Unnest Your Data

Contents:

You can unnest Array or Object values into separate rows or columns using the following transformations.

Flatten Array Values into Rows

Array values can be flattened into individual values in separate rows.

This section describes how to flatten the values in an Array into separate rows in your dataset.

Source:

In the following example dataset, students took the same test three times, and their scores were stored in any array in the `Scores` column.

LastName	FirstName	Scores
Adams	Allen	[81,87,83,79]
Burns	Bonnie	[98,94,92,85]
Cannon	Chris	[88,81,85,78]

Transformation:

When the data is imported, you might have to re-type the `Scores` column as an array:

Transformation Name	`Change column data type`
Parameter: Columns	`Scores`
Parameter: New type	`Array`

You can now flatten the `Scores` column data into separate rows:

Transformation Name	`Expand Array into rows`
Parameter: Column	`Scores`

Results:

LastName	FirstName	Scores
Adams	Allen	81
Adams	Allen	87

Adams	Allen	83
Adams	Allen	79
Burns	Bonnie	98
Burns	Bonnie	94
Burns	Bonnie	92
Burns	Bonnie	85
Cannon	Chris	88
Cannon	Chris	81
Cannon	Chris	85
Cannon	Chris	78

> **Tip:** You can use aggregation functions on the above data to complete values like average, minimum, and maximum scores. When these aggregation calculations are grouped by student, you can perform the calculations for each student.

Unnest Array Values into New Columns

You can also split out the individual values in an array into separate columns.

This section describes how to unnest the values in an Array into separate columns in your dataset.

Source:

In the following example dataset, students took the same test three times, and their scores were stored in any array in the `Scores` column.

LastName	FirstName	Scores
Adams	Allen	[81,87,83,79]
Burns	Bonnie	[98,94,92,85]
Cannon	Chris	[88,81,85,78]

Transformation:

When the data is imported, you might have to re-type the `Scores` column as an array:

Transformation Name	`Change column data type`
Parameter: Columns	`Scores`
Parameter: New type	`Array`

You can now unnest the `Scores` column data into separate columns:

Transformation Name	`Unnest Objects into columns`
Parameter: Column	`Scores`
Parameter: Parameter: Paths to elements	`[0]`

Parameter: Parameter: Paths to elements	`[1]`
Parameter: Parameter: Paths to elements	`[2]`
Parameter: Parameter: Paths to elements	`[3]`
Parameter: Remove elements from original	`true`
Parameter: Include original column name	`true`

In the above transformation:

- Each path is specified in a separate row.
  - The `[x]` syntax indicates that the path is the *xth* element of the array.
  - The first element of an array is referenced using `[0]`.
- You can choose to delete the element from the original or not. Deleting the element can be a helpful way of debugging your transformation. If all of the elements are gone, then the transformation is complete.
- If you include the original column name in the output column names, you have some contextual information for the outputs.

Results:

LastName	FirstName	Scores_0	Scores_1	Scores_2	Scores_3
Adams	Allen	81	87	83	79
Burns	Bonnie	98	94	92	85
Cannon	Chris	88	81	85	78

Flatten and Unnest Together

The following example illustrates how flatten and unnest can be used together to reshape your data.

This example illustrates you to use the flatten and unnest transforms.

Source:

You have the following data on student test scores. Scores on individual scores are stored in the `Scores` array, and you need to be able to track each test on a uniquely identifiable row. This example has two goals:

1. One row for each student test
2. Unique identifier for each student-score combination

LastName	FirstName	Scores
Adams	Allen	[81,87,83,79]
Burns	Bonnie	[98,94,92,85]
Cannon	Charles	[88,81,85,78]

Transformation:

When the data is imported from CSV format, you must add a `header` transform and remove the quotes from the `Scores` column:

Transformation Name	`Rename column with row(s)`
Parameter: Option	`Use row(s) as column names`
Parameter: Type	`Use a single row to name columns`
Parameter: Row number	`1`

Transformation Name	`Replace text or pattern`
Parameter: Column	`colScores`
Parameter: Find	`'\"'`
Parameter: Replace with	`''`
Parameter: Match all occurrences	`true`

Validate test date: To begin, you might want to check to see if you have the proper number of test scores for each student. You can use the following transform to calculate the difference between the expected number of elements in the `Scores` array (4) and the actual number:

Transformation Name	`New formula`
Parameter: Formula type	`Single row formula`
Parameter: Formula	`(4 - arraylen(Scores))`
Parameter: New column name	`'numMissingTests'`

When the transform is previewed, you can see in the sample dataset that all tests are included. You might or might not want to include this column in the final dataset, as you might identify missing tests when the recipe is run at scale.

Unique row identifier: The `Scores` array must be broken out into individual rows for each test. However, there is no unique identifier for the row to track individual tests. In theory, you could use the combination of `LastName-FirstName-Scores` values to do so, but if a student recorded the same score twice, your dataset has duplicate rows. In the following transform, you create a parallel array called `Tests`, which contains an index array for the number of values in the `Scores` column. Index values start at `0`:

Transformation Name	`New formula`
Parameter: Formula type	`Single row formula`
Parameter: Formula	`range(0,arraylen(Scores))`
Parameter: New column name	`'Tests'`

Also, we will want to create an identifier for the source row using the `sourcerownumber` function:

Transformation Name	`New formula`

Parameter: Formula type	`Single row formula`
Parameter: Formula	`sourcerownumber()`
Parameter: New column name	`'orderIndex'`

One row for each student test: Your data should look like the following:

LastName	FirstName	Scores	Tests	orderIndex
Adams	Allen	[81,87,83,79]	[0,1,2,3]	2
Burns	Bonnie	[98,94,92,85]	[0,1,2,3]	3
Cannon	Charles	[88,81,85,78]	[0,1,2,3]	4

Now, you want to bring together the `Tests` and `Scores` arrays into a single nested array using the `arrayzip` function:

Transformation Name	`New formula`
Parameter: Formula type	`Single row formula`
Parameter: Formula	`arrayzip([Tests,Scores])`

Your dataset has been changed:

LastName	FirstName	Scores	Tests	orderIndex	column1
Adams	Allen	[81,87,83,79]	[0,1,2,3]	2	[[0,81],[1,87],[2,83],[3,79]]
Adams	Bonnie	[98,94,92,85]	[0,1,2,3]	3	[[0,98],[1,94],[2,92],[3,85]]
Cannon	Charles	[88,81,85,78]	[0,1,2,3]	4	[[0,88],[1,81],[2,85],[3,78]]

Use the following to unpack the nested array:

Transformation Name	`Expand arrays to rows`
Parameter: Column	`column1`

Each test-score combination is now broken out into a separate row. The nested Test-Score combinations must be broken out into separate columns using the following:

Transformation Name	`Unnest Objects into columns`
Parameter: Column	`column1`
Parameter: Paths to elements	`'[0]','[1]'`

After you delete `column1`, which is no longer needed you should rename the two generated columns:

Transformation Name	`Rename columns`
Parameter: Option	`Manual rename`
Parameter: Column	`column_0`

Parameter: New column name	`'TestNum'`

Transformation Name	`Rename columns`
Parameter: Option	`Manual rename`
Parameter: Column	`column_1`
Parameter: New column name	`'TestScore'`

Unique row identifier: You can do one more step to create unique test identifiers, which identify the specific test for each student. The following uses the original row identifier `OrderIndex` as an identifier for the student and the `TestNumber` value to create the `TestId` column value:

Transformation Name	`New formula`
Parameter: Formula type	`Single row formula`
Parameter: Formula	`(orderIndex * 10) + TestNum`
Parameter: New column name	`'TestId'`

The above are integer values. To make your identifiers look prettier, you might add the following:

Transformation Name	`Merge columns`
Parameter: Columns	`'TestId00','TestId'`

Extending: You might want to generate some summary statistical information on this dataset. For example, you might be interested in calculating each student's average test score. This step requires figuring out how to properly group the test values. In this case, you cannot group by the `LastName` value, and when executed at scale, there might be collisions between first names when this recipe is run at scale. So, you might need to create a kind of primary key using the following:

Transformation Name	`Merge columns`
Parameter: Columns	`'LastName','FirstName'`
Parameter: Separator	`'-'`
Parameter: New column name	`'studentId'`

You can now use this as a grouping parameter for your calculation:

Transformation Name	`New formula`
Parameter: Formula type	`Single row formula`
Parameter: Formula	`average(TestScore)`
Parameter: Group rows by	`studentId`
Parameter: New column name	`'avg_TestScore'`

Results:

After you delete unnecessary columns and move your columns around, the dataset should look like the following:

TestId	LastName	FirstName	TestNum	TestScore	studentId	avg_TestScore
TestId0021	Adams	Allen	0	81	Adams-Allen	82.5
TestId0022	Adams	Allen	1	87	Adams-Allen	82.5
TestId0023	Adams	Allen	2	83	Adams-Allen	82.5
TestId0024	Adams	Allen	3	79	Adams-Allen	82.5
TestId0031	Adams	Bonnie	0	98	Adams-Bonnie	92.25
TestId0032	Adams	Bonnie	1	94	Adams-Bonnie	92.25
TestId0033	Adams	Bonnie	2	92	Adams-Bonnie	92.25
TestId0034	Adams	Bonnie	3	85	Adams-Bonnie	92.25
TestId0041	Cannon	Chris	0	88	Cannon-Chris	83
TestId0042	Cannon	Chris	1	81	Cannon-Chris	83
TestId0043	Cannon	Chris	2	85	Cannon-Chris	83
TestId0044	Cannon	Chris	3	78	Cannon-Chris	83

Unnest Object Values into New Columns

This example shows how you can unnest Object data into separate columns. The example contains vehicle identifiers, and the `Properties` column contains key-value pairs describing characteristics of each vehicle.

This example shows how you can unpack data nested in an Object into separate columns.

Source:

You have the following information on used cars. The `VIN` column contains vehicle identifiers, and the `Properties` column contains key-value pairs describing characteristics of each vehicle. You want to unpack this data into separate columns.

VIN	Properties
XX3 JT4522	year=2004,make=Subaru,model=Impreza,color=green,mileage=125422,cost=3199
HT4 UJ9122	year=2006,make=VW,model=Passat,color=silver,mileage=102941,cost=4599
KC2 WZ9231	year=2009,make=GMC,model=Yukon,color=black,mileage=68213,cost=12899
LL8 UH4921	year=2011,make=BMW,model=328i,color=brown,mileage=57212,cost=16999

Transformation:

Add the following transformation, which identifies all of the key values in the column as beginning with alphabetical characters.

- The `valueafter` string identifies where the corresponding value begins after the key.
- The `delimiter` string indicates the end of each key-value pair.

Transformation Name	`Convert keys/values into Objects`
Parameter: Column	`Properties`

Parameter: Key	`` `{alpha}+` ``
Parameter: Separator between key and value	`` `=` ``
Parameter: Delimiter between pair	`','`

Now that the Object of values has been created, you can use the `unnest` transform to unpack this mapped data. In the following, each key is specified, which results in separate columns headed by the named key:

> **NOTE**: Each key must be entered on a separate line in the Path to elements area.

Transformation Name	`Unnest Objects into columns`
Parameter: Column	`extractkv_Properties`
Parameter: Paths to elements	`year`
Parameter: Paths to elements	`make`
Parameter: Paths to elements	`model`
Parameter: Paths to elements	`color`
Parameter: Paths to elements	`mileage`
Parameter: Paths to elements	`cost`

Results:

When you delete the unnecessary Properties columns, the dataset now looks like the following:

VIN	year	make	model	color	mileage	cost
XX3 JT4522	2004	Subaru	Impreza	green	125422	3199
HT4 UJ9122	2006	VW	Passat	silver	102941	4599
KC2 WZ9231	2009	GMC	Yukon	black	68213	12899
LL8 UH4921	2011	BMW	328i	brown	57212	16999

Extract a Set of Values

This example shows how to extract values (for example, hashtag values) from a column and convert them into a column of arrays.

In this example, you extract one or more values from a source column and assemble them in an Array column.

Suppose you need to extract the hashtags from customer tweets to another column. In such cases, you can use the `{hashtag}` Trifacta pattern to extract all hashtag values from a customer's tweets into a new column.

Source:

The following dataset contains customer tweets across different locations.

User Name	Location	Customer tweets

James	U.K	Excited to announce that we've transitioned Wrangler from a hybrid desktop application to a completely cloud-based service! #dataprep #businessintelligence #CommitToCleanData # London
Mark	Berlin	Learnt more about the importance of identifying issues in your data—early and often #CommitToCleanData #predictivetransformations #realbusinessintelligence
Catherine	Paris	Clean data is the foundation of your analysis. Learn more about what we consider the five tenets of sound #dataprep, starting with #1a prioritizing and setting targets. #startwiththeuser #realbusinessintelligence #Paris
Dave	New York	Learn how #NewYorklife onboarded as part of their #bigdata #dataprep initiative to unlock hidden insights and make them accessible across departments.
Christy	San Francisco	How can you quickly determine the number of times a user ID appears in your data?#dataprep #pivot #aggregation#machinelearning initiatives #SFO

Transformation:

The following transformation extracts the hashtag messages from customer tweets.

Transformation Name	`Extract matches into Array`
Parameter: Column	`customer_tweets`
Parameter: Pattern matching elements in the list	`` `{hashtag}` ``
Parameter: New column name	`Hashtag tweets`

Then, the source column can be deleted.

Results:

User Name	Location	Hashtag tweets
James	U.K	["#dataprep", "#businessintelligence", "#CommitToCleanData", " # London"]
Mark	Berlin	["#CommitToCleanData", "#predictivetransformations", "#realbusinessintelligence", "0"]
Catherine	Paris	["#dataprep", "#startwiththeuser","#realbusinessintelligence", "# Paris"]
Dave	New York	["#NewYorklife", "dataprep", "bigdata", "0"]
Christy	SanFrancisco	["dataprep", "#pivot", "#aggregation", "#machinelearning"]

Nesting Data

Trifacta application provides multiple methods for transforming tabular data into nested Arrays and Objects. For more information on nesting data, see *Nest Your Data*.