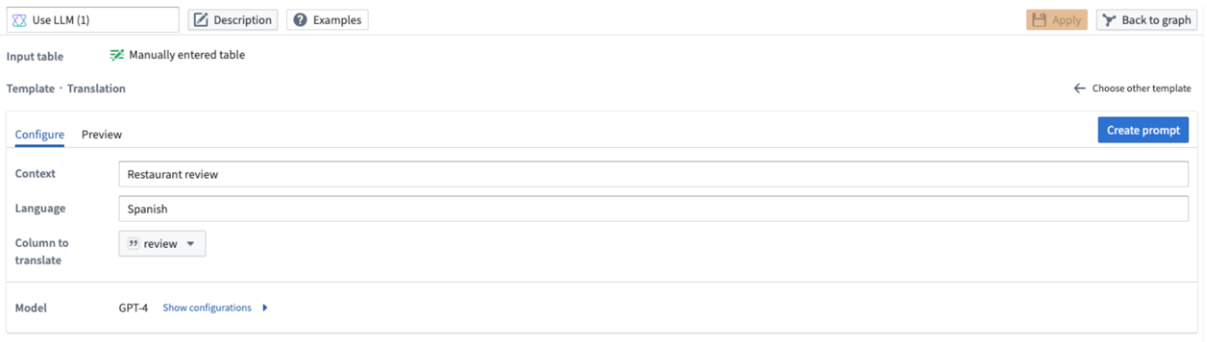
Summarization size: 1 Sentences

Column to summarize: review

Model: GPT-4

Translation

To translate your data into a different language, use the translation prompt. Specify the language you want to translate the data to in the **Language** field. In our example below, we want to translate the restaurant reviews to Spanish, so we specify Spanish under the **Language** field.



Input table: Manually entered table

Template · Translation

Context: Restaurant review

Language: Spanish

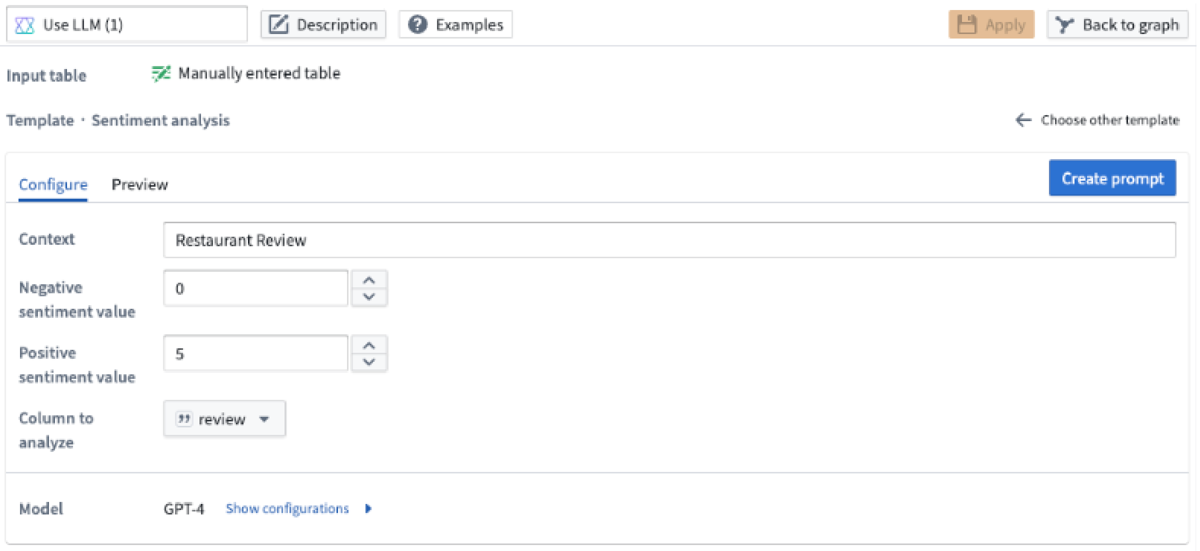
Column to translate: review

Model: GPT-4

Sentiment analysis

Use the sentiment analysis prompt when you want to assign a numeric score to your data based on its positive or negative sentiment.

In this template, you can configure the scale of the output score. For our example below, we want a number from zero to five where five denotes a review being the most positive and zero being the most negative.


Use LLM (1)

Input table: Manually entered table

Template · Sentiment analysis

Context: Restaurant Review

Negative sentiment value: 0

Positive sentiment value: 5

Column to analyze: review

Model: GPT-4

Entity extraction

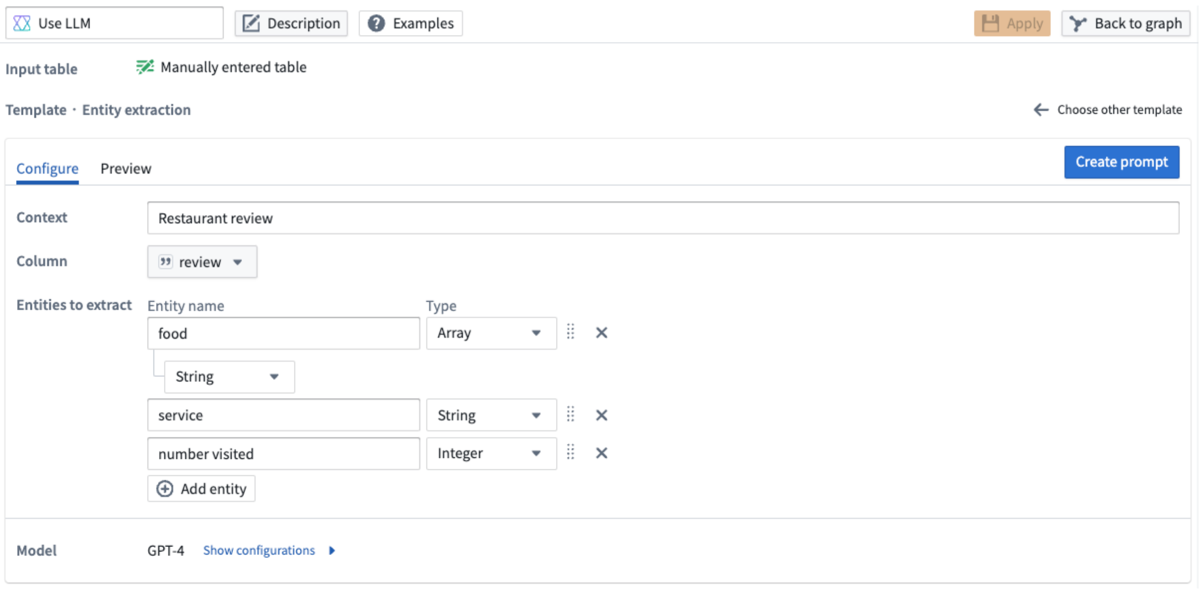
Use the entity extraction prompt when there are specific elements you want to extract from your data. In our example, we want to extract all the food, service, and times visited elements in our restaurant reviews.

In particular, we want to extract all food elements in a **String Array**, the service quality as a **String**, and an **Integer** denoting the number of times that person has visited the restaurant.

To obtain those results, we update the **Entities to extract** field. Enter food, service, and number visited under **Entity name** with the following properties:

- For food, specify an Array for the **Type** and select String as the type for that array.
- For service, select String as the type
- For number visited, select Integer.

The LLM output is now configured to conform to our specified types for this example.


Use LLM

Input table: Manually entered table

Template · Entity extraction

Context: Restaurant review

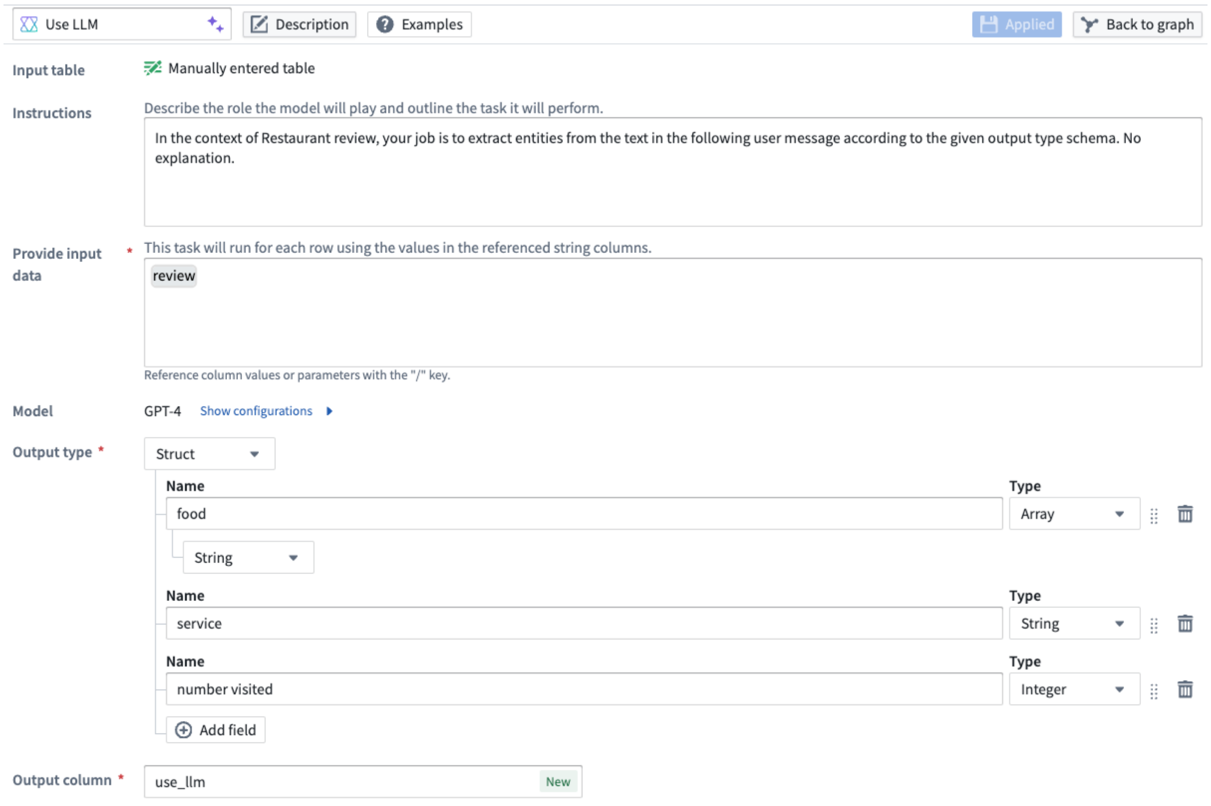
Column: review

Entities to extract:

Entity name	Type
food	Array
	String
service	String
number visited	Integer

Model: GPT-4

You can also adjust the types of the extracted entities within the struct under the **Output type** on the prompt page.



Input table: Manually entered table

Instructions: Describe the role the model will play and outline the task it will perform.

In the context of Restaurant review, your job is to extract entities from the text in the following user message according to the given output type schema. No explanation.

Provide input data *: This task will run for each row using the values in the referenced string columns.

review

Reference column values or parameters with the "/" key.

Model: GPT-4 Show configurations

Output type *: Struct

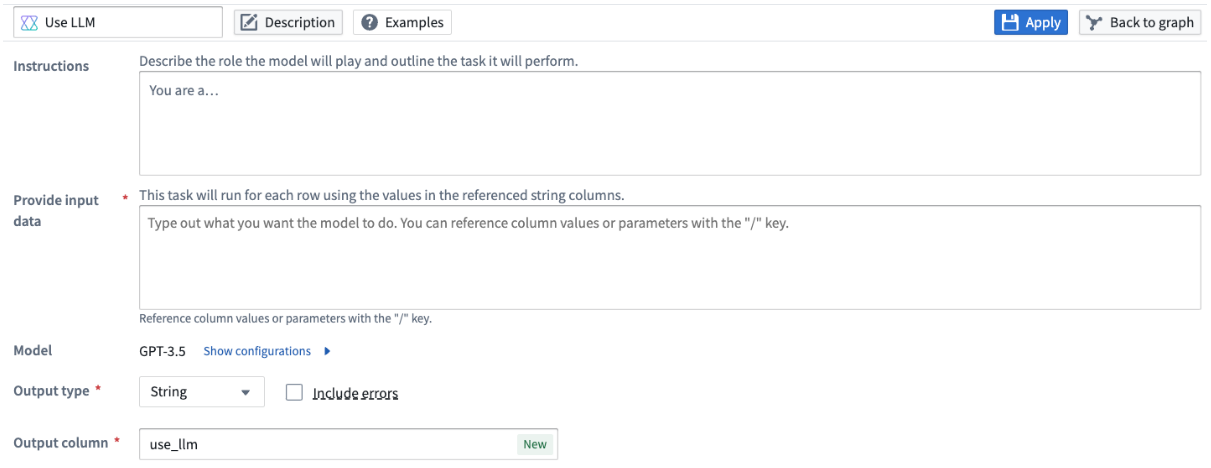
Name	Type
food	Array (String)
service	String
number visited	Integer

Add field

Output column *: use_llm New

Empty prompt

If none of the prompt templates fit your use case, you can create your own by selecting **Empty prompt**.



Instructions: Describe the role the model will play and outline the task it will perform.

You are a…

Provide input data *: This task will run for each row using the values in the referenced string columns.

Type out what you want the model to do. You can reference column values or parameters with the "/" key.

Reference column values or parameters with the "/" key.

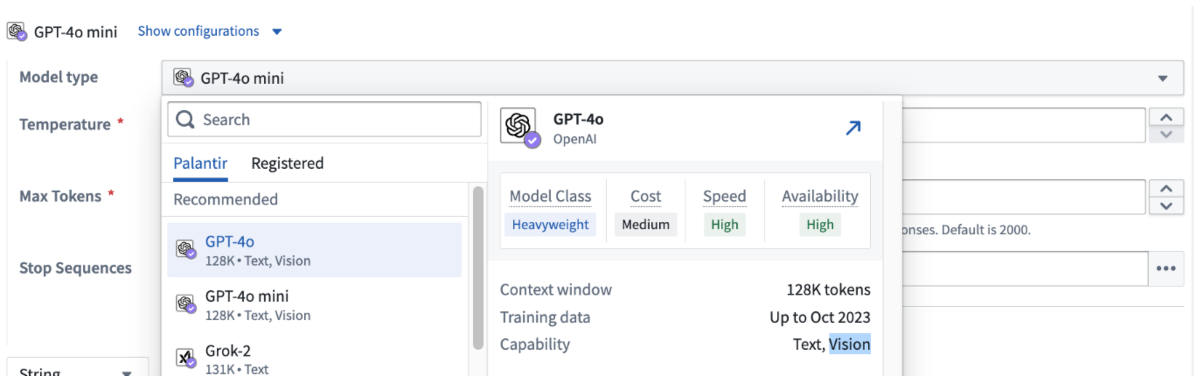
Model: GPT-3.5 Show configurations

Output type *: String ☐ Include errors

Output column *: use_llm New

Vision

Pipeline Builder also supports vision capabilities, allowing vision compatible models to analyze images and answer questions based on visual input. To check whether a model has vision capabilities, check for the **Vision** label under the **Capability** of the model in the model selector.


GPT-4o mini Show configurations

Model type GPT-4o mini

Temperature *

Max Tokens *

Stop Sequences

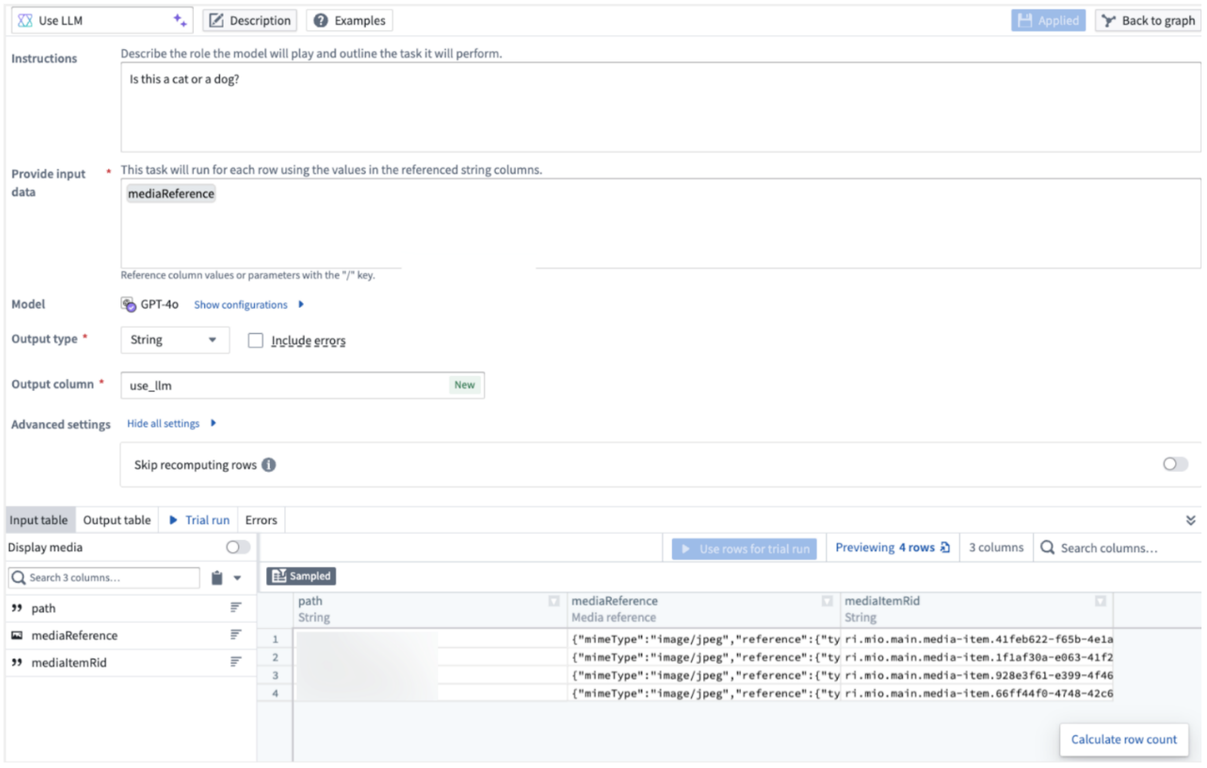
GPT-4o	128K • Text, Vision
GPT-4o mini	128K • Text, Vision
Grok-2	131K • Text

GPT-4o
OpenAI

Model Class	Cost	Speed	Availability
Heavyweight	Medium	High	High

Context window	128K tokens
Training data	Up to Oct 2023
Capability	Text, Vision

To use vision functionality, enter the media reference column in the **Provide input data** section of an empty prompt template and select the desired vision model.



Instructions

Describe the role the model will play and outline the task it will perform.

Is this a cat or a dog?

Provide input data *

This task will run for each row using the values in the referenced string columns.

mediaReference

Reference column values or parameters with the "/" key.

Model GPT-4o Show configurations

Output type * String Include errors

Output column * use_llm New

Advanced settings Hide all settings

Skip recomputing rows

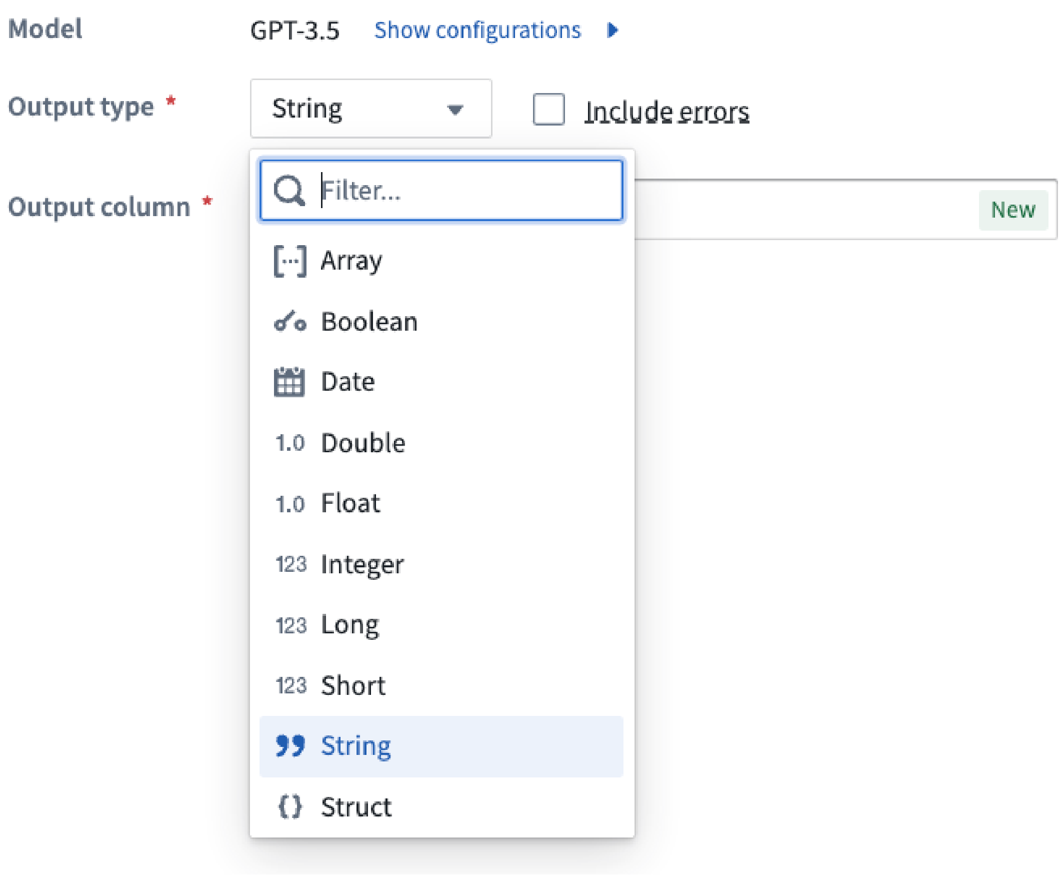
	path String	mediaReference Media reference	mediaItemRid String
1		{"mimeType":"image/jpeg","reference":{"ty	ri.mio.main.media-item.41feb622-f65b-4e1a
2		{"mimeType":"image/jpeg","reference":{"ty	ri.mio.main.media-item.1f1af30a-e063-41f2
3		{"mimeType":"image/jpeg","reference":{"ty	ri.mio.main.media-item.928e3f61-e399-4f46
4		{"mimeType":"image/jpeg","reference":{"ty	ri.mio.main.media-item.66ff44f0-4748-42c6

Currently, the vision prompt does not support media sets as a direct input. Use the **Convert Media Set to Table Rows** transform to get the mediaReference column that you can feed into the **Use LLM** node.

Optional configurations

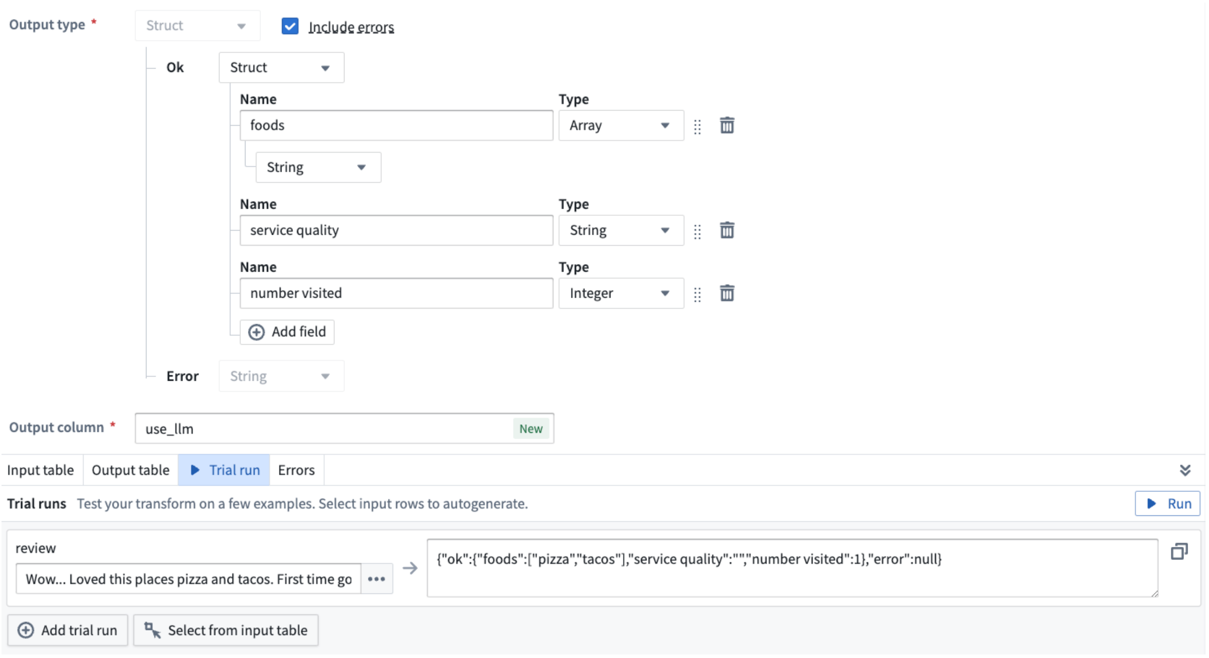
Output types

On the prompt page, you can designate the desired output type for your LLM output to conform to. Select the **Output type** option located near the bottom of the screen, then choose the preferred type from the dropdown menu.



Include errors

Also on the prompt page, you can configure your output to show the LLM errors alongside your output value. This configuration will change your output type to a struct consisting of your original output type and the error string. To include the LLM error, tick the box next to **Include errors**.



Wow... Loved this places pizza and tacos. First time go

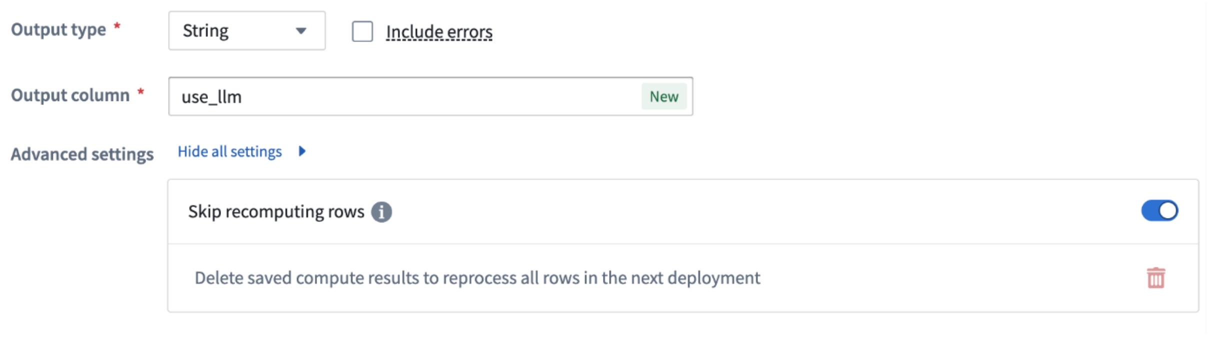
```
{"ok":{"foods":["pizza","tacos"],"service quality":"","number visited":1},"error":null}
```

To change your output back to the original output without errors, untick the **Include errors** box.

Skip computing already processed rows

This is a new feature that may not yet be available on all enrollments.

To save on compute costs and time, you can skip computing already processed rows by toggling **Skip recomputing rows**.


Output type * String ☐ Include errors

Output column * use_llm New

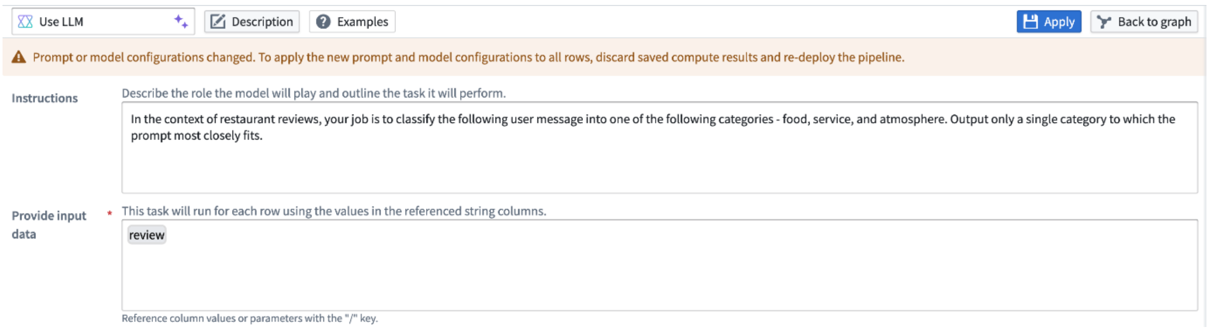
Advanced settings Hide all settings

Skip recomputing rows

Delete saved compute results to reprocess all rows in the next deployment

When **Skip recomputing rows** is enabled, rows will be compared with previously processed rows based on the columns and parameters passed into the input prompt. Matching rows with the same column and parameter values will get the cached output value without reprocessing in future deployments.

The cache can be cleared if changes that require all rows to be recomputed are made to the prompt. A warning banner will appear over the LLM view.



Prompt or model configurations changed. To apply the new prompt and model configurations to all rows, discard saved compute results and re-deploy the pipeline.

Instructions: Describe the role the model will play and outline the task it will perform.

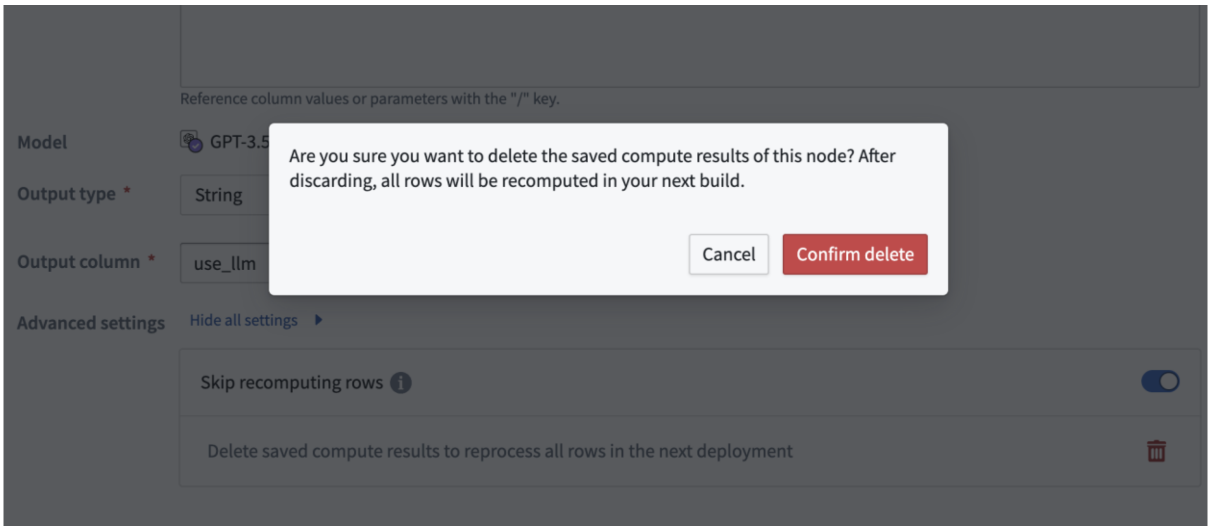
In the context of restaurant reviews, your job is to classify the following user message into one of the following categories - food, service, and atmosphere. Output only a single category to which the prompt most closely fits.

Provide input data *: This task will run for each row using the values in the referenced string columns.

review

Reference column values or parameters with the "/" key.

To clear the cache, select the red wastebasket icon. If the cache is cleared, all rows will be reprocessed in the next deployment.


Reference column values or parameters with the "/" key.

Model GPT-3.5

Output type * String

Output column * use_llm

Are you sure you want to delete the saved compute results of this node? After discarding, all rows will be recomputed in your next build.

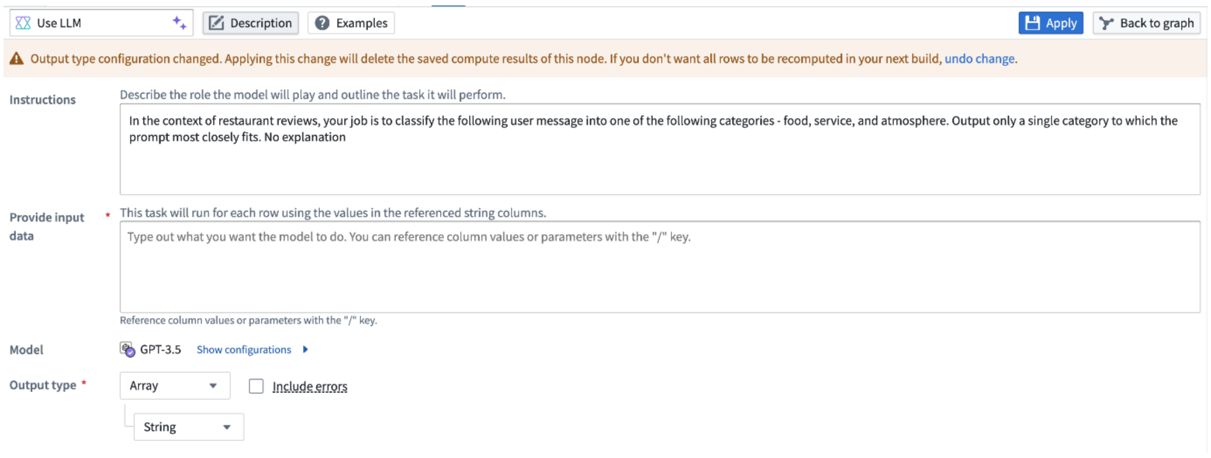
Cancel | Confirm delete

Advanced settings Hide all settings

Skip recomputing rows

Delete saved compute results to reprocess all rows in the next deployment

The cache will automatically be cleared if the output type is changed. When this happens, a warning banner will appear. If this was a mistake, you can select **undo change** in the banner.



Output type configuration changed. Applying this change will delete the saved compute results of this node. If you don't want all rows to be recomputed in your next build, undo change.

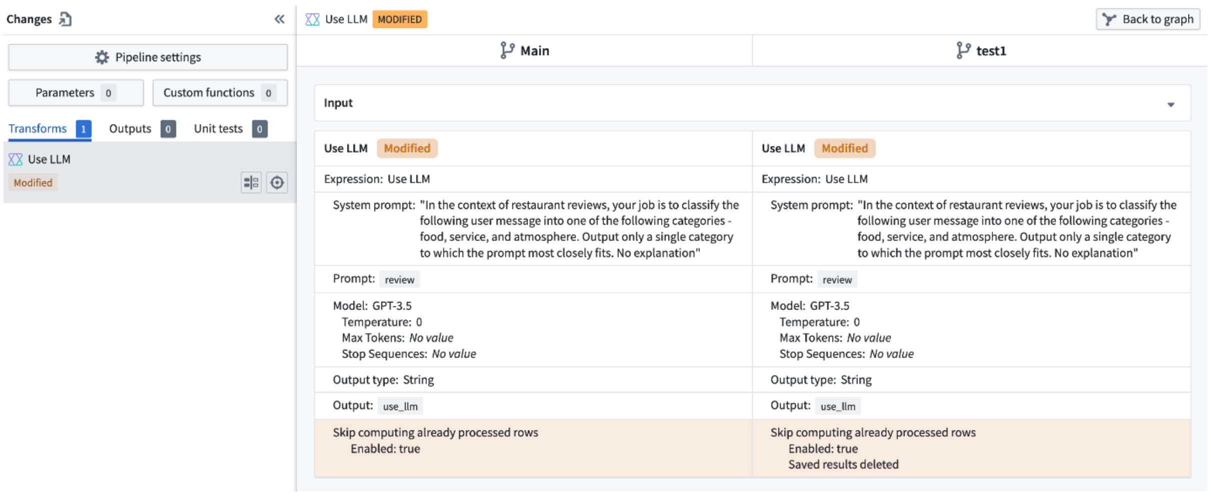
Instructions: Describe the role the model will play and outline the task it will perform.
In the context of restaurant reviews, your job is to classify the following user message into one of the following categories - food, service, and atmosphere. Output only a single category to which the prompt most closely fits. No explanation

Provide input data *: This task will run for each row using the values in the referenced string columns.
Type out what you want the model to do. You can reference column values or parameters with the "/" key.
Reference column values or parameters with the "/" key.

Model: GPT-3.5 Show configurations

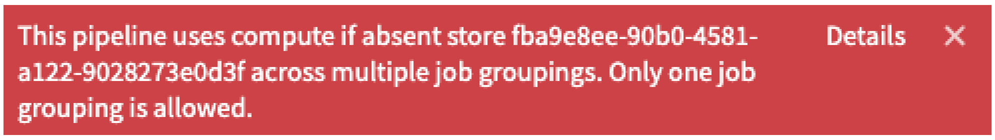
Output type *: Array — String; Include errors

Any changes to the cache's state will show up in the **Changes** page when merging a branch.



Use LLM Modified	Use LLM Modified
Expression: Use LLM	Expression: Use LLM
System prompt: "In the context of restaurant reviews, your job is to classify the following user message into one of the following categories - food, service, and atmosphere. Output only a single category to which the prompt most closely fits. No explanation"	System prompt: "In the context of restaurant reviews, your job is to classify the following user message into one of the following categories - food, service, and atmosphere. Output only a single category to which the prompt most closely fits. No explanation"
Prompt: review	Prompt: review
Model: GPT-3.5 Temperature: 0 Max Tokens: *No value* Stop Sequences: *No value*	Model: GPT-3.5 Temperature: 0 Max Tokens: *No value* Stop Sequences: *No value*
Output type: String	Output type: String
Output: use_llm	Output: use_llm
Skip computing already processed rows Enabled: true	Skip computing already processed rows Enabled: true Saved results deleted

If a use LLM node with multiple downstream outputs has **Skip recomputing rows** enabled, you must put these outputs in the same job group. Otherwise, you will get the following error when attempting to deploy:



This pipeline uses compute if absent store fba9e8ee-90b0-4581-a122-9028273e0d3f across multiple job groupings. Only one job grouping is allowed.

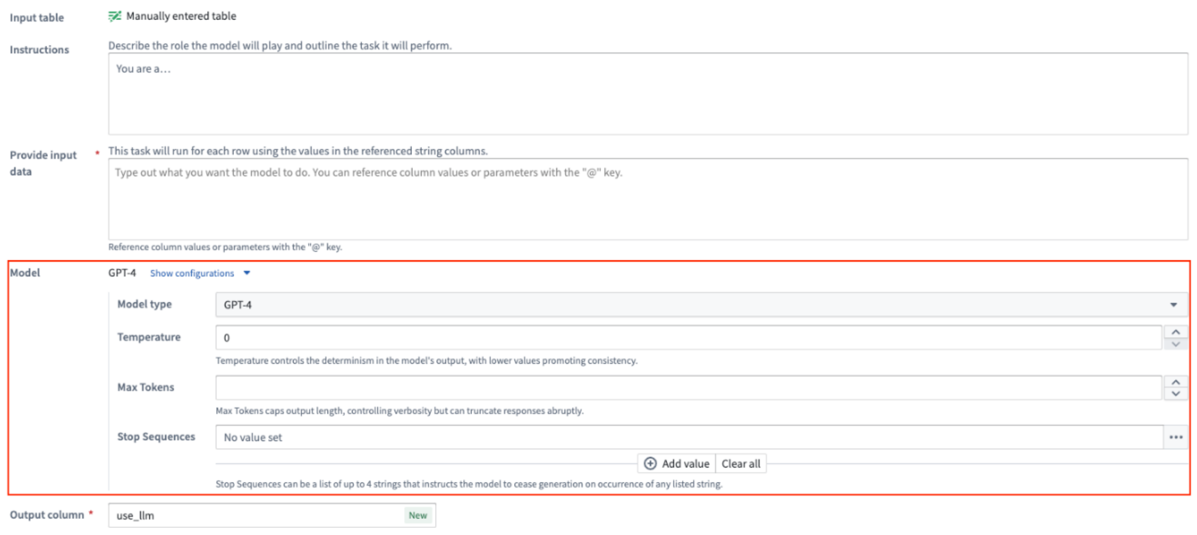
Create a new job group outside of the default job group to fix this error.

[Advanced] Show configurations for model

For every prompt, you can configure the model being used for that Use LLM node:

- **Model Type**: The model type of the GPT instance, such as 3.5 or 4. The Use LLM node also supports open source models like Mistral AI's Mixtral 8x7b.
- **Temperature**: Higher value temperatures will make the output more random while lower values will make it more focused and deterministic.

- **Max Tokens**: This will limit the number of tokens in the output. You can consider tokens as small text pieces that language models use as building blocks to understand and process written language.
- **Stop Sequence**: This will stop the LLM generation if it hits any of the stop sequence values. You can configure up to four stop sequences.



Input table: Manually entered table

Instructions: Describe the role the model will play and outline the task it will perform.
You are a…

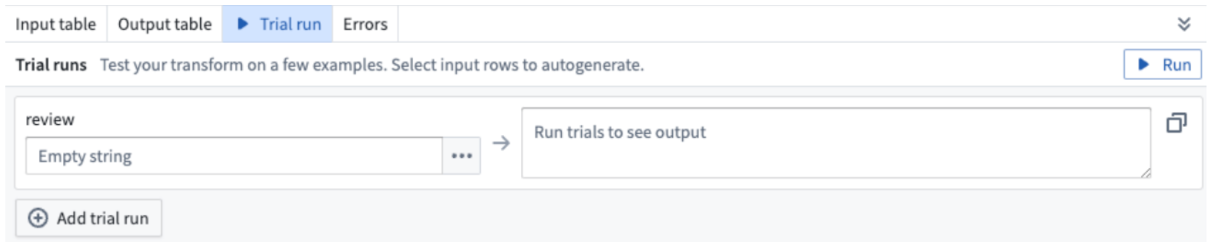
Provide input data *: This task will run for each row using the values in the referenced string columns.
Type out what you want the model to do. You can reference column values or parameters with the "@" key.
Reference column values or parameters with the "@" key.

Model: GPT-4 Show configurations
Model type: GPT-4
Temperature: 0
Temperature controls the determinism in the model's output, with lower values promoting consistency.
Max Tokens:
Max Tokens caps output length, controlling verbosity but can truncate responses abruptly.
Stop Sequences: No value set
Add value | Clear all
Stop Sequences can be a list of up to 4 strings that instructs the model to cease generation on occurrence of any listed string.

Output column *: use_llm New

Trial runs

At the bottom of each Use LLM board, you have the option to test out your specific LLM with examples. Select the **Trial run** tab and enter the value you want to test on the left hand side. Then select **Run**.



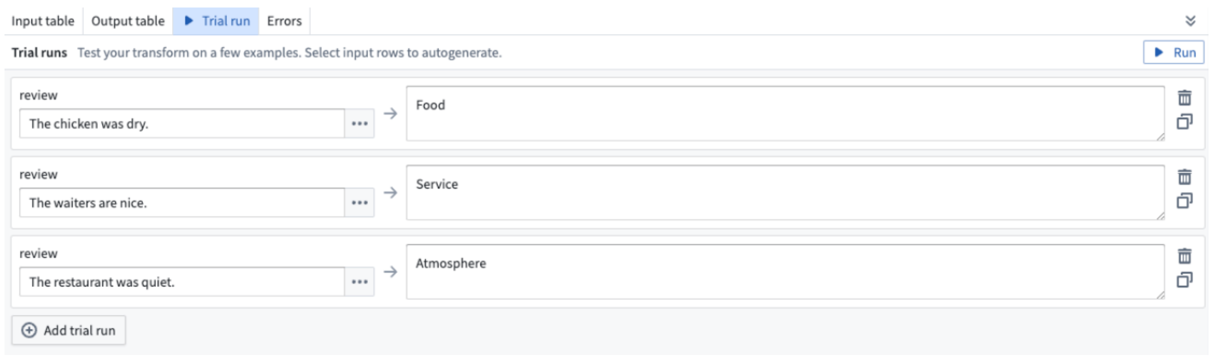
Input table | Output table | Trial run | Errors

Trial runs Test your transform on a few examples. Select input rows to autogenerate. Run

review: Empty string → Run trials to see output

Add trial run

To test out more examples, you can select **Add trial run**.



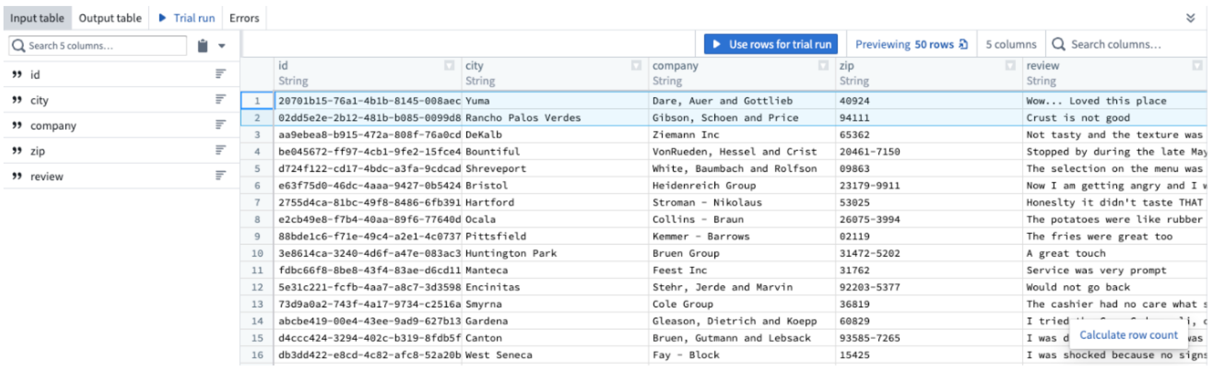
Input table | Output table | Trial run | Errors

Trial runs Test your transform on a few examples. Select input rows to autogenerate. Run

review	output
The chicken was dry.	Food
The waiters are nice.	Service
The restaurant was quiet.	Atmosphere

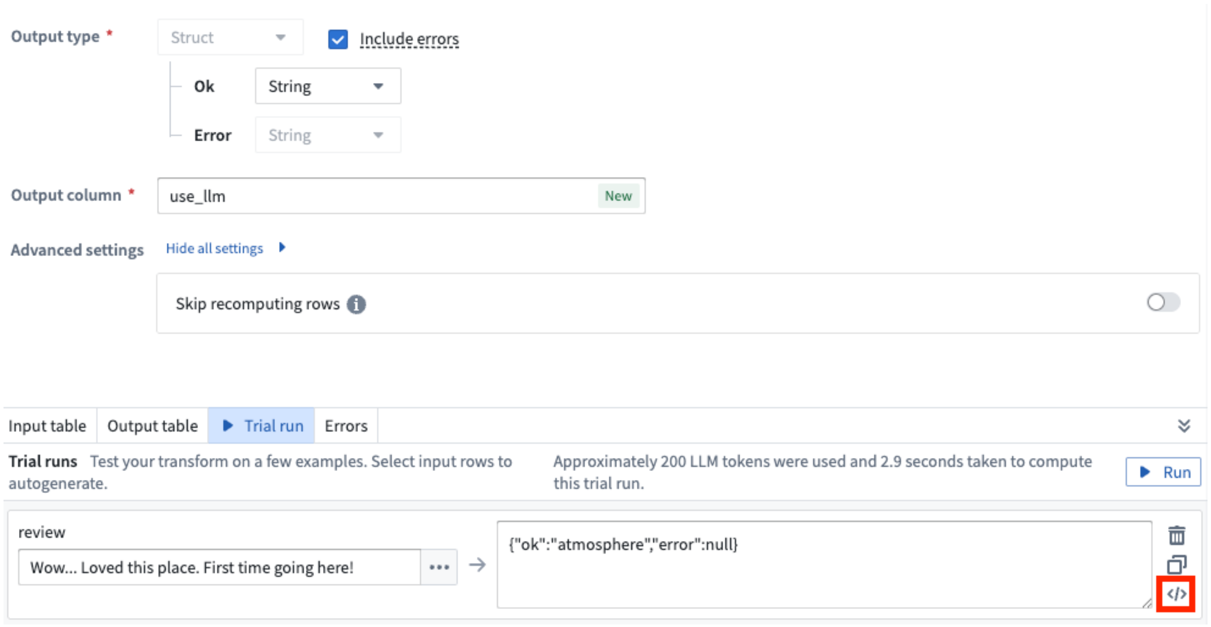
Add trial run

To add examples directly from your input data, navigate to the **Input table** tab and select the rows you want to use in your trial run. Select **Use rows for trial run**, then you will automatically be brought back to the **Trial run tab** with the rows that you selected, populated as trial runs.



	id String	city String	company String	zip String	review String
1	20701b15-76a1-4b1b-8145-008aec	Yuma	Dare, Auer and Gottlieb	40924	Wow... Loved this place
2	02dd5e2e-2b12-481b-b085-0099d8	Rancho Palos Verdes	Gibson, Schoen and Price	94111	Crust is not good
3	aa9ebea8-b915-472a-808f-76a0cd	DeKalb	Ziemann Inc	65362	Not tasty and the texture was
4	be045672-ff97-4cb1-9fe2-15fce4	Bountiful	VonRueden, Hessel and Crist	20461-7150	Stopped by during the late May
5	d724f122-cd17-4bdc-a3fa-9cdcad	Shreveport	White, Baumbach and Rolfson	09863	The selection on the menu was
6	e63f75d0-46dc-4aaa-9427-0b5424	Bristol	Heidenreich Group	23179-9911	Now I am getting angry and I
7	2755d4ca-81bc-49f8-8486-6fb391	Hartford	Stroman - Nikolaus	53025	Honeslty it didn't taste THAT
8	e2cb49e8-f7b4-40aa-89f6-77640d	Ocala	Collins - Braun	26075-3994	The potatoes were like rubber
9	88bde1c6-f71e-49c4-a2e1-4c0737	Pittsfield	Kemmer - Barrows	02119	The fries were great too
10	3e8614ca-3240-4d6f-a47e-083ac3	Huntington Park	Bruen Group	31472-5202	A great touch
11	fdbc66f8-8be8-43f4-83ae-d6cd11	Manteca	Feest Inc	31762	Service was very prompt
12	5e31c221-fcfb-4aa7-a8c7-3d3598	Encinitas	Stehr, Jerde and Marvin	92203-5377	Would not go back
13	73d9a0a2-743f-4a17-9734-c2516a	Smyrna	Cole Group	36819	The cashier had no care what
14	abcbe419-00e4-43ee-9ad9-627b13	Gardena	Gleason, Dietrich and Koepp	60829	I tried [illegible]
15	d4ccc424-3294-402c-b319-8fdb5f	Canton	Bruen, Gutmann and Lebsack	93585-7265	I was d [illegible] was
16	db3dd422-e8cd-4c82-afc8-52a20b	West Seneca	Fay - Block	15425	I was shocked because no signs

After running the trial runs, you can see the raw prompts sent to the model and the raw outputs. Simply select the </> icon to the right of the selected trial run.



Output type * Struct [✓ Include errors]

Ok: String

Error: String

Output column * use_llm

Advanced settings

Skip recomputing rows

Trial runs Test your transform on a few examples. Select input rows to autogenerate.

Approximately 200 LLM tokens were used and 2.9 seconds taken to compute this trial run.

review

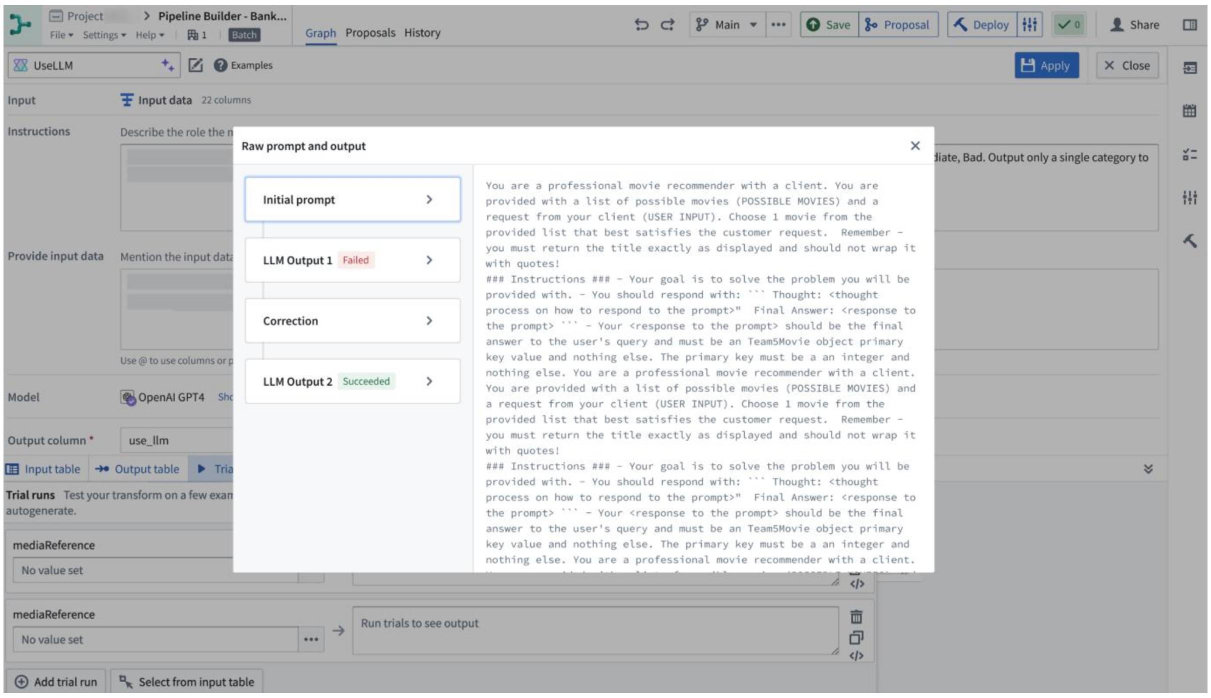
Wow... Loved this place. First time going here!

{"ok":"atmosphere","error":null}

This will open up a dialog with details including:

- Initial prompt: The exact prompt sent to the model including any modifications or additions our backend makes.
- Intermediate LLM outputs: All outputs from the LLM including any failed outputs.
- Corrections: Details on corrections made for any failures.
- Final output: The ultimate result provided by the LLM.

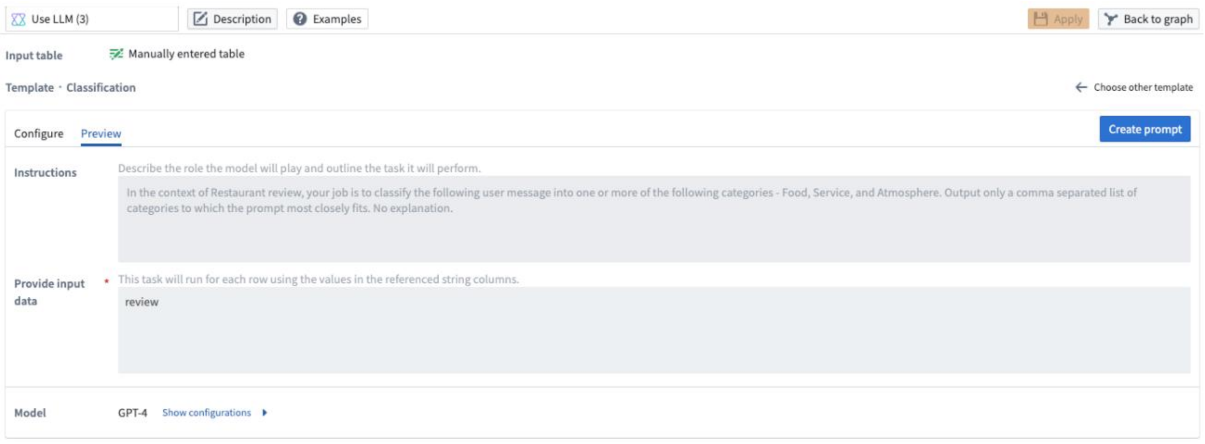
Select the respective title on the left side to see the raw text on the right side of the panel.



```
You are a professional movie recommender with a client. You are
provided with a list of possible movies (POSSIBLE MOVIES) and a
request from your client (USER INPUT). Choose 1 movie from the
provided list that best satisfies the customer request.  Remember -
you must return the title exactly as displayed and should not wrap it
with quotes!
### Instructions ### - Your goal is to solve the problem you will be
provided with. - You should respond with: ``` Thought: <thought
process on how to respond to the prompt>"  Final Answer: <response to
the prompt> ``` - Your <response to the prompt> should be the final
answer to the user's query and must be an Team5Movie object primary
key value and nothing else. The primary key must be a an integer and
nothing else. You are a professional movie recommender with a client.
You are provided with a list of possible movies (POSSIBLE MOVIES) and
a request from your client (USER INPUT). Choose 1 movie from the
provided list that best satisfies the customer request.  Remember -
you must return the title exactly as displayed and should not wrap it
with quotes!
### Instructions ### - Your goal is to solve the problem you will be
provided with. - You should respond with: ``` Thought: <thought
process on how to respond to the prompt>"  Final Answer: <response to
the prompt> ``` - Your <response to the prompt> should be the final
answer to the user's query and must be an Team5Movie object primary
key value and nothing else. The primary key must be a an integer and
nothing else. You are a professional movie recommender with a client.
```

Preview and create

If you use one of the five templates, you can preview the LLM prompt instructions before creating the prompt by selecting the **Preview** tab. You will only be able to view but not edit the instructions in the **Preview** tab. If you want to edit the template, go back to the **Configure** tab.



Input table: Manually entered table

Template · Classification

Instructions: Describe the role the model will play and outline the task it will perform.

In the context of Restaurant review, your job is to classify the following user message into one or more of the following categories - Food, Service, and Atmosphere. Output only a comma separated list of categories to which the prompt most closely fits. No explanation.

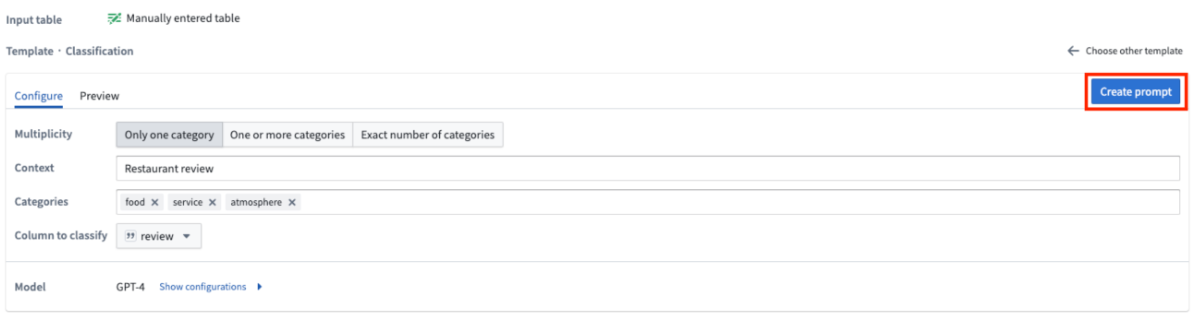
Provide input data •: This task will run for each row using the values in the referenced string columns.

review

Model: GPT-4

You should select **Create prompt** to edit the name of the new output column and preview the results in the **Output column**.

Once you select **Create prompt**, you will not be able to go back to the template for that particular board.



Input table: Manually entered table

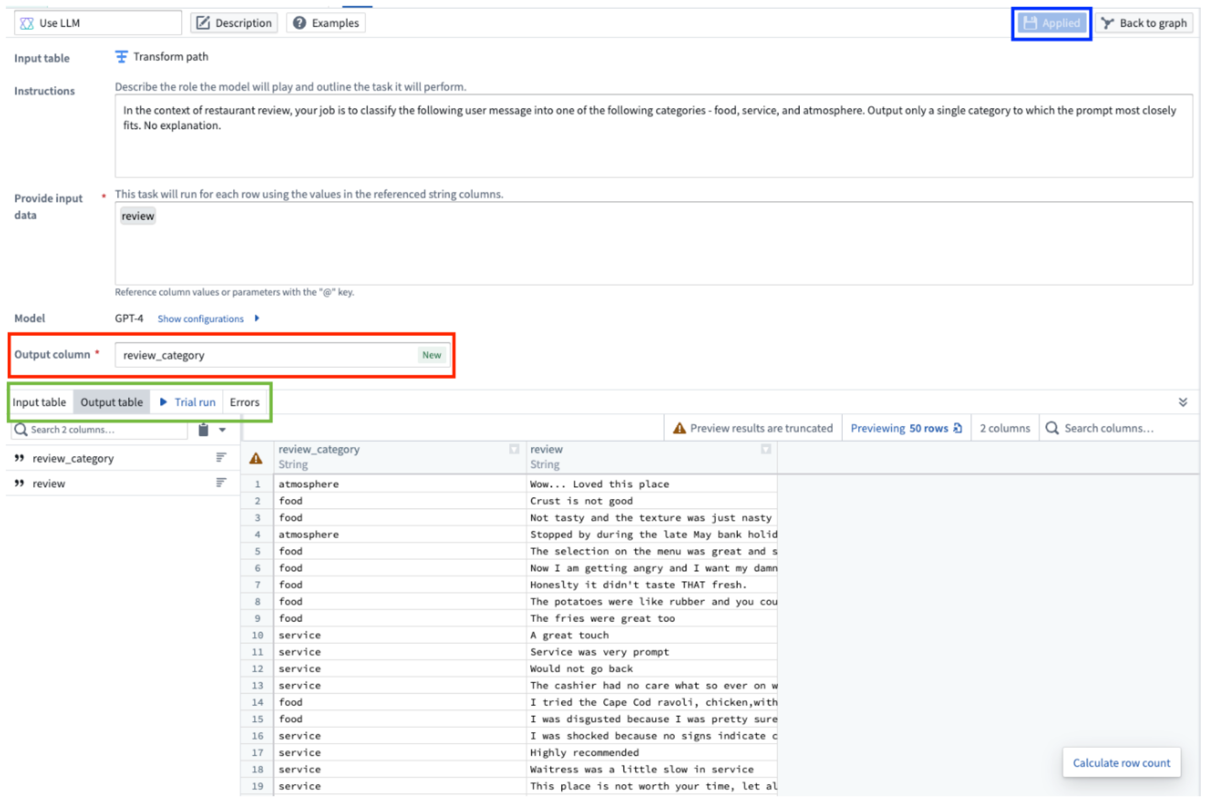
Template · Classification

Choose other template

Configure | Preview | Create prompt

Multiplicity	[✓ Only one category] One or more categories Exact number of categories
Context	Restaurant review
Categories	food, service, atmosphere
Column to classify	review
Model	GPT-4 Show configurations

To change the output column name, edit the **Output column** section. To view your changes applied to the output table preview, select **Applied**. To preview the output table, select the **Output table** tab. This preview will only show the first 50 rows.



Use LLM | Description | Examples | Applied | Back to graph

Input table: Transform path

Instructions: Describe the role the model will play and outline the task it will perform.

In the context of restaurant review, your job is to classify the following user message into one of the following categories - food, service, and atmosphere. Output only a single category to which the prompt most closely fits. No explanation.

Provide input data *: This task will run for each row using the values in the referenced string columns.

review

Reference column values or parameters with the "@" key.

Model: GPT-4 Show configurations

Output column *: review_category New

Input table | Output table | Trial run | Errors

Preview results are truncated | Previewing 50 rows | 2 columns | Search columns...

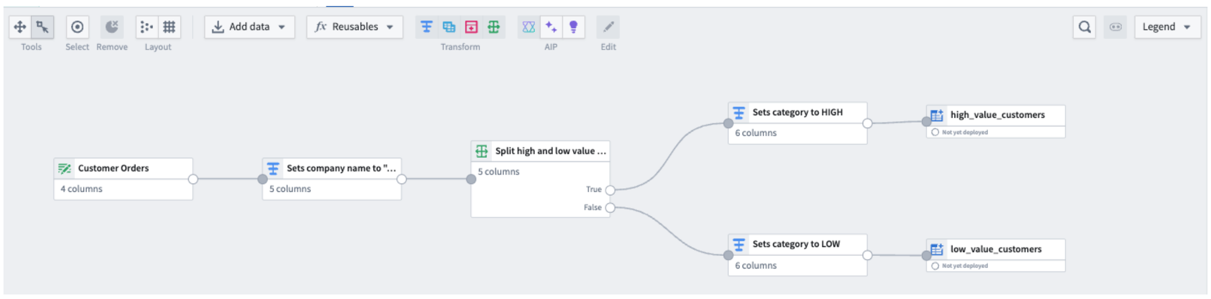
	review_category (String)	review (String)
1	atmosphere	Wow... Loved this place
2	food	Crust is not good
3	food	Not tasty and the texture was just nasty
4	atmosphere	Stopped by during the late May bank holid
5	food	The selection on the menu was great and s
6	food	Now I am getting angry and I want my damn
7	food	Honeslty it didn't taste THAT fresh.
8	food	The potatoes were like rubber and you cou
9	food	The fries were great too
10	service	A great touch
11	service	Service was very prompt
12	service	Would not go back
13	service	The cashier had no care what so ever on w
14	food	I tried the Cape Cod ravoli, chicken,with
15	food	I was disgusted because I was pretty sure
16	service	I was shocked because no signs indicate c
17	service	Highly recommended
18	service	Waitress was a little slow in service
19	service	This place is not worth your time, let al

Calculate row count

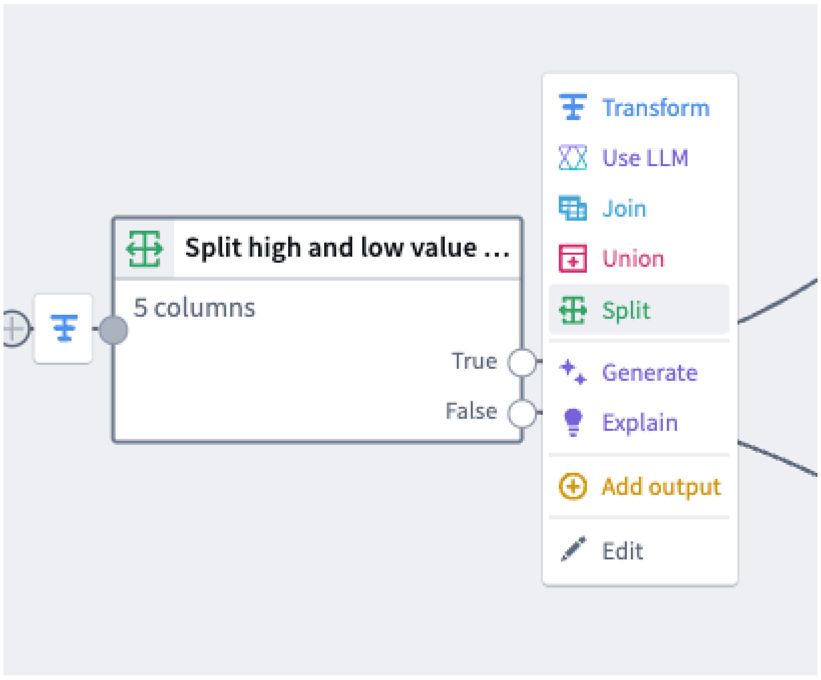
Finally, when you are finished configuring your Use LLM node, select **Apply** in the top right. This allows you to add transform logic to the output of your LLM board, and to view the preview of the first 50 rows when you select the LLM board in the main Pipeline Builder workspace.

Split transform

The Split transform is used to partition input data into two distinct outputs based on a specified condition. This transform evaluates each row of the input data against the defined condition and directs the rows to one of the two outputs accordingly.



To use the Split transform, select any dataset node in your graph and select **Split**.



Condition

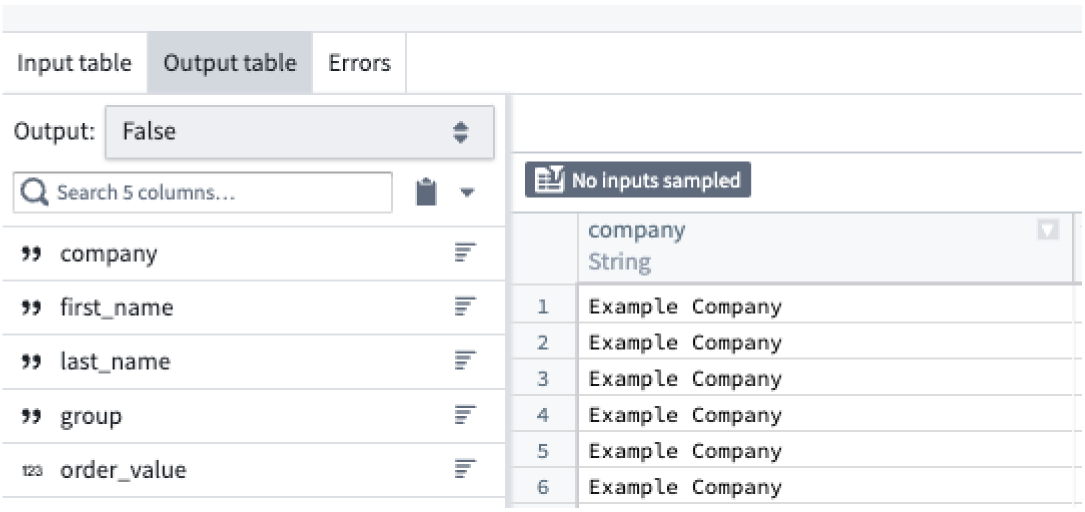
The Split transform allows you to define a condition that determines how the input data is divided. This condition is a logical expression that evaluates to either **True** or **False**.

Outputs

The **True** output will contain rows for which the condition evaluates to true. These rows are directed to the first output.

The **False** output will contain rows for which the condition evaluates to false. These rows are directed to the second output.

To preview the outputs, you can use the dropdown in the top left of the bottom panel. Select **True** to see the True output or **False** to see the False output.



Input table | Output table | Errors

Output: False

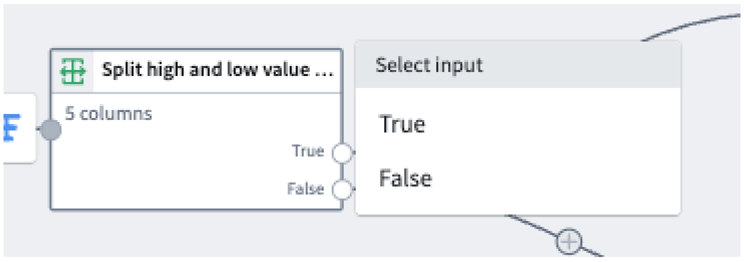
Search 5 columns...

- company
- first_name
- last_name
- group
- order_value

No inputs sampled

	company String
1	Example Company
2	Example Company
3	Example Company
4	Example Company
5	Example Company
6	Example Company

To use the outputs in downstream transforms, select the transform, and then select either the **True** or **False** output to use as your next input.

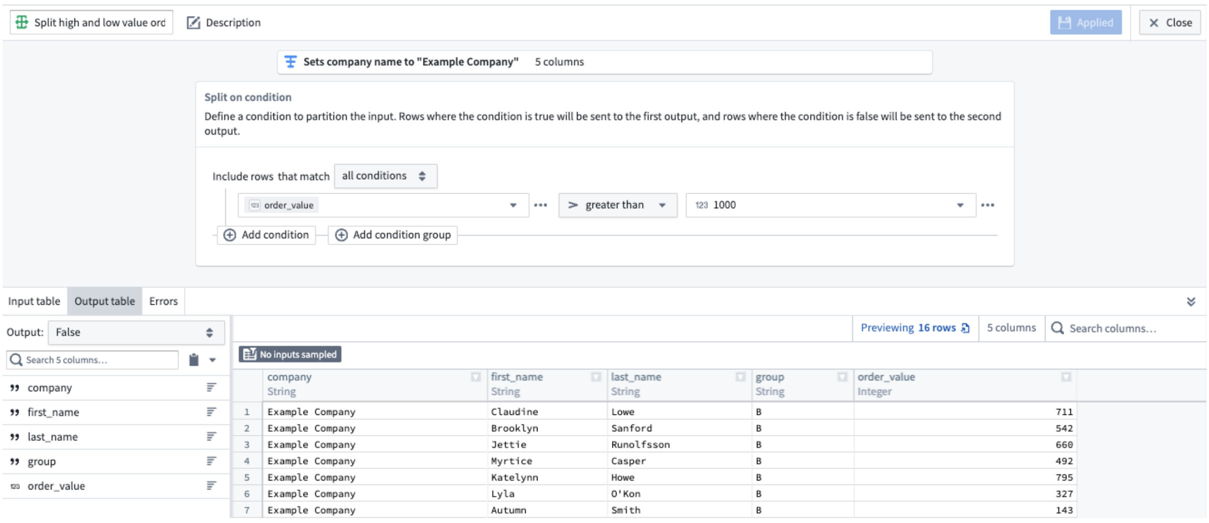


Example

Consider a dataset of customer orders. You want to separate orders into two categories: high-value and low-value orders, based on a threshold value.

In this case, the **Condition** will be order_value > 1000.

The **True** output will contain orders where order_value exceeds 1000, and the **False** output will contain orders where order_value does not exceed 1000.



Split high and low value ord | Description | Applied | Close

Sets company name to "Example Company" 5 columns

Split on condition

Define a condition to partition the input. Rows where the condition is true will be sent to the first output, and rows where the condition is false will be sent to the second output.

Include rows that match: all conditions

order_value | > greater than | 1000

Add condition | Add condition group

Input table | Output table | Errors

Output: False

Search 5 columns...

- company
- first_name
- last_name
- group
- order_value

Previewing 16 rows | 5 columns | Search columns...

No inputs sampled

	company String	first_name String	last_name String	group String	order_value Integer
1	Example Company	Claudine	Lowe	B	711
2	Example Company	Brooklyn	Sanford	B	542
3	Example Company	Jettie	Runolfsson	B	660
4	Example Company	Myrtice	Casper	B	492
5	Example Company	Katelynn	Howe	B	795
6	Example Company	Lyla	O'Kon	B	327
7	Example Company	Autumn	Smith	B	143

Frequent pattern mining

Pipeline Builder simplifies the process of frequent pattern mining by using the power of the Frequent Pattern Growth (FP-Growth) algorithm in a transform. The algorithm enables you to easily construct and manage mining workflows and discover valuable and frequent patterns in large datasets.

Frequent pattern mining is a data mining technique used to identify recurring patterns or associations within large datasets. The primary goal of frequent pattern mining is to discover relationships between items or events that occur together, more frequently than expected, by chance. These patterns, often referred to as frequent item sets, can help uncover hidden associations and dependencies in the data, facilitating better decision-making and prediction.

Frequent pattern mining can be applied in numerous ways across various domains, including market basket analysis, recommendation systems, bioinformatics, network traffic analysis, customer attribute analysis, explainable AI (XAI), and more. By identifying frequent patterns, organizations can gain valuable insights, enhance their strategies, and improve overall efficiency.

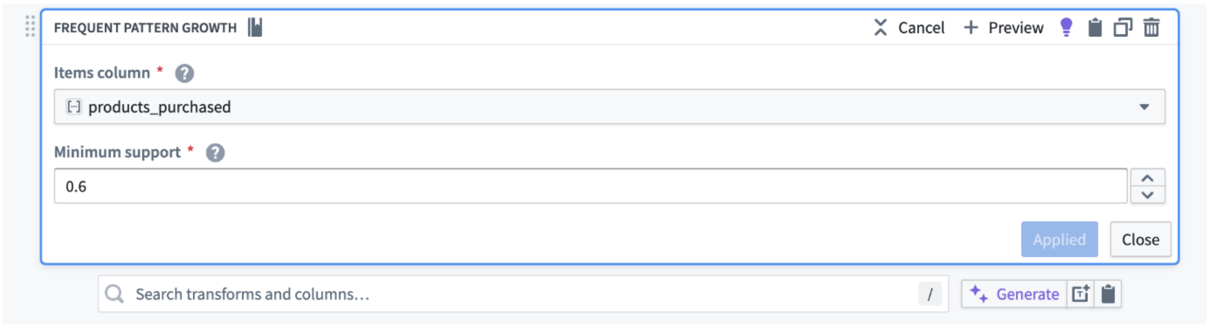
Frequent pattern mining example: Market basket analysis

A common application of frequent pattern mining in retail is called "market basket analysis". Using the Pipeline Builder Frequent Pattern Growth transform, you can identify the combinations of products that frequently occur in the same transactions.

For example, a supermarket might have a dataset of past purchases (transactions) by its customers. Each transaction contains a set of products that were bought together. Below is a simplified example dataset of such transactions:

transaction_id	products_purchased
1	[Bread, Butter, Milk]
2	[Bread, Butter]
3	[Bread, Diapers, Beer]
4	[Milk, Diapers, Beer, Butter]
5	[Bread, Butter, Diapers]

The frequent pattern growth transform takes an Items column and a Minimum support value as inputs. In this example, the products_purchased column is the items column. Since only frequent patterns will be in the output, the Minimum support is set to 0.6; the transform will only return patterns that occur in at least 60% of the transactions. The following screenshot shows how to configure the transform for this example:


FREQUENT PATTERN GROWTH

Items column *

products_purchased

Minimum support *

0.6

The output dataset of the transform is the following:

pattern	pattern_occurence	total_count
[Bread]	4	5
[Butter]	4	5
[Bread, Butter]	3	5
[Diapers]	3	5

In this case, frequent pattern mining reveals that Bread and Butter often appear together in transactions (they are a frequent item set that occurs three times in five transactions). This information could be used to drive various business strategies, such as product placements (placing bread and butter close together to increase sales) or promotions (discounts on butter when bought with bread, and vice versa).

The above is a simplified example of much larger and more complex datasets found in real use cases; the use of efficient algorithms like FP-Growth is crucial for effective frequent pattern mining.