



Unable to locate subtitle

AWS Glue DataBrew Developer Guide



AWS Glue DataBrew Developer Guide: ***Unable to locate subtitle***

Table of Contents

What is DataBrew?	1
Core concepts and terms	2
Projects	2
Datasets	3
Recipes	3
Jobs	3
Data lineage	3
Data profile	4
Product and service integrations	4
Setting up	7
Setting up a new AWS account	7
Setting up the AWS CLI	9
Setting up IAM permissions	10
Setting up IAM policies for DataBrew	11
Adding users and groups with DataBrew permissions	23
Adding an IAM role with DataBrew permissions	23
Setting up AWS IAM Identity Center (IAM Identity Center)	24
Login steps for an IAM Identity Center-enabled user	26
Using DataBrew in JupyterLab	26
Prerequisites	27
Configuring JupyterLab to use the extension	29
Enabling the DataBrew extension for JupyterLab	30
Getting started	33
Prerequisites	33
Step 1: Create a project	33
Step 2: Summarize the data	34
Step 3: Add more transformations	35
Step 4: Review your DataBrew resources	36
Step 5: Create a data profile	37
Step 6: Transform the dataset	38
Step 7: (Optional) Clean up	40
Datasets	41
Supported file types for data sources	41
Supported connections for data sources and outputs	43

Using datasets	47
Deleting a dataset	51
Connecting to your data	51
Using JDBC drivers to connect data	52
Supported JDBC drivers	54
Connecting to data in a text file with DataBrew	55
Connecting data in multiple files in Amazon S3	57
Schemas when using multiple files as a dataset	57
Using parameterized paths for Amazon S3	57
Data types	68
Advanced data types	69
Advanced data types	69
Validating data quality	71
Validating data quality rules	72
Acting on validation results	72
Creating a ruleset with data quality rules	73
Creating a profile job	75
Inspecting validation results for and updating data quality rules	76
Available checks	76
Projects	95
Creating a project	96
Overview of a DataBrew project session	97
Grid view	98
Schema view	100
Profile view	101
Deleting a project	104
Recipes	105
Publishing a new recipe version	106
Defining a recipe structure	106
Using conditions	110
Jobs	113
Recipe jobs	113
Example of column partitioning	118
Automating job runs with a schedule	118
Working with cron expressions for recipe jobs	119
Deleting jobs and job schedules	122

Profile jobs	122
Building a profile job configuration programmatically	124
Security	139
Data protection	139
Encryption at rest	141
Encryption in transit	143
Key management	144
Identifying and handling PII	144
DataBrew dependency on other AWS services	145
Identity and access management	145
Authenticating with identities	146
Managing access using policies	147
AWS Glue DataBrew and AWS Lake Formation	149
How AWS Glue DataBrew works with IAM	149
Identity-based policy examples	153
AWS Managed Policies for DataBrew	156
Troubleshooting	162
Logging and monitoring	163
Compliance validation	164
Resilience	165
Infrastructure security	165
Using AWS Glue DataBrew with your VPC	165
Using AWS Glue DataBrew with VPC endpoints	166
Configuration and vulnerability analysis in AWS Glue DataBrew	167
Monitoring DataBrew	168
Monitoring with CloudWatch	169
Automating with CloudWatch Events	169
Monitoring with CloudWatch Logs	171
Logging API calls with CloudTrail	172
DataBrew Information in CloudTrail	172
Understanding DataBrew Log File Entries	173
Using AWS User Notifications with AWS Glue Databrew	174
Recipe step and function reference	175
Basic column recipe steps	177
CHANGE_DATA_TYPE	178
DELETE	179

DUPLICATE	179
JSON_TO_STRUCTS	180
MOVE_AFTER	181
MOVE_BEFORE	181
MOVE_TO_END	182
MOVE_TO_INDEX	182
MOVE_TO_START	183
RENAME	183
SORT	184
TO_BOOLEAN_COLUMN	185
TO_DOUBLE_COLUMN	186
TO_NUMBER_COLUMN	187
TO_STRING_COLUMN	187
Data cleaning recipe steps	188
CAPITAL_CASE	189
FORMAT_DATE	189
LOWER_CASE	190
UPPER_CASE	191
SENTENCE_CASE	191
ADD_DOUBLE_QUOTES	192
ADD_PREFIX	192
ADD_SINGLE_QUOTES	193
ADD_SUFFIX	193
EXTRACT_BETWEEN_DELIMITERS	194
EXTRACT_BETWEEN_POSITIONS	194
EXTRACT_PATTERN	195
EXTRACT_VALUE	196
REMOVE_COMBINED	197
REPLACE_BETWEEN_DELIMITERS	201
REPLACE_BETWEEN_POSITIONS	201
REPLACE_TEXT	202
Data quality recipe steps	203
ADVANCED_DATATYPE_FILTER	204
ADVANCED_DATATYPE_FLAG	205
DELETE_DUPLICATE_ROWS	207
EXTRACT_ADVANCED_DATATYPE_DETAILS	207

FILL_WITH_AVERAGE	208
FILL_WITH_CUSTOM	209
FILL_WITH_EMPTY	209
FILL_WITH_LAST_VALID	210
FILL_WITH_MEDIAN	210
FILL_WITH_MODE	211
FILL_WITH_MOST_FREQUENT	212
FILL_WITH_NULL	212
FILL_WITH_SUM	213
FLAG_DUPLICATE_ROWS	213
FLAG_DUPLICATES_IN_COLUMN	214
GET_ADVANCED_DATATYPE	215
REMOVE_DUPLICATES	215
REMOVE_INVALID	216
REMOVE_MISSING	216
REPLACE_WITH_AVERAGE	217
REPLACE_WITH_CUSTOM	217
REPLACE_WITH_EMPTY	218
REPLACE_WITH_LAST_VALID	219
REPLACE_WITH_MEDIAN	219
REPLACE_WITH_MODE	220
REPLACE_WITH_MOST_FREQUENT	221
REPLACE_WITH_NULL	221
REPLACE_WITH_ROLLING_AVERAGE	222
REPLACE_WITH_ROLLING_SUM	223
REPLACE_WITH_SUM	223
PII recipe steps	224
CRYPTOGRAPHIC_HASH	225
DECRYPT	226
DETERMINISTIC_DECRYPT	227
DETERMINISTIC_ENCRYPT	228
ENCRYPT	230
MASK_CUSTOM	231
MASK_DATE	232
MASK_DELIMITER	233
MASK_RANGE	234

REPLACE_WITH_RANDOM_BETWEEN	235
REPLACE_WITH_RANDOM_DATE_BETWEEN	235
SHUFFLE_ROWS	236
Outlier detection and handling recipe steps	236
FLAG_OUTLIERS	237
REMOVE_OUTLIERS	239
REPLACE_OUTLIERS	241
RESCALE_OUTLIERS_WITH_Z_SCORE	243
RESCALE_OUTLIERS_WITH_SKEW	245
Column structure recipe steps	247
BOOLEAN_OPERATION	248
CASE_OPERATION	263
FLAG_COLUMN_FROM_NULL	275
FLAG_COLUMN_FROM_PATTERN	275
MERGE	276
SPLIT_COLUMN_BETWEEN_DELIMITER	277
SPLIT_COLUMN_BETWEEN_POSITIONS	277
SPLIT_COLUMN_FROM_END	278
SPLIT_COLUMN_FROM_START	279
SPLIT_COLUMN_MULTIPLE_DELIMITER	279
SPLIT_COLUMN_SINGLE_DELIMITER	280
SPLIT_COLUMN_WITH_INTERVALS	281
Column formatting recipe steps	281
NUMBER_FORMAT	281
FORMAT_PHONE_NUMBER	283
Data structure recipe steps	285
NEST_TO_ARRAY	285
NEST_TO_MAP	286
NEST_TO_STRUCT	287
UNNEST_ARRAY	287
UNNEST_MAP	288
UNNEST_STRUCT	288
UNNEST_STRUCT_N	289
GROUP_BY	290
JOIN	291
PIVOT	292

SCALE	293
TRANSPOSE	294
UNION	295
UNPIVOT	296
Data science recipe steps	297
BINARIZATION	297
BUCKETIZATION	298
CATEGORICAL_MAPPING	299
ONE_HOT_ENCODING	300
SCALE	293
SKEWNESS	302
TOKENIZATION	303
Mathematical functions	304
ABSOLUTE	305
ADD	306
CEILING	306
DEGREES	307
DIVIDE	307
EXPONENT	308
FLOOR	309
IS_EVEN	309
IS_ODD	310
LN	311
LOG	311
MOD	312
MULTIPLY	312
NEGATE	313
PI	313
POWER	314
RADIANS	315
RANDOM	315
RANDOM_BETWEEN	316
ROUND	316
SIGN	317
SQUARE_ROOT	317
SUBTRACT	318

Aggregate functions	319
ANY	319
AVERAGE	320
COUNT	320
COUNT_DISTINCT	321
KTH_LARGEST	322
KTH_LARGEST_UNIQUE	322
MAX	323
MEDIAN	323
MIN	324
MODE	324
STANDARD_DEVIATION	325
SUM	326
VARIANCE	326
Text functions	327
CHAR	328
ENDS_WITH	329
EXACT	329
FIND	330
LEFT	331
LEN	332
LOWER	333
MERGE_COLUMNS_AND_VALUES	334
PROPER	335
REMOVE_SYMBOLS	336
REMOVE_WHITESPACE	337
REPEAT_STRING	338
RIGHT	339
RIGHT_FIND	340
STARTS_WITH	341
STRING_GREATER_THAN	342
STRING_GREATER_THAN_EQUAL	343
STRING_LESS_THAN	344
STRING_LESS_THAN_EQUAL	345
SUBSTRING	346
TRIM	347

UNICODE	348
UPPER	349
Date and time functions	350
CONVERT_TIMEZONE	351
DATE	351
DATE_ADD	352
DATE_DIFF	353
DATE_FORMAT	354
DATE_TIME	355
DAY	356
HOUR	357
MILLISECOND	358
MINUTE	358
MONTH	359
MONTH_NAME	360
NOW	361
QUARTER	361
SECOND	362
TIME	363
TODAY	364
UNIX_TIME	364
UNIX_TIME_FORMAT	365
WEEK_DAY	366
WEEK_NUMBER	367
YEAR	367
Window functions	368
FILL	369
NEXT	370
PREV	370
ROLLING_AVERAGE	371
ROLLING_COUNT_A	372
ROLLING_KTH_LARGEST	372
ROLLING_KTH_LARGEST_UNIQUE	373
ROLLING_MAX	374
ROLLING_MIN	375
ROLLING_MODE	375

ROLLING_STANDARD_DEVIATION	376
ROLLING_SUM	377
ROLLING_VARIANCE	378
ROW_NUMBER	378
SESSION	379
Web functions	380
IP_TO_INT	380
INT_TO_IP	381
URL_PARAMS	382
Other functions	383
COALESCE	383
GET_ACTION_RESULT	384
GET_STEP_DATAFRAME	384
Quotas and constraints	386
Document history	387
AWS Glossary	394

What is AWS Glue DataBrew?

AWS Glue DataBrew is a visual data preparation tool that enables users to clean and normalize data without writing any code. Using DataBrew helps reduce the time it takes to prepare data for analytics and machine learning (ML) by up to 80 percent, compared to custom developed data preparation tasks, such as filtering anomalies, converting data to standard formats, and correcting invalid values.

Using DataBrew, business analysts, data scientists, and data engineers can more easily collaborate to get insights from raw data. Because DataBrew is serverless, no matter what your technical level, you can explore and transform terabytes of raw data without needing to create clusters or manage any infrastructure.

With the intuitive DataBrew interface, you can interactively discover, visualize, clean, and transform raw data. DataBrew makes smart suggestions to help you identify data quality issues that can be difficult to find and time-consuming to fix. With DataBrew preparing your data, you can use your time to act on the results and iterate more quickly. You can save transformation as steps in a recipe, which you can update or reuse later with other datasets, and deploy on a continuing basis.

The following image shows how DataBrew works at a high level.



To use DataBrew, you create a project and connect to your data. In the project workspace, you see your data displayed in a grid-like visual interface. Here, you can explore the data and see value distributions and charts to understand its profile.

To prepare the data, you can choose from more than 250 point-and-click transformations. These include removing nulls, replacing missing values, fixing schema inconsistencies, creating columns based on functions, and many more. You can also use transformations to apply natural language processing (NLP) techniques to split sentences into phrases. Immediate previews show a portion of your data before and after transformation, so you can modify your recipe before applying it to the entire dataset.

After DataBrew has run your recipe on your dataset, the output is stored in Amazon Simple Storage Service (Amazon S3). After your cleansed, prepared dataset is in Amazon S3, another of your data storage or data management systems can ingest it.

Core concepts and terms in AWS Glue DataBrew

Following, you can find an overview of the core concepts and terminology in AWS Glue DataBrew. After you read this section, see [Getting started with AWS Glue DataBrew](#), which walks you through the process of creating projects and connecting datasets and running jobs.

Topics

- [Project](#)
- [Dataset](#)
- [Recipe](#)
- [Job](#)
- [Data lineage](#)
- [Data profile](#)

Project

The interactive data preparation workspace in DataBrew is called a *project*. Using a data project, you manage a collection of related items: data, transformations, and scheduled processes. As part of creating a project, you choose or create a dataset to work on. Next, you create a *recipe*, which is a set of instructions or steps that you want DataBrew to act on. These actions transform your raw data into a form that is ready to be consumed by your data pipeline.

Dataset

Dataset simply means a set of data—rows or records that are divided into columns or fields. When you create a DataBrew project, you connect to or upload data that you want to transform or prepare. DataBrew can work with data from any source, imported from formatted files, and it connects directly to a growing list of data stores.

For DataBrew, a *dataset* is a read-only connection to your data. DataBrew collects a set of descriptive metadata to refer to the data. No actual data can be altered or stored by DataBrew. For simplicity, we use dataset to refer to both the actual dataset and the metadata DataBrew uses.

Recipe

In DataBrew, a *recipe* is a set of instructions or steps for data that you want DataBrew to act on. A recipe can contain many steps, and each step can contain many actions. You use the transformation tools on the toolbar to set up all the changes that you want to make to your data. Later, when you're ready to see the finished product of your recipe, you assign this job to DataBrew and schedule it. DataBrew stores the instructions about the data transformation, but it doesn't store any of your actual data. You can download and reuse recipes in other projects. You can also publish multiple versions of a recipe.

Job

DataBrew takes on the job of transforming your data by running the instructions that you set up when you made a recipe. The process of running these instructions is called a *job*. A job can put your data recipes into action according to a preset schedule. But you aren't confined to a schedule. You can also run jobs on demand. If you want to profile some data, you don't need a recipe. In that case, you can just set up a profile job to create a data profile.

Data lineage

DataBrew tracks your data in a visual interface to determine its origin, called a *data lineage*. This view shows you how the data flows through different entities from where it originally came. You can see its origin, other entities it was influenced by, what happened to it over time, and where it was stored.

Data profile

When you profile your data, DataBrew creates a report called a *data profile*. This summary tells you about the existing shape of your data, including the context of the content, the structure of the data, and its relationships. You can make a data profile for any dataset by running a data profile job.

Product and service integrations

Use this section to know which products and services integrate with DataBrew.

DataBrew works with the following AWS services for networking, management, and governance:

- [Amazon CloudFront](#)
- [AWS CloudFormation](#)
- [AWS CloudTrail](#)
- [Amazon CloudWatch](#)
- [AWS Step Functions](#)

DataBrew works with the following AWS data lakes and data stores:

- [AWS Lake Formation](#)
- [Amazon S3](#)

DataBrew supports the following file formats and extensions for uploading data.

Format	File extension (optional)	Extensions for compressed files (required)
Comma-separated values	.csv	.gz .snappy .lz4 .bz2

Format	File extension (optional)	Extensions for compressed files (required)
		.deflate
Microsoft Excel workbook	.xlsx	No compression support
JSON (JSON document and JSON lines)	.json, .jsonl	.gz .snappy .lz4 .bz2 .deflate
Apache ORC	.orc	.zlib .snappy
Apache Parquet	.parquet	.gz .snappy .lz4

DataBrew writes output files to Amazon S3, and supports the following file formats and extensions.

Format	File extension (uncompressed)	File extensions (compressed)
Comma-separated values	.csv	.csv.snappy , .csv.gz, .csv.lz4, csv.bz2, .csv.deflate , csv.br
Tab-separated values	.csv	.tsv.snappy , .tsv.gz, .tsv.lz4, tsv.bz2, .tsv.deflate , tsv.br

Format	File extension (uncompressed)	File extensions (compressed)
Apache Parquet	.parquet	.parquet.snappy , .parquet.gz , .parquet.lz4 , .parquet.lzo , .parquet.br
AWS Glue Parquet	Not supported	.glue.parquet.snappy
Apache Avro	.avro	.avro.snappy , .avro.gz, .avro.lz4 , .avro.bz2 , .avro.deflate , .avro.br
Apache ORC	.orc	.orc.snappy , .orc.lzo, .orc.zlib
XML	.xml	.xml.snappy , .xml.gz, .xml.lz4, .xml.bz2, .xml.deflate , .xml.br
JSON (JSON Lines format only)	.json	.json.snappy , .json.gz, .json.lz4 , json.bz2, .json.deflate , .json.br
Tableau Hyper	Not supported	Not applicable

Setting up AWS Glue DataBrew

Before you get started with AWS Glue DataBrew, you need to set up some permissions, a user, and a role. Start by doing the following steps:

1. Signing up for an AWS account as needed, and creating AWS Identity and Access Management (IAM) policies to enable users to run DataBrew:
 - Signing up for a new AWS account and adding a user. For more information, see [Setting up a new AWS account](#).
 - [Adding an IAM policy for a console user](#). A user with these permissions can access DataBrew on the AWS Management Console.
 - [Adding permissions for data resources for an IAM role](#). An IAM role with these permissions can access data on behalf of the user.

You need to be an IAM administrator to create users, roles, and policies.

2. [Adding users or groups for DataBrew](#). A user or group with the correct permissions attached can access DataBrew on the console.
3. [Adding a role with permissions to access data for DataBrew](#). A role with the correct permissions can access data on the user's behalf.

Setting up a new AWS account

If you don't have an AWS account, sign up for an AWS account and create an IAM admin user.

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://signin.aws.amazon.com/signup?request_type=register.
2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign

administrative access to a user, and use only the root user to perform [tasks that require root user access](#).

To create an administrator user, choose one of the following options.

Choose one way to manage your administrator	To	By	You can also
In IAM Identity Center (Recommended)	Use short-term credentials to access AWS. This aligns with the security best practices . For information about best practices , see Security best practices in IAM in the <i>IAM User Guide</i> .	Following the instructions in Getting started in the <i>AWS IAM Identity Center User Guide</i> .	Configure programmatic access by Configuring the AWS CLI to use AWS IAM Identity Center in the <i>AWS Command Line Interface User Guide</i> .
In IAM (Not recommended)	Use long-term credentials to access AWS.	Following the instructions in Create an IAM user for emergency access in the <i>IAM User Guide</i> .	Configure programmatic access by Manage access keys for IAM users in the <i>IAM User Guide</i> .

For more information, see the following topics in the *IAM User Guide*:

- [What is IAM?](#)
- [Getting set up with IAM](#)
- [Creating an administration user and group \(console\)](#)

Setting up the AWS CLI

If you plan to use JupyterLab or the DataBrew API, make sure to install the AWS Command Line Interface (AWS CLI). You don't need it to use the DataBrew console or perform the steps in the Getting Started exercises.

To set up the AWS CLI

1. Download and configure the AWS CLI by using the steps found following:
 - [Installing the AWS CLI](#)
 - [Configuration Basics](#)
2. Verify the setup by entering the following DataBrew command at the command prompt.

```
aws databrew help
```

If this statement returns the error "aws: error: argument command: Invalid choice" followed by a long list of services, uninstall the AWS CLI, and then reinstall. This action doesn't overwrite your existing configuration.

AWS CLI commands use the default AWS Region from your configuration, unless you set it with a parameter or a profile. You can add the `--region` parameter to each command.

If you prefer, you can add a [named profile](#) in `~/.aws/config` or `%UserProfile%/.aws/config` (on Microsoft Windows). Named profiles can also preserve other settings, as shown in the following example.

```
[profile databrew]  
aws_access_key_id = ACCESS-KEY-ID-OF-IAM-USER  
aws_secret_access_key = SECRET-ACCESS-KEY-ID-OF-IAM-USER  
region = us-east-1  
output = text
```

Setting up AWS Identity and Access Management (IAM) permissions

Before you get started, you need to set up a few things in IAM. You need to be an administrator or have help from one. However, if you have an account with administrator access, you can do these tasks yourself. You can find simple instructions for each task in this section.

Following is an overview of what you need to do:

- As part of this process, you add a user. You don't have to add a new user, you can use an existing one. You attach DataBrew permissions so that the user can open the DataBrew console.
- Create an IAM role. A role allows certain actions and gives permissions when it is used, within limits. For example, it only works for users in your AWS account. You can add more limitations later.
- Create the IAM policy or policies that you need. A policy is a list of things that a user is allowed to do. To create a policy, you open another console page and paste in the text from a file you download.

Note

What we provide here is basic setup information. We recommend that you take time to customize your permissions so they meet your security and compliance needs. If you need help, contact your administrator or AWS Support.

To add the required permissions

1. Create IAM policies to enable users to run DataBrew by doing the following:
 - [Add a custom IAM policy for a console user](#). If you don't need a custom policy, you can choose the AWS-managed policy instead. Just add it to the user in step 2. A user with these permissions can access the DataBrew service console.
 - [Add permissions for data resources](#). An IAM role with these permissions can access data on behalf of the user.

You need to be an administrator to create users, roles, and policies.

2. [Add users or groups for DataBrew](#). A user or group with the correct permissions attached can access the DataBrew console.
3. [Add a role with permissions to access data for DataBrew](#). A role with the correct permissions can access data on the user's behalf.

Setting up IAM policies for DataBrew

You use IAM policies to manage permissions. A policy makes it easier to add related permissions all at once, rather than one at a time.

We recommend that you create the policies using the same names we provide. We use the names shown following for these policies throughout the documentation. Using these names also makes it easier if you ever need to contact AWS Support. However, you can choose to change both the policy names and their contents. For more information about IAM policies, see [Create a customer managed policy](#) in the *IAM User Guide*.

After you create the policies needed to use DataBrew, you attach them to users and roles. How to do this is covered later in this section.

Topics

- [Adding an IAM policy for a console user](#)
- [Adding permissions for data resources for an IAM role](#)
- [Configuring IAM policies for DataBrew](#)

Adding an IAM policy for a console user

Setting up permissions for a user for the AWS Management Console is optional, but if you require console access, take this step first.

To set up permissions to reach DataBrew on the console, choose one of the following:

- Use the policy that's managed by AWS: `AwsGlueDataBrewFullAccessPolicy`. If you choose this option, skip to the next policy, [Adding permissions for data resources for an IAM role](#).
- Create the policy described in this section, `AwsGlueDataBrewCustomUserPolicy`. This option enables you to customize the policy with additional custom security requirements.

The following policy grants the permissions needed to run the DataBrew console. You provide those permissions by using IAM.

To define the `AwsGlueDataBrewCustomUserPolicy` IAM policy for DataBrew (console)

1. Download the JSON for the [AwsGlueDataBrewCustomUserPolicy](#) IAM policy.
2. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
3. In the navigation pane, choose **Policies**.
4. For each policy, choose **Create Policy**.
5. On the **Create Policy** screen, navigate to the **JSON** tab.
6. Copy the policy JSON statement that you downloaded. Paste it over the sample statement in the editor.
7. Verify that the policy is customized to your account, security requirements, and required AWS resources. If you need to make changes, you can make them in the editor.
8. Choose **Review policy**.

To define the `AwsGlueDataBrewCustomUserPolicy` IAM policy for DataBrew (AWS CLI)

1. Download the JSON for the [AwsGlueDataBrewCustomUserPolicy](#) IAM policy.
2. Customize the policy as described in the first step of the previous procedure.
3. Run the following command to create the policy.

```
aws iam create-policy --policy-name AwsGlueDataBrewCustomUserPolicy --policy-document file://iam-policy-AwsGlueDataBrewCustomUserPolicy.json
```

Adding permissions for data resources for an IAM role

To connect to data, AWS Glue DataBrew needs to have an IAM role that it can pass on behalf of the user. Following, you can find how to create the policy that you later attach to an IAM role.

The `AwsGlueDataBrewDataResourcePolicy` policy grants the permissions needed to connect to data using DataBrew. For any operation that accesses data in another AWS resource, such as accessing your objects in Amazon S3, DataBrew needs permission to access the resource on your behalf.

To define the `AwsGlueDataBrewDataResourcePolicy` IAM policy for DataBrew (console)

1. Download the JSON for [AwsGlueDataBrewDataResourcePolicy](#).
2. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
3. In the navigation pane, choose **Policies**.
4. For each policy, choose **Create Policy**.
5. On the **Create Policy** screen, navigate to the **JSON** tab.
6. Copy the policy JSON statement that you downloaded. Paste it over the sample statement in the editor.
7. Verify that the policy is customized to your account, security requirements, and required AWS resources. If you need to make changes, you can make them in the editor.
8. Choose **Review policy**.

To define the `AwsGlueDataBrewDataResourcePolicy` IAM policy for DataBrew (AWS CLI)

1. Download the JSON for [AwsGlueDataBrewDataResourcePolicy](#).
2. Customize the policy as described in the first step of the previous procedure.
3. Run the following command to create the policy.

```
aws iam create-policy --policy-name AwsGlueDataBrewDataResourcePolicy --policy-document file://iam-policy-AwsGlueDataBrewDataResourcePolicy.json
```

Configuring IAM policies for DataBrew

Following, you can find details and examples about IAM policies that you can use with DataBrew. Details about the basic policies are provided here. Plus, there are more examples that are not required to use DataBrew. They are additional configurations that you might use in certain situations.

Topics

- [AwsGlueDataBrewCustomUserPolicy](#)
- [AwsGlueDataBrewDataResourcePolicy](#)

- [IAM policy to use Amazon S3 objects with DataBrew](#)
- [IAM policy to use encryption with DataBrew](#)

AwsGlueDataBrewCustomUserPolicy

The `AwsGlueDataBrewCustomUserPolicy` policy grants most of the permissions required to use the DataBrew console. Some of the resources that are specified in this policy refer to services that are used by DataBrew. These include names for AWS Glue Data Catalog, Amazon S3 buckets, Amazon CloudWatch Logs, and AWS KMS resources. It is similar to the AWS-managed policy named `AwsGlueDataBrewFullAccessPolicy`.

The following table describes the permissions granted by this policy.

Action	Resource	Description
"databrew:*"	"*"	Grants permission to run all DataBrew API operations.
"glue:GetDatabases"	"*"	Allows listing of AWS Glue databases and tables.
"glue:GetPartitions"		
"glue:GetTable"		
"glue:GetTables"		
"glue:GetDataCatalogEncryptionSettings"		
"dataexchange:ListDataSets"	"*"	Allows listing of AWS Data Exchange resources in datasets.
"dataexchange:ListDataSetRevisions"		
"dataexchange:ListRevisionAssets"		
"dataexchange:CreateJob"		

Action	Resource	Description
"dataexchange:StartJob"		
"dataexchange:GetJob"		
"kms:DescribeKey"	"*"	Allows listing of AWS KMS keys to use for encryption of job output.
"kms:ListKeys"		
"kms:ListAliases"		
"kms:GenerateDataKey"	"arn:aws:kms:::key/key_ids"	Allows encrypting of job output.
"s3:ListAllMyBuckets"	"arn:aws:s3:::bucket_name/*", "arn:aws:s3:::bucket_name"	Allows listing of Amazon S3 buckets for projects, datasets, and jobs. Allows sending output files to S3.
"s3:GetBucketCORS"		
"s3:GetBucketLocation"		
"s3:GetEncryptionConfiguration"		
"sts:GetCallerIdentity"	"*"	Get information about the current caller.
"cloudtrail:LookupEvents",	"*"	Allow listing AWS CloudTrail events for datasets (data lineage).
"iam:ListRoles"	"*"	Allows listing IAM roles to use for projects and jobs.
"iam:GetRole"		

AwsGlueDataBrewDataResourcePolicy

The AwsGlueDataBrewDataResourcePolicy policy grants the permissions needed to connect to data and to configure DataBrew.

The following table describes the permissions granted by this policy.

Action	Resource	Description
"s3:GetObject"	"arn:aws:s3:::bucket_name/*", "arn:aws:s3:::bucket_name"	Allows you to preview your files.
"s3:PutObject" "s3:PutBucketCORS"	"arn:aws:s3:::bucket_name/*", "arn:aws:s3:::bucket_name"	Allows sending output files to S3.
"s3:DeleteObject"	"arn:aws:s3:::bucket_name/*", "arn:aws:s3:::bucket_name"	Allows deleting an object created by DataBrew.
"s3:ListBucket"	"arn:aws:s3:::bucket_name/*", "arn:aws:s3:::bucket_name"	Allows listing of Amazon S3 buckets from projects, datasets, and jobs.
"kms:Decrypt"	"arn:aws:kms:::key/key_ids"	Allows decrypting for encrypted datasets.
"kms:GenerateDataKey"	"arn:aws:kms:::key/key_ids"	Allows encrypting of job output.
"ec2:DescribeVpcEndpoints" "ec2:DescribeRouteTables" "ec2:DeleteNetworkInterface" "ec2:DescribeNetworkInterfaces"	"*"	Allows the setup of Amazon EC2 network items, such as virtual private clouds (VPCs), when running jobs and projects.

Action	Resource	Description
"ec2:DescribeSecurityGroups"		
"ec2:DescribeSubnets"		
"ec2:DescribeVpcAttribute"		
"ec2:CreateNetworkInterface"		
"ec2:DeleteNetworkInterface"	"*"	Allows deleting a network interface in a VPC.
"ec2:CreateTags" "ec2:DeleteTags"	"arn:aws:ec2:::network-interface/*", "arn:aws:ec2:::security-group/*"	Allows creating and deleting tags. You need these permissions if you use an AWS Glue Data Catalog with a VPC enabled. DataBrew passes data to AWS Glue to run your jobs and projects. These permissions allow tagging of Amazon EC2 resources created for development endpoints. AWS Glue tags Amazon EC2 network interfaces, security groups, and instances with <code>aws-glue-service-resource</code>.

Action	Resource	Description
"logs:CreateLogGroup" "logs:CreateLogStream" "logs:PutLogEvents"	"arn:aws:logs::log-group:/aws-glue-databrew/*"	Allows writing logs to Amazon CloudWatch Logs DataBrew writes logs to log groups whose names begin with aws-glue-databrew .
"lakeformation:GetDataAccess"	"*"	Allows access to AWS Lake Formation, provided "Glue": "GetTable" is also allowed Using Lake Formation requires further configuration in the Lake Formation console.

IAM policy to use Amazon S3 objects with DataBrew

The `AwsGlueDataBrewSpecificS3BucketPolicy` policy grants the permissions needed to access S3 on behalf of nonadministrative users.

Customize the policy as follows:

1. Replace the Amazon S3 paths in the policy so they point to the paths that you want to use. In the sample text, *BUCKET-NAME-1/SPECIFIC-OBJECT-NAME* represents a specific object or file. *BUCKET-NAME-2/* represents all objects (*) whose path name starts with BUCKET-NAME-2/. Update these to name the buckets that you are using.
2. (Optional) Use wildcards in the Amazon S3 paths to further restrict permissions. For more information, see [IAM policy elements: Variables and tags](#) in the *IAM User Guide*.

Security Best Practice: To prevent unauthorized access to Amazon S3 buckets with similar names in other AWS accounts, include the `aws:ResourceAccount` condition key in your policy. This ensures that DataBrew can only access buckets within your own AWS account, even when using wildcard resource ARNs. Add the following condition to your policy statements:

```
"Condition": {
  "StringEquals": {
    "aws:ResourceAccount": "123456789012"
  }
}
```

Replace 123456789012 with your actual AWS account ID.

As part of doing this, you might restrict permissions for the actions `s3:PutObject` and `s3:PutBucketCORS`. These actions are required only for users who create DataBrew projects, because those users need to be able to send output files to S3.

For more information and to see some examples of what you can add to an IAM policy for Amazon S3, see [Bucket Policy Examples](#) in the *Amazon S3 Developer Guide*.

The following table describes the permissions granted by this policy.

Action	Resource	Description
"s3:GetObject"	"arn:aws:s3:::bucket_name/*", "arn:aws:s3:::bucket_name"	Allows you to preview your files.
"s3:PutObject" "s3:PutBucketCORS"	"arn:aws:s3:::bucket_name/*", "arn:aws:s3:::bucket_name"	Allows sending output files to S3.
"s3:DeleteObject"	"arn:aws:s3:::bucket_name/*", "arn:aws:s3:::bucket_name"	Allows deleting an object.

To define the `AwsGlueDataBrewSpecificS3BucketPolicy` IAM policy for DataBrew (console)

1. Download the JSON for the [AwsGlueDataBrewSpecificS3BucketPolicy](#) IAM policy.
2. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
3. In the navigation pane, choose **Policies**.
4. For each policy, choose **Create Policy**.
5. On the **Create Policy** screen, navigate to the **JSON** tab.
6. Paste in the policy JSON statement over the sample statement in the editor.
7. Verify that the policy is customized to your account, security requirements, and required AWS resources. If you need to make changes, you can make them in the editor.
8. Choose **Review policy**.

To define the `AwsGlueDataBrewSpecificS3BucketPolicy` IAM policy for DataBrew (AWS CLI)

1. Download the JSON for [AwsGlueDataBrewSpecificS3BucketPolicy](#).
2. Customize the policy as described in the first step of the previous procedure.
3. Run the following command to create the policy.

```
aws iam create-policy --policy-name AwsGlueDataBrewSpecificS3BucketPolicy --policy-document file://iam-policy-AwsGlueDataBrewSpecificS3BucketPolicy.json
```

IAM policy to use encryption with DataBrew

The `AwsGlueDataBrewS3EncryptedPolicy` policy grants the permissions needed to access S3 objects encrypted with AWS Key Management Service (AWS KMS) on behalf of nonadministrative users.

Customize the policy as follows:

1. Replace the Amazon S3 paths in the policy so that they point to the paths you want to use. In the sample text, *BUCKET-NAME-1/SPECIFIC-OBJECT-NAME* represents a specific object or file. *BUCKET-NAME-2/* represents all objects (*) whose path name starts with BUCKET-NAME-2/. Update these to name the buckets you are using.

2. (Optional) Use wildcards in the Amazon S3 paths to further restrict permissions. For more information, see [IAM policy elements: Variables and tags](#).

As part of doing this, you might restrict permissions for the actions `s3:PutObject` and `s3:PutBucketCORS`. These actions are required only for users who create DataBrew projects, because those users need to be able to send output files to S3.

For more information and to see some examples of what you can add to an IAM policy for Amazon S3, see [Bucket Policy Examples](#).

3. Find the following resource ARNs in the `ToUseKms` file.

```
"arn:aws:kms:AWS-REGION-NAME:AWS-ACCOUNT-ID-WITHOUT-DASHES:key/KEY-IDS",
"arn:aws:kms:AWS-REGION-NAME:AWS-ACCOUNT-ID-WITHOUT-DASHES:key/KEY-IDS"
```

4. Change the example AWS account to your AWS account number (without hyphens).
5. Change the sample list to instead list the IAM roles you want to use. We recommend scoping your IAM policies to the smallest permissions set possible. However, you can allow your user to access all IAM roles, for example if you are using a personal learning account with sample data. To allow the list to access all IAM roles, change the sample list to one entry: `"arn:aws:iam::111122223333:role/*"`.

The following table describes the permissions granted by this policy.

Action	Resource	Description
"s3:GetObject"	"arn:aws:s3:::bucket_name/*", "arn:aws:s3:::bucket_name"	Allows you to preview your files.
"s3:ListBucket"	"arn:aws:s3:::bucket_name/*", "arn:aws:s3:::bucket_name"	Allows listing of Amazon S3 buckets from projects, datasets, and jobs.
"s3:PutObject"	"arn:aws:s3:::bucket_name/*",	Allows sending output files to S3.

Action	Resource	Description
	"arn:aws:s3:::bucket_name"	
"s3:DeleteObject"	"arn:aws:s3:::bucket_name/*", "arn:aws:s3:::bucket_name"	Allows deleting an object created by DataBrew.
"kms:Decrypt"	"arn:aws:kms:::key/key_ids"	Allows decrypting for encrypted datasets.
"kms:GenerateDataKey*"	"arn:aws:kms:::key/key_ids"	Allows encrypting of job output.

To define the `AwsGlueDataBrewS3EncryptedPolicy` IAM policy for DataBrew (console)

1. Download the JSON for the [AwsGlueDataBrewS3EncryptedPolicy](#) IAM policy.
2. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
3. In the navigation pane, choose **Policies**.
4. For each policy, choose **Create Policy**.
5. On the **Create Policy** screen, navigate to the **JSON** tab.
6. Paste in the policy JSON statement over the sample statement in the editor.
7. Verify that the policy is customized to your account, security requirements, and required AWS resources. If you need to make changes, you can make them in the editor.
8. Choose **Review policy**.

To define the `AwsGlueDataBrewS3EncryptedPolicy` IAM policy for DataBrew (AWS CLI)

1. Download the JSON for [AwsGlueDataBrewS3EncryptedPolicy](#).
2. Customize the policy as described in the first step of the previous procedure.
3. Run the following command to create the policy.

```
aws iam create-policy --policy-name AwsGlueDataBrewS3EncryptedPolicy --policy-document file://iam-policy-AwsGlueDataBrewS3EncryptedPolicy.json
```

Adding users or groups with DataBrew permissions

You assign policies to roles, and roles to users and groups to manage permissions. For more information, see [IAM Identities \(users, groups, and roles\)](#) in the *IAM User Guide*.

Before you begin, you need to have at least one user to assign permissions to.

Use the following procedure to set up DataBrew permissions for users who need to work in the DataBrew console, or run DataBrew commands in the CLI.

To set up DataBrew permissions

1. Create an access key for you user to use the AWS CLI for DataBrew, and other development tools.
2. Enable **AWS Management Console access** to allow the user to use the AWS console.
3. Create a role for DataBrew users or groups.
4. Choose the policy you are using. Do one of the following:
 - If you created `AwsGlueDataBrewCustomUserPolicy`, select it from the list.
 - To use the AWS-managed policy, select `AwsGlueDataBrewFullAccessPolicy` from the list.
5. Assign that policy to the role.
6. Set the Trust relationships for the role so that a user or group can assume the relevant role.
 - If you are not using groups, trust the user with the role.
 - If you are using groups, trust the group with the role and add the user to the group.

Adding an IAM role with data resource permissions

You use IAM roles to manage policies that are assigned together. An IAM role can be used by someone acting in a particular role, such as a DataBrew user or DataBrew itself. For more information, see [IAM Roles](#) in the *IAM User Guide*.

Use the following procedure to create an IAM role that is required for DataBrew projects to access data.

To attach the required IAM policy to a new IAM role for DataBrew

1. In the navigation pane, choose **Roles, Create Role**.
2. For **Select type of trusted entity**, choose the card labeled **AWS service**.
3. Choose **DataBrew** from the list, then choose **Next: Permissions**.
4. Enter **AwsGlueDataBrewDataResourcePolicy** in the search box (the IAM policy you created in an earlier step). Select the policy and choose **Next: Tags**.
5. Choose **Next: Review**.
6. For **Role name**, enter **AwsGlueDataBrewDataAccessRole**, and choose **Create role**.

Setting up AWS IAM Identity Center (IAM Identity Center)

Using AWS IAM Identity Center (IAM Identity Center), your users can sign in to DataBrew with a simple URL, without signing in to the AWS Management Console and without needing an AWS account.

To set up IAM Identity Center

1. Open the [AWS Organizations console](#), and create an organization if you don't already have one. All features are enabled by default for this organization.

For more information, see [AWS IAM Identity Center Prerequisites](#) and [Creating and managing an organization](#).

2. Open the [AWS IAM Identity Center console](#)
3. Choose your identity source.

By default, you get an IAM Identity Center store for quick and easy user management. Optionally, you can connect an external identity provider instead, or connect an AWS Managed Microsoft AD directory with your on-premises Active Directory. In this guide, we use the default IAM Identity Center store.

For more information, see [Choose your identity source](#) in the *AWS IAM Identity Center User Guide*.

4. Create a permission set for DataBrew access:

- a. In the IAM Identity Center navigation pane, choose **AWS accounts**, and then choose **Permission sets**.
- b. On the **Create permission set** page, choose **Create a custom permission set**.
- c. For **Relay state**, enter `https://console.aws.amazon.com/databrew/home?region=us-east-1#landing`.

Entering this enables your users to go directly to DataBrew.

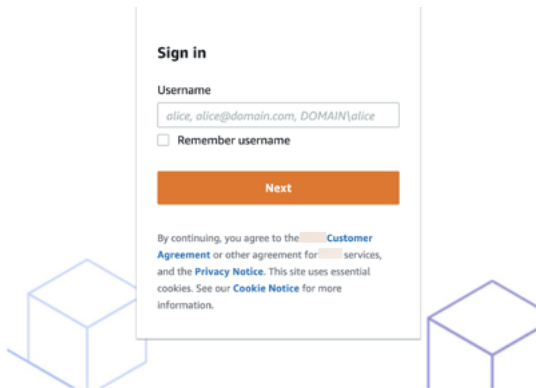
- d. Choose **Attach AWS managed policies**, search for DataBrew, and choose **AwsGlueDataBrewFullAccessPolicy**. Choosing this gives your users all the permissions that they need for DataBrew. You can find more details in [Adding an IAM policy for a console user](#).
 - e. (Optional) Choose **Create a custom permissions policy** and customize the permissions for your users.
5. In the IAM Identity Center navigation pane, choose **Groups**, and choose **Create group**. Enter the group name and choose **Create**.
 6. Add a user to IAM Identity Center store:
 - a. In the IAM Identity Center navigation pane, choose **Users**.
 - b. On the **Add user** screen, enter the required information and choose **Send an email to the user with password setup instructions**. The user should get an email about the next setup steps.
 - c. Choose **Next: Groups**, choose the group that you want, and choose **Add user**.

Users should receive an email inviting them to use SSO. In this email, they need to choose **Accept invitation** and set the password. They can also find the portal URL in the email. They can use this URL to access DataBrew.

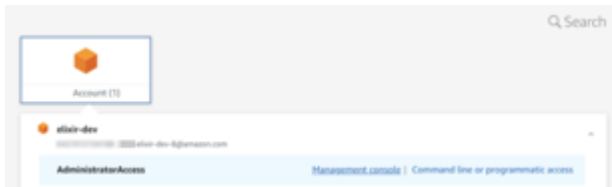
7. Assign each user to an account:
 - a. Open the [IAM Identity Center console](#), and in the navigation pane, choose **AWS accounts**.
 - b. Choose **AWS organization** and choose an AWS account.
 - c. On the **Assign Users** screen, choose the **Groups** tab and choose the group that you want.
 - d. Choose **Next: Permission sets**.
 - e. Choose the permission set for DataBrew, and choose **Finish**.

Login steps for an IAM Identity Center-enabled user

1. Sign into AWS using an IAM Identity Center-enabled account.



2. Click on **AWS Account** identity



3. Click **Management console** for one-click re-direction to the DataBrew console.

Using DataBrew as an extension in JupyterLab

Warning

AWS Glue DataBrew JupyterLab extension support is ending on December 31, 2024 as JupyterLab 3 will reach end of support. For more information, see [JupyterLab 3 end of maintenance](#).

If you prefer to prepare data in a Jupyter Notebook environment, you can use all the capabilities of AWS Glue DataBrew in JupyterLab.

JupyterLab is a web-based interactive development environment for Jupyter Notebook. In the local JupyterLab webpage, you can add sections for a terminal, a SQL session, Python, and more. After installing the AWS Glue DataBrew extension, you can add a section for the DataBrew console. It runs with any existing notebooks or other extensions that you already have, directly from the JupyterLab environment.

Topics

- [Prerequisites](#)
- [Configuring JupyterLab to use the extension](#)
- [Enabling the DataBrew extension for JupyterLab](#)

Prerequisites

Before you begin, set up the following items:

- An AWS account – If you don't have one yet, start with [Setting up a new AWS account](#).
- An AWS Identity and Access Management (IAM) user with access to the permissions needed for DataBrew – For more information, see [Adding users or groups with DataBrew permissions](#).
- An IAM role to use in DataBrew operations – You can use the default, if `AwsGlueDataBrewDataAccessRole` is configured. To set up additional IAM roles, see [Adding an IAM role with data resource permissions](#).
- A JupyterLab installation (version 2.2.6 or greater) – For more information, see the following topics in the [JupyterLab documentation](#):
 - [JupyterLab prerequisites](#)
 - [JupyterLab installation](#) – We recommend using `pip install jupyterlab`.
- A Node.js installation (version 12.0 or greater).
- An AWS Command Line Interface (AWS CLI) installation – For more information, see [Setting up the AWS CLI](#).
- An AWS Jupyter proxy installation (`pip install aws-jupyter-proxy`)– This extension is used with an AWS service endpoint to securely pass your AWS credentials. For more information, see [aws-jupyter-proxy](#) on GitHub.

To verify that you have the prerequisites installed, you can run a test that's similar to the following at the command line, as shown in the following example.

```
echo "  
AWS CLI:"  
which aws  
aws --version  
aws configure list  
aws sts get-caller-identity
```

```

echo "
Python (current environment):"
which python
python --version

echo "
Node.JS:"
which node
node --version

echo "
Jupyter:"
where jupyter
jupyter --version
jupyter serverextension list
pip3 freeze | grep jupyter

```

The output should look something like the following. The directories vary by operating system and configuration.

```

AWS CLI:
/usr/local/bin/aws
aws-cli/2.1.2 Python/3.7.4 Darwin/19.6.0 exe/x86_64

  Name                               Value                               Type      Location
  ----                               -
  profile                            <not set>                          None      None
  access_key                         *****VXW4                         shared-credentials-file
  secret_key                         *****MRJN                         shared-credentials-file
  region                             us-east-1                          config-file  ~/.aws/config
{
  "UserId": "",
  "Account": "111122223333",
  "Arn": "arn:aws:iam::111122223333:user/user2"
}

Python (current environment):
/usr/local/opt/python /libexec/bin/python
Python 3.8.5

Node.JS:
/usr/local/bin/node
v15.0.1

```

```
Jupyter:
/usr/local/bin/jupyter
jupyter core      : 4.6.3
jupyter-notebook  : 6.0.3
qtconsole         : 4.7.5
ipython           : 7.16.1
ipykernel         : 5.3.2
jupyter client    : 6.1.6
jupyter lab       : 2.2.9
nbconvert         : 5.6.1
ipywidgets        : 7.5.1
nbformat          : 5.0.7
traitlets         : 4.3.3

config dir: /usr/local/etc/jupyter
  aws_jupyter_proxy enabled
    - Validating...
      aws_jupyter_proxy OK
  jupyterlab enabled
    - Validating...
      jupyterlab 2.2.9 OK

aws-jupyter-proxy==0.1.0
jupyter-client==6.1.7
jupyter-core==4.7.0
jupyterlab==2.2.9
jupyterlab-pygments==0.1.2
jupyterlab-server==1.2.0
```

Configuring JupyterLab to use the extension

After you install JupyterLab, you need to configure it to secure data access and to enable server extensions.

To configure a password and encryption

1. Set a password to protect the data that you plan to add in the extension. Jupyter provides a password utility. Run the following command and enter your preferred password at the prompt.

```
jupyter notebook password
```

The output looks something like the following.

```
Enter password:  
Verify password:  
[NotebookPasswordApp] Wrote hashed password to /home/ubuntu/.jupyter/  
jupyter_notebook_config.json
```

2. Enable encryption on the Jupyter server. If you install Jupyter on your local machine, and no one can access it over the network, you can skip this step.

To set up encryption with Transport Layer Security (TLS), create a certificate customized for your environment. For more information, [Using Let's Encrypt](#) in [Securing a server](#) in the Jupyter documentation.

3. To start JupyterLab, run the following command at the command prompt.

```
jupyter lab
```

For more information, see [Starting JupyterLab](#) in the JupyterLab documentation.

4. While JupyterLab is running, you can access it at a URL similar to the following: <http://localhost:8888/lab>. If you set up encryption, use https instead of http. If you customized the port, substitute your port number instead of 8888.

Use the following procedure to enable the third-party extensions.

To enable third-party extensions in JupyterLab

1. On the JupyterLab webpage, choose the **Extension Manager** icon in the menu at left.
2. Read the warning about the risks of running third-party extensions. Only install extensions from developers that you trust.
3. To enable third-party extensions in JupyterLab, choose **Enable**.
4. Follow the prompts to rebuild and reload JupyterLab.

Enabling the DataBrew extension for JupyterLab

After you have a secure installation of JupyterLab with extensions enabled, install the DataBrew extension so you can run DataBrew in your notebook.

To install the extensions for DataBrew (console)

1. To start JupyterLab, run the following command at the command prompt.

```
jupyter lab
```

2. On the JupyterLab webpage, choose the **Extension Manager** icon in the menu at left.
3. Search for the DataBrew extension by entering "**brew**" for **Search** at top left.
4. Locate **aws_glue_databrew_jupyter** in the list, but don't click it. If you click the highlighted name of the extension, a new browser window opens with the [aws_glue_databrew_jupyter](#) page on GitHub.
5. To install the DataBrew extension, choose one of the following:
 - At the command line, run `jupyter labextension install aws_glue_databrew_jupyter`.
 - Choose **Install** at the bottom of the extension card, underneath "**aws_glue_databrew_jupyter**" in gray lettering.

DataBrew extension is compatible with JupyterLab version 1.2 and 2.x.

6. To verify that it installed, run `jupyter labextension list`. The output should look something like the following.

```
JupyterLab v2.2.9
Known labextensions:
  app dir: /usr/local/share/jupyter/lab # varies by OS
    aws_glue_databrew_jupyter v1.0.1  enabled  OK
```

7. Rebuild JupyterLab by using one of the following:
 - At the command prompt, run `jupyter lab build`.
 - In the webpage, choose **Rebuild** at top left.
8. When the build is complete, do one of the following:
 - At the command prompt, run `jupyter lab`.
 - In the webpage, choose **Reload** on the **Build Complete** message.
9. In the JupyterLab webpage, close the **Extension Manager** by choosing its icon in the menu at left.

To open the extension, choose **Launch AWS Glue DataBrew** from the **Other** section on the **Launcher** tab. The extension uses your current AWS CLI configuration for access keys and AWS region settings.

After you complete the setup, you can use the **AWS Glue DataBrew** tab to interact with DataBrew from within JupyterLab.

Getting started with AWS Glue DataBrew

You can use the following tutorial to guide you in creating your first DataBrew project. You load a sample dataset, run transformations on that dataset, build a recipe to capture those transformations, and run a job to write the transformed data to Amazon S3.

Topics

- [Prerequisites](#)
- [Step 1: Create a project](#)
- [Step 2: Summarize the data](#)
- [Step 3: Add more transformations](#)
- [Step 4: Review your DataBrew resources](#)
- [Step 5: Create a data profile](#)
- [Step 6: Transform the dataset](#)
- [Step 7: \(Optional\) Clean up](#)

Prerequisites

Before you proceed, follow the applicable instructions in [Setting up AWS Glue DataBrew](#). Then continue to [Step 1: Create a project](#).

Step 1: Create a project

In this step, you use the DataBrew console to quickly get started with a sample project.

To create a project

1. Sign in to the AWS Management Console and open the DataBrew console at <https://console.aws.amazon.com/databrew/>.
2. Make sure that your AWS Region is selected at upper-right on the DataBrew console. For a list of AWS Regions supported by DataBrew, see [DataBrew endpoints and quotas](#) in the *AWS General Reference*.
3. On the navigation pane, choose **Projects**, and then choose **Create project**.
4. On the **Project details** pane, do the following:

- For **Project name**, enter chess-project.
 - For **Attached recipe**, create a new recipe. A suggested name for the recipe is provided (chess-project-recipe).
5. On the **Select a dataset** pane, choose **Sample files**.
 6. On the **Sample files** pane, choose **Famous chess game moves**. This dataset contains detailed information on more than 20,000 games of chess.

For **Dataset name** a suggested name for the dataset is provided (chess-games).

7. On the **Access permissions** pane, choose `AwsGlueDataBrewDataAccessRole`. This is a service-linked role that lets DataBrew access your Amazon S3 buckets on your behalf.
8. Choose **Create project**, and wait until DataBrew finishes preparing the project. The window looks similar to the following.

The data that you see represents a sample from the chess-games dataset. By default, the sample consists of the first 500 rows from the dataset. You can change this project setting later.

The toolbar provides access to hundreds of data transforms that you can apply to the data.

The recipe pane at right in the DataBrew console tracks the transformations you applied so far.

Step 2: Summarize the data

In this step, you build a DataBrew recipe—a set of transformations that can be applied to this dataset and others like it. When the recipe is complete, you publish it so that it's available for use.

In the game of chess, players can be rated based on how well they perform against other players. (For more information, see https://en.wikipedia.org/wiki/Chess_rating_system). For this tutorial, you focus on only the games where both players were Class A, meaning that their ratings were 1800 or more.

To summarize the data

1. On the transformation toolbar, choose **Filter, By Condition, Greater than or equal to**.
2. Set these options as follows:
 - **Source column** - white_rating

- **Filter condition** – Greater than or equal to 1800

To see how the transform works, choose **Preview changes**. Then choose **Apply**.

3. Repeat the previous step, but this time set **Source column** to `black_rating`. After you apply your changes, the sample data contains only those games where the players on each side (black and white) were Class A or above.
4. Summarize the data to determine how many games were won by each side. To do this, on the transformation toolbar, choose **Group**.
5. For the **Group** properties, do the following:
 - a. In the first row, choose `winner` for **Column name**. Leave **Aggregate** set to **Group by**.
 - b. In the second row, choose `victory_status` for the **Column name**. Leave **Aggregate** set to **Group by**.
 - c. Choose **Add another column**.
 - d. In the third row, choose `winner` for **Column name**. Set **Aggregate** to **Count**.
 - e. For **Group type**, choose **Group as new table**. The preview pane shows you what the result will look like.
 - f. Choose **Finish**.
6. Choose **Publish** to save your work, at right on the recipe pane.
7. For **Version Description**, enter **First version of my recipe**. Then choose **Publish**.

Step 3: Add more transformations

In this step, you add more transformations to your recipe and publish another version of it. To refine our example, we use the information that not all chess games result in a clear winner; some games are played to a draw.

To add more recipe transformations and republish

1. From the transformation toolbar, choose **Filter, By Condition, Is not** to remove the games that were played to a draw.
2. Set these options as follows:
 - **Source column** - `victory_status`

- **Filter condition** – Is not draw

To add this transform to your recipe, choose **Apply**.

3. Change the data in `victory_status` so that it's more meaningful. To do this, from the transformation toolbar choose **Clean, Replace, Replace value or pattern**.
4. Set these options as follows:
 - **Source column** - `victory_status`
 - **Specify values to replace** – Value or pattern
 - **Value to be replaced** - `mate`
 - **Replace with value** - `checkmate`

To add this transform to your recipe, choose **Apply**.

5. Repeat the previous step, but change `resign` to `other player resigned`.
6. Repeat the previous step, but change `outoftime` to `time ran out`.
7. Choose **Publish** to save your work, at right on the recipe pane.

Step 4: Review your DataBrew resources

Now that you worked with a sample project, review the DataBrew resources you created so far.

To review your DataBrew resources

1. On the navigation pane, choose **Datasets**.

When you created the sample project, DataBrew created a dataset for you (`chess-games`). The source data file is stored in Amazon S3, and is in Microsoft Excel format (`chess-games.xlsx`). The file contains metadata from over 20,000 games of chess. The `chess-games` dataset provides the information that DataBrew needs to read the data in that file.

2. On the navigation pane, choose **Projects**.

You should see the project that you worked with in the previous steps (`chess-project`). Every project requires a dataset, in this case `chess-games`. Every project also requires a recipe, so that you can add data transformation steps as you go along. When you created this sample project, DataBrew created a new (empty) recipe for you, and attached it to the project.

3. On the navigation pane, choose **Recipes**, and in the **Recipe name** column, choose **chess-project-recipe**. This shows you the recipe that DataBrew created for your project, and that you've refined by adding transformation steps to it.
4. At left, view the recipe versions that have been published. Choose one of these to view its **Recipe steps** tab, which shows the recipe details and steps for that version.
5. View the **Data lineage** tab, which shows where the data came from and how it's being used. For more details, choose any of the icons in the diagram.

Step 5: Create a data profile

When you work with on a project, DataBrew displays statistics such as the number of rows in the sample and the distribution of unique values in each column. These statistics, and many more, represent a *profile* of the sample.

To request a data profile, create and run a profile job.

To profile a dataset

1. On the navigation pane, choose **Jobs**.
2. On the **Profile jobs** tab, choose **Create job**.
3. For **Job name**, enter `chess-data-profile`.
4. For **Job type**, choose **Create a profile job**.
5. On the **Job input** pane, do the following:
 - For **Run on**, choose **Dataset**.
 - Choose **Select a dataset** to view a list of available datasets, and choose `chess-games`.
6. On the **Job output settings** pane, do the following:
 - For **File type**, choose **JSON** (JavaScript Object Notation).
 - Choose **S3 location** to view a list of available Amazon S3 buckets, and choose the bucket to use. Then choose **Browse**. In the list of folders, choose `databrew-output`, and chose **Select**.
7. On the **Access permissions** pane, choose `AwsGlueDataBrewDataAccessRole`. This is a service linked role that lets DataBrew access your Amazon S3 buckets on your behalf.
8. Choose **Create and run job**. DataBrew creates a job with your settings, and then runs it.

9. On the **Job run history** pane, wait for the job status to change from Running to Succeeded.
10. To view the profile, choose **VIEW PROFILE**:



The **DATASETS** window is shown. Take some time to explore the following tabs:

- Dataset preview
- Profile overview
- Column statistics
- Data lineage statistics

Step 6: Transform the dataset

Until now, you tested your recipe on only a sample of the dataset. Now it's time to transform the entire dataset by creating a DataBrew recipe job.

When the job runs, DataBrew applies your recipe to all of the data in the dataset, and writes the transformed data to an Amazon S3 bucket. The transformed data is separate from the original dataset. DataBrew doesn't alter the source data.

Before you proceed, ensure that you have an Amazon S3 bucket in your account that you can write to. In that bucket, create a folder to capture the job output from DataBrew. To do these steps, use the following procedure.

To create an S3 bucket and folder to capture job output

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/databrew/>.

If you already have an Amazon S3 bucket available, and you have write permissions for it, skip the next step.

2. If you don't have an Amazon S3 bucket, choose **Create bucket**. For **Bucket name**, enter a unique name for your new bucket. Choose **Create bucket**.
3. From the list of buckets, choose the one that you want to use.
4. Choose **Create folder**.

5. For **Folder name**, enter databrew-output, and choose **Create folder**.

After you create an Amazon S3 bucket and folder to contain the job, run your job by using the following procedure.

To create and run a recipe job

1. On the navigation pane, choose **Jobs**.
2. On the **Recipe jobs** tab, choose **Create job**.
3. For **Job name**, enter chess-winner-summary.
4. For **Job type**, choose **Create a recipe job**.
5. On the **Job input** pane, do the following:
 - For **Run on**, choose **Dataset**.
 - Choose **Select a dataset** to view a list of available datasets, and choose chess-games.
 - Choose **Select a recipe** to view a list of available recipes, and choose chess-project-recipe.
6. On the **Job output settings** pane, do the following:
 - **File type** – chose **CSV** (comma-separated values).
 - **S3 location** - choose this field to view a list of available Amazon S3 buckets, and choose the bucket to use. Then choose **Browse**. In the list of folders, choose databrew-output, and choose **Select**.
7. On the **Access permissions** pane, choose AwsGlueDataBrewDataAccessRole. This service-linked role lets DataBrew access your Amazon S3 buckets on your behalf.
8. Choose **Create and run job**. DataBrew creates a job with your settings, and then runs it.
9. On the **Job run history** pane, wait for the job status to change from Running to Succeeded.
10. Choose **Output** to access the Amazon S3 console. Choose your S3 bucket, and then choose the databrew-output folder to access the job output.
11. (Optional) Choose **Download** to download the file and view its contents.

Step 7: (Optional) Clean up

The walkthrough is complete. You can keep using the DataBrew and Amazon S3 resources that you created, or delete them.

To clean up resources

1. Open the DataBrew console at <https://console.aws.amazon.com/databrew/>, and on the navigation pane, choose **Projects**.
2. Choose your project (**Sample project**). For **Actions**, choose **Delete**.
3. On the **Delete Sample project** pane, choose **Delete attached recipe**. Then choose **Delete**. Your project, along with its recipe and jobs, will be deleted.
4. On the navigation pane, choose **Datasets**.
5. Choose your dataset (chess-games), and for **Actions**, choose **Delete**.
6. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>. Delete the databrew-output folder and its contents.

(Optional) If you're sure that you no longer need your Amazon S3 bucket, you can delete it.

Connecting to data with AWS Glue DataBrew

In AWS Glue DataBrew, a *dataset* represents data that's either uploaded from a file or stored elsewhere. For example, data can be stored in Amazon S3, in a supported JDBC data source, or an AWS Glue Data Catalog. If you're not uploading a file directly to DataBrew, the dataset also contains details on how DataBrew can connect to the data.

When you create your dataset (for example, `inventory-dataset`), you enter the connection details only once. From that point, DataBrew can access the underlying data for you. With this approach, you can create projects and develop transformations for your data, without having to worry about connection details or file formats.

Topics

- [Supported file types for data sources](#)
- [Supported connections for data sources and outputs](#)
- [Using datasets in AWS Glue DataBrew](#)
- [Connecting to your data](#)
- [Connecting to data in a text file with DataBrew](#)
- [Connecting data in multiple files in Amazon S3](#)
- [Data types](#)
- [Advanced data types](#)

Supported file types for data sources

The following file requirements apply to files stored in Amazon S3 and to files that you upload from a local drive. DataBrew supports the following file formats: comma-separated value (CSV), Microsoft Excel, JSON, ORC, and Parquet. You can use files with a nonstandard extension or no extension if the file is of one of the supported types.

If DataBrew is unable to infer the file type, make sure to select the correct file type yourself (CSV, Excel, JSON, ORC, or Parquet). Compressed CSV, JSON, ORC, and Parquet files are supported, but CSV and JSON files must include the compression codec as the file extension. If you are importing a folder, all files in the folder must be of the same file type.

File formats and supported compression algorithms are shown in the following table.

 Note

CSV, Excel, and JSON files must be encoded with Unicode (UTF-8).

Format	File extension (optional)	Extensions for compressed files (required)
Comma-separated values	.csv	.gz .snappy .lz4 .bz2 .deflate
Microsoft Excel workbook	.xlsx	No compression support
JSON (JSON document and JSON lines)	.json, .jsonl	.gz .snappy .lz4 .bz2 .deflate
Apache ORC	.orc	.zlib .snappy
Apache Parquet	.parquet	.gz .snappy .lz4

Supported connections for data sources and outputs

You can connect to the following data sources for DataBrew recipe jobs. These include any source of data that isn't a file you're uploading directly to DataBrew. The data source that you're using might be called a database, a data warehouse, or something else. We refer to all data providers as data sources or connections.

You can create a dataset using any of the following as data sources.

You can also use Amazon S3, AWS Glue Data Catalog, or JDBC databases supported through Amazon RDS for the output of DataBrew recipe jobs. Amazon AppFlow and AWS Data Exchange aren't supported data stores for the output of DataBrew recipe jobs.

- **Amazon S3**

You can use S3 to store and protect any amount of data. To create a dataset, you specify an S3 URL where DataBrew can access a data file, for example: `s3://your-bucket-name/inventory-data.csv`

DataBrew can also read all of the files in an S3 folder, which means that you can create a dataset that spans multiple files. To do this, specify an S3 URL in this form: `s3://your-bucket-name/your-folder-name/`.

DataBrew supports only the following Amazon S3 storage classes: Standard, Reduced Redundancy, Standard-IA, and S3 One Zone-IA. DataBrew ignores files with other storage classes. DataBrew also ignores empty files (files containing 0 bytes). For more information about Amazon S3 storage classes, see [Using Amazon S3 storage classes](#) in the *Amazon S3 Console User Guide*.

- **AWS Glue Data Catalog**

You can use the Data Catalog to define references to data that's stored in the AWS Cloud. With the Data Catalog, you can build connections to individual tables in the following services:

- Data Catalog Amazon S3
- Data Catalog Amazon Redshift
- Data Catalog Amazon RDS
- AWS Glue

DataBrew can also read all of the files in an Amazon S3 folder, which means that you can create a dataset that spans multiple files. To do this, specify an Amazon S3 URL in this form: `s3://your-bucket-name/your-folder-name/`

To be used with DataBrew, Amazon S3 tables defined in the AWS Glue Data Catalog, must have a table property added to them called a `classification`, which identifies the format of the data as `csv`, `json`, or `parquet`, and the `typeOfData` as `file`. If the table property was not added when the table was created, you can add it using the AWS Glue console.

DataBrew supports only the Amazon S3 storage classes Standard, Reduced Redundancy, Standard-IA, and S3 One Zone-IA. DataBrew ignores files with other storage classes. DataBrew also ignores empty files (files containing 0 bytes). For more information about Amazon S3 storage classes, see [Using Amazon S3 storage classes](#) in the *Amazon S3 Console User Guide*.

DataBrew can also access AWS Glue Data Catalog S3 tables from other accounts if an appropriate resource policy is created. You can create a policy in the AWS Glue console on the **Settings** tab under **Data Catalog**. The following is an example policy specifically for a single AWS Region.

Warning

This is a highly permissive resource policy that grants `*$ACCOUNT_TO*` unrestricted access to the Data Catalog of `*$ACCOUNT_FROM*`. In most cases, we recommend that you lock your resource policy down to specific catalogs or tables. For more information, see [AWS Glue resource policies for access control](#) in the *AWS Glue Developer Guide*.

In some cases, you might want to create a project or run a job in AWS Glue DataBrew in `*$ACCOUNT_TO*` with an AWS Glue Data Catalog S3 table in `*$ACCOUNT_FROM*` that points to an S3 location that is also in `*$ACCOUNT_FROM*`. In such cases, the IAM role used when creating the project and job in `*$ACCOUNT_TO*` must have permission to list and get objects in that S3 location from `*$ACCOUNT_FROM*`. For more information, see [Granting cross-account access](#) in the *AWS Glue Developer Guide*.

- **Data connected using JDBC drivers**

You can create a dataset by connecting to data with a supported JDBC driver. For more information, see [Using drivers with AWS Glue DataBrew](#).

DataBrew officially supports the following data sources using Java Database Connectivity (JDBC):

- Microsoft SQL Server
- MySQL
- Oracle
- PostgreSQL
- Amazon Redshift
- Snowflake Connector for Spark

The data sources can be located anywhere that you can connect to them from DataBrew. This list includes only JDBC connections that we've tested and can therefore support.

Amazon Redshift and Snowflake Connector for Spark data sources can be connected in either of the following ways:

- With a table name.
- With a SQL query that spans multiple tables and operations.

SQL queries are executed when you start a project or a job run.

To connect to data that requires an unlisted JDBC driver, make sure that the driver is compatible with JDK 8. To use the driver, store it in S3 in a bucket where you can access it with your IAM role for DataBrew. Then point your dataset at the driver file. For more information, see [Using drivers with AWS Glue DataBrew](#).

Example query for a SQL-based dataset:

```
SELECT
    *
FROM
    public.customer as c
JOIN
    public.customer_address as ca on c.current_address=ca.current_address
WHERE
    ca.address_id>0 AND ca.address_id<10001 ORDER BY ca.address_id
```

Limitations of Custom SQL

If you use a JDBC connection to access data for a DataBrew dataset, keep in mind the following:

- AWS Glue DataBrew does not validate the custom SQL you provide as part of dataset creation. The SQL query will be executed when you start a project or job run. DataBrew takes the query you provide and passes it to the database engine using the default or provided JDBC drivers.
- A dataset created with an invalid query will fail when it is used in a project or job. Validate your query before creating the dataset.
- The **Validate SQL** feature is only available for Amazon Redshift-based data sources.
- If you want to use a dataset in a project, limit SQL query runtime to under three minutes to avoid a timeout during project loading. Check the query runtime before creating a project.
- **Amazon AppFlow**

Using Amazon AppFlow, you can transfer data into Amazon S3 from third-party Software-as-a-Service (SaaS) applications such as Salesforce, Zendesk, Slack, and ServiceNow. You can then use the data to create a DataBrew dataset.

In Amazon AppFlow, you create a connection and a flow to transfer data between your third-party application and a destination application. When using Amazon AppFlow with DataBrew, make sure that the Amazon AppFlow destination application is Amazon S3. Amazon AppFlow destination applications other than Amazon S3 don't appear in the DataBrew console. For more information on transferring data from your third-party application and creating Amazon AppFlow connections and flows, see the [Amazon AppFlow documentation](#).

When you choose **Connect new dataset** in the **Datasets** tab of DataBrew and click Amazon AppFlow, you see all flows in Amazon AppFlow that are configured with Amazon S3 as the destination application. To use a flow's data for your dataset, choose that flow.

Choosing **Create flow**, **Manage flows**, and **View details** for Amazon AppFlow in the DataBrew console opens the Amazon AppFlow console so that you can perform those tasks.

After you create a dataset from Amazon AppFlow, you can run the flow and view the latest flow run details when viewing dataset details or job details. When you run the flow in DataBrew, the dataset is updated in S3 and is ready to be used in DataBrew.

The following situations can arise when you select an Amazon AppFlow flow in the DataBrew console to create a dataset:

- **Data hasn't been aggregated** - If the flow trigger is **Run on demand** or is **Run on schedule** with full data transfer, make sure to aggregate the data for the flow before using it to create a DataBrew dataset. Aggregating the flow combines all records in the flow into a single

file. Flows with the trigger type **Run on schedule** with incremental data transfer, or **Run on event** don't require aggregation. To aggregate data in Amazon AppFlow, choose **Edit flow configuration > Destination details > Additional settings > Data transfer preference**.

- **Flow hasn't been run** - If the run status for a flow is empty, it means one of the following:
 - If the trigger for running the flow is **Run on demand**, the flow has not yet been run.
 - If the trigger for running the flow is **Run on event**, the triggering event has not yet occurred.
 - If the trigger for running the flow is **Run on schedule**, a scheduled run has not yet occurred.

Before creating a dataset with a flow, choose **Run flow** for that flow.

For more information, see [Amazon AppFlow flows](#) in the Amazon AppFlow User Guide.

- **AWS Data Exchange**

You can choose from hundreds of third-party data sources that are available in AWS Data Exchange. By subscribing to these data sources, you get the most up-to-date version of the data.

To create a dataset, you specify the name of a AWS Data Exchange data product that you're subscribed to and entitled to use.

Using datasets in AWS Glue DataBrew

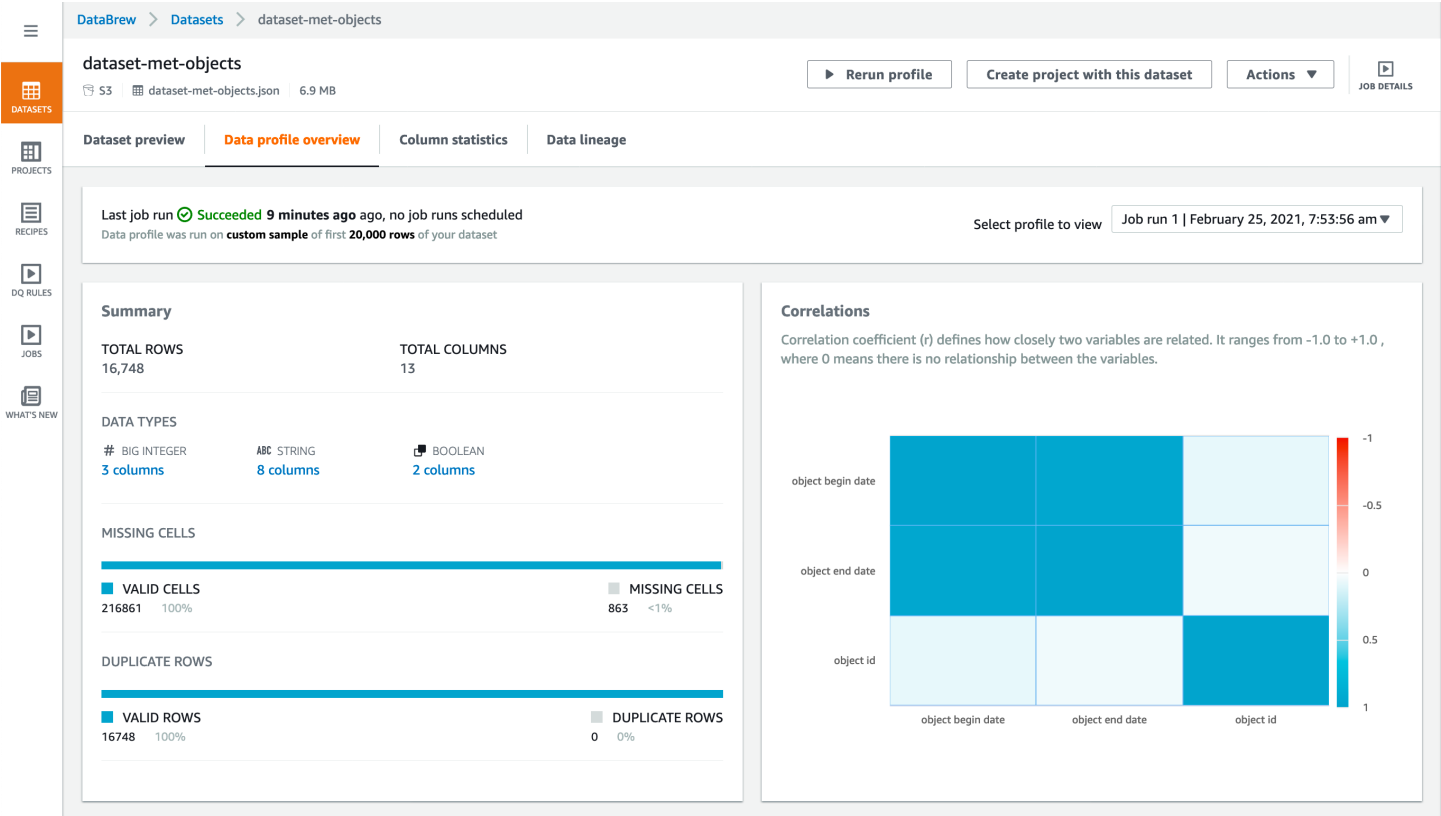
To view a list of your datasets in the DataBrew console, choose **DATASET** at left. In the datasets page, you can view detailed information for each dataset by clicking its name or choosing **Actions**, **Edit** from its context menu.

To create a new dataset, you choose **DATASET**, **Connect new dataset**. Different data sources have different connection parameters, and you enter these so that DataBrew can connect. When you save your connection and choose **Create dataset**, DataBrew connects to your data and begins loading data. For more information, see [Connecting to your data](#).

The dataset page has the following elements to help you explore your data.

Dataset preview – On this tab, you can find connection information for the dataset and an overview of the overall structure of the dataset, as shown following.

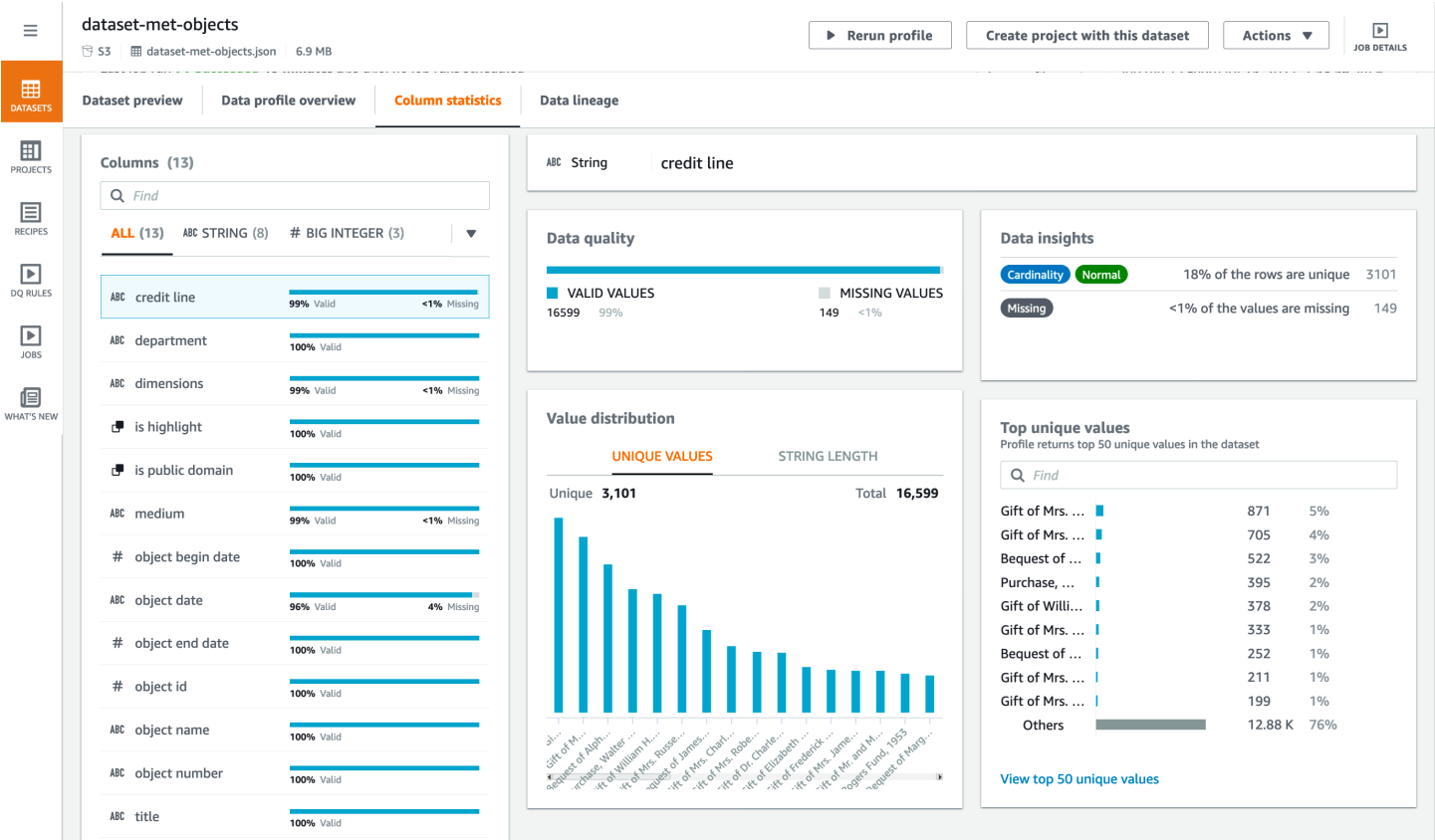
Data profile overview – On this tab, you can find a graphical data profile of statistics and volumetrics for your dataset, as shown following.



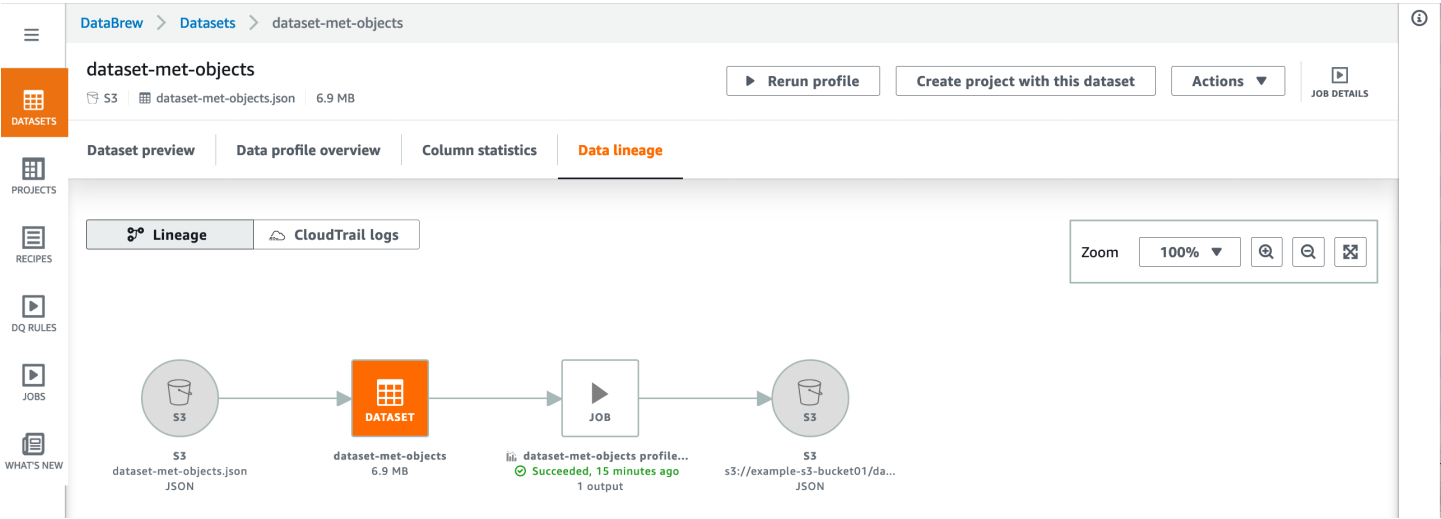
 **Note**

To create a data profile, run a DataBrew profile job on your dataset. For information about how to do this, see [Step 5: Create a data profile](#).

Column statistics – On this tab, you can find detailed statistics about each column in your dataset, as shown following.



Data lineage – This tab shows a graphical representation of how your dataset was created and how it's used in DataBrew, as shown following.



Topics

- [Deleting a dataset](#)

Deleting a dataset

If you no longer need a dataset, you can delete it. Deleting a dataset doesn't affect the underlying data source in any way. It simply removes the information that DataBrew used to access the data source.

You can't delete a dataset if any other DataBrew resources rely on it. For example, if you currently have a DataBrew project that uses the dataset, delete the project first before you delete the dataset.

To delete a dataset, choose **Dataset** from the navigation pane. Choose the dataset that you want to delete, and then for **Actions**, choose **Delete**.

Connecting to your data

For more information on connecting to the following data sources, choose the section that applies to you.

- **AWS Glue Data Catalog** – You can use the Data Catalog to define references to data objects stored in the AWS Cloud, including the following services:
 - Amazon Redshift
 - Aurora MySQL
 - Aurora PostgreSQL
 - Amazon RDS for MySQL
 - Amazon RDS for PostgreSQL

DataBrew recognizes all Lake Formation permissions that have been applied to Data Catalog resources, so DataBrew users can only access these resources if they're authorized.

To create a dataset, you specify a Data Catalog database name and a table name. DataBrew takes care of the other connection details.

- **AWS Data Exchange** – You can choose from hundreds of third-party data sources that are available in AWS Data Exchange. By subscribing to these data sources, you always have the most up-to-date version of the data.

To create a dataset, you specify the name of a Data Exchange data product that you're subscribed to or entitled to use.

- **JDBC driver connections** – You can create a dataset by connecting DataBrew to a JDBC-compatible data source. DataBrew supports connecting to the following sources through JDBC:
 - Amazon Redshift
 - Microsoft SQL Server
 - MySQL
 - Oracle
 - PostgreSQL
 - Snowflake

Topics

- [Using drivers with AWS Glue DataBrew](#)
- [Supported JDBC drivers](#)

Using drivers with AWS Glue DataBrew

A *database driver* is a file or URL that implements a database connection protocol, for example Java Database Connectivity (JDBC). The driver functions as an adaptor or a translator between a specific database management system (DBMS) and another system.

In this case, it allows AWS Glue DataBrew to connect to your data. Then you can access a database object, like a table or view, from a supported data source. The data source that you're using might be called a database, a data warehouse, or something else. However, for the purpose of this documentation we refer to all data providers as data sources or connections.

To use a JDBC driver or jar file, download the file or files you need and put them in an S3 bucket. The IAM role that you use to access the data needs to have read permissions for both the driver files.

Note

With AWS Glue 4.0, connecting to Snowflake as a data source is supported natively. You don't need to provide custom jar files. In AWS Glue DataBrew, choose Snowflake as the External source connection and provide the URL of your Snowflake instance. The URL will use a hostname in the form `https://account_identifier.snowflakecomputing.com`.

Provide the data access credentials, Snowflake database name, and Snowflake schema name. Additionally, if your Snowflake user does not have a default warehouse set, you will need to provide a warehouse name.

Snowflake connections use an AWS Secrets Manager secret to provide credential information. Your project and job roles in must have permission to read this secret.

Connection access

External source

 **Snowflake**
JDBC Spark connector

JDBC URL

JDBC URL for your database.

JDBC URL format for Snowflake database is jdbc:snowflake://<account_name>.snowflakecomputing.com/?db=<database_name>&warehouse=<warehouse_name>

Database access credentials

☐ Enter credentials ☒ Connect with Secrets Manager

Secrets

Choose a secret with keys "user" and "password" from [Secrets Manager](#)

Choose a secret

To use drivers with DataBrew

1. Find out which version of your data source you're on, using the method provided by the product.
2. Find the latest version of connectors and driver required. You can locate this information on the data providers website.
3. Download the required version of the JDBC files. These are normally stored as Java ARchives (.JAR) files.
4. Either upload the drivers from the console to your S3 bucket or provide the S3 path to your .JAR files.
5. Enter the basic connection details, for example class, instance, and so on.
6. Enter any additional configuration information that your data source needs, for example virtual private cloud (VPC) information.

Supported JDBC drivers

Product	Supported version	Driver instructions and downloads	SQL queries supported
Microsoft SQL Server	v6.x or higher	Microsoft JDBC Driver for SQL Server	Not supported
MySQL	v5.1 or higher	MySQL Connectors	Not supported
Oracle	v11.2 or higher	Oracle JDBC downloads	Not supported
PostgreSQL	v4.2.x or higher	PostgreSQL JDBC driver	Not supported
Amazon Redshift	v4.1 or higher	Connecting to Amazon Redshift with JDBC	Supported
Snowflake	To see your Snowflake version, use CURRENT_VERSION as described	To connect to Snowflake you need both of the following: • Snowflake JDBC Driver • Snowflake Connector for Spark	Supported

Product	Supported version	Driver instructions and downloads	SQL queries supported
	in the Snowflake documentation.		

To connect to databases or data warehouses that require a different version of the driver from what DataBrew natively supports, you can provide a JDBC driver of your choice. The driver must be compatible with JDK 8 or Java 8. For instructions on how to find the latest driver version for your database, see [Using drivers with AWS Glue DataBrew](#).

Connecting to data in a text file with DataBrew

You can configure the following format options for the input files that DataBrew supports:

- **Comma-separated value (CSV) files**
 - **Delimiters**

The default delimiter is a comma for .csv files. If your file uses a different delimiter, choose the delimiter for **CSV delimiter** in the **Additional configurations** section when you create your dataset. The following delimiters are supported for .csv files:

- Comma (,)
- Colon (:)
- Semi-colon (;)
- Pipe (|)
- Tab (\t)
- Caret (^)
- Backslash (\)
- Space
- **Column header values**

Your CSV file can include a header row as the first row of the file. If it doesn't, DataBrew creates a header row for you.

- If your CSV file includes a header row, choose **Treat first row as header**. If you do, the first row of your CSV file is treated as containing the column header values.
- If your CSV file doesn't include a header row, choose **Add default header**. If you do, DataBrew creates a header row for the file and doesn't treat your first row of data as containing header values. The headers that DataBrew creates consist of an underscore and a number for each column in the file, in the format `Column_1`, `Column_2`, `Column_3`, and so on.

- **JSON files**

DataBrew supports two formats for JSON files, JSON Lines and JSON document. JSON Lines files contain one row per line. In JSON document files, all rows are contained in a single JSON structure or an array. You can specify your JSON file type in the **Additional configurations** section when you create a JSON dataset. The default format is JSON Lines.

- **Excel files**

The following apply to Excel sheets in DataBrew:

- **Excel sheet loading**

By default, DataBrew loads the first sheet in your Excel file. However, you can specify a different sheet number or sheet name in the **Additional configurations** section when you create an Excel dataset.

- **Column header values**

Your Excel sheets can include a header row as the first row of the file, but if they don't, DataBrew will create a header row for you.

- If your Excel sheets include a header row, choose **Treat first row as header**. If you do, the first row of your Excel sheets is treated as containing the column header values.
- If your Excel file doesn't include a header row, choose **Add default header**. By doing this, you specify that DataBrew should create a header row for the file and not treat your first row of data as containing header values. The headers that DataBrew creates consist of an underscore and a number for each column in the file, in the format `Column_1`, `Column_2`, `Column_3`, and so on.

Connecting data in multiple files in Amazon S3

With the DataBrew console, you can navigate Amazon S3 buckets and folders and choose a file for your dataset. However, a dataset doesn't need to be limited to one file.

Suppose that you have an S3 bucket named `my-databrew-bucket` that contains a folder named `databrew-input`. In that folder, suppose that you have a number of JSON files, all with the same file format and `.json` file extension. On the console, you can specify a source URL of `s3://my-databrew-bucket/databrew-input/`. On the DataBrew console, you can then choose this folder. Your dataset consists of all the JSON files in that folder.

DataBrew can process all of the files in an S3 folder, but only if the following conditions are true:

- All of the files in the folder have the same format.
- All of the files in the folder have the same file extension.

For more information on supported file formats and extensions, see [DataBrew input formats](#).

Schemas when using multiple files as a dataset

When using multiple files as a DataBrew dataset, the schemas have to be the same across all the files. Otherwise, the Project Workspace automatically tries to choose one of the schemas from the multiple files and tries to conform the rest of the dataset files to that schema. This behavior results in the view that is shown during Project Workspace to be irregular, and as a result, the job output will also be irregular.

If your files must have different schemas, you need to create multiple datasets and profile them separately.

Using parameterized paths for Amazon S3

In some cases, you might want to create a dataset with files that follow a certain naming convention, or a dataset that can span multiple Amazon S3 folders. Or you might want to reuse the same dataset for identically structured data that is periodically generated in an S3 location with a path that depends on certain parameters. An example is a path named for the date of data production.

DataBrew supports this approach with parameterized S3 paths. A *parameterized path* is an Amazon S3 URL containing regular expressions or custom path parameters, or both.

Defining a dataset with an S3 path using regular expressions

Regular expressions in the path can be useful to match several files from one or more folders and at the same time filter out unrelated files in those folders.

Here is a couple of examples:

- Define a dataset including all JSON files from a folder whose name begins with `invoice`.
- Define a dataset including all files in folders with `2020` in their names.

You can implement this type of approach by using regular expressions in a dataset S3 path. These regular expressions can replace any substring in the key of the S3 URL (but not the bucket name).

As an example of a key in an S3 URL, see the following. Here, `my-bucket` is the bucket name, `US East (Ohio)` is the AWS Region, and `puppy.png` is the key name.

```
https://my-bucket.s3.us-west-2.amazonaws.com/puppy.png
```

In a parameterized S3 path, any characters between two angle brackets (`<` and `>`) are treated as regular expressions. Two examples are the following:

- `s3://my-databrew-bucket/databrew-input/invoice<.*>/data.json` matches all files named `data.json`, within all of the subfolders of `databrew-input` whose names begin with `invoice`.
- `s3://my-databrew-bucket/databrew-input/<.*>2020<.*>/` matches all files in folders with `2020` in their names.

In these examples, `.*` matches zero or more characters.

Note

You can only use regular expressions in the key part of the S3 path—the part that goes after the bucket name. Thus, `s3://my-databrew-bucket/<.*>-input/` is valid, but `s3://my-<.*>-bucket/<.*>-input/` isn't.

We recommend that you test your regular expressions to ensure that they match only the S3 URLs that you want, and not ones that you don't want.

Here are some other examples of regular expressions:

- `<\d{2}>` matches a string that consists of exactly two consecutive digits, for example `07` or `03`, but not `1a2`.
- `<[a-z]+.*>` matches a string that begins with one or more lowercase Latin letters and has zero or more other characters after it. An example is `a3`, `abc/def`, or `a-z`, but not `A2`.
- `<[^/]+>` matches a string that contains any characters except for a slash (`/`). In an S3 URL, slashes are used for separating folders in the path.
- `<.*=.*>` matches a string that contains an equals sign (`=`), for example `month=02`, `abc/day=2`, or `=10`, but not `test`.
- `<\d.*\d>` matches a string that begins and ends with a digit and can have any other characters in between the digits, for example `1abc2`, `01-02-03`, or `2020/Jul/21`, but not `123a`.

Defining a dataset with an S3 path using custom parameters

Defining a parameterized dataset using custom parameters offers advantages over using regular expressions when you might want to provide parameters for an S3 location:

- You can achieve the same results as with a regular expression, without needing to know the syntax for regular expressions. You can define parameters using familiar terms like "starts with" and "contains."
- When you define a dynamic dataset using parameters in the path, you can include a time range in your definition, such as "past month" or "past 24 hours." That way, your dataset definition will be used later with new incoming data.

Here are some examples of when you might want to use dynamic datasets:

- To connect multiple files that are partitioned by *last updated* date or other meaningful attributes into a single dataset. You can then capture these partition attributes as additional columns in a dataset.
- To restrict files in a dataset to S3 locations that satisfy certain conditions. For example, suppose that your S3 path contains date-based folders like `folder/2021/04/01/`. In this case, you can parameterize the date and restrict it to a certain range like "between Mar 01 2021 and Apr 01 2021" or "Past week."

To define a path using parameters, define the parameters and add them to your path using the following format:

```
s3://my-databrew-bucket/some-folder/{parameter1}/file-{parameter2}.json
```

Note

As with regular expressions in an S3 path, you can only use parameters in the key part of the path—the part that goes after the bucket name.

Two fields are required in a parameter definition, name and type. The type can be **String**, **Number**, or **Date**. Parameters of type **Date** must have a definition of the date format so that DataBrew can correctly interpret and compare date values. Optionally, you can define matching conditions for a parameter. You can also choose to add matching values of a parameter as a column to your dataset when it's being loaded by a DataBrew job or interactive session.

Example

Let's consider an example of defining a dynamic dataset using parameters in the DataBrew console. In this example, assume that the input data is regularly written into an S3 bucket using locations like these:

- `s3://databrew-dynamic-datasets/new-cases/UR/daily-report-2021-03-30.csv`
- `s3://databrew-dynamic-datasets/new-cases/UR/daily-report-2021-03-31.csv`
- `s3://databrew-dynamic-datasets/new-cases/US/daily-report-2021-03-30.csv`
- `s3://databrew-dynamic-datasets/new-cases/US/daily-report-2021-03-31.csv`

There are two dynamic parts here: a country code, like US, and a date in the file name like 2021-03-30. Here, you can apply the same cleanup recipe for all files. Let's say that you want to perform your cleanup job daily. Following is how you can define a parameterized path for this scenario:

1. Navigate to a specific file.
2. Then select a varying part, like a date, and replace it with a parameter. In this case, replace a date.

Enter your source from S3 [Info](#)

For you to select a folder, all files in the folder need to share the same file type. If there are different schemas, they will be merged.

`s3://databrew-dynamic-datasets/new-cases/US/daily-report-2021-03-23.csv`

Format is: `s3://bucket/prefix`

[S3 Buckets](#) > [databrew-dynamic-datasets](#) > [new-cases](#) > [US](#)

Create custom parameter

☐ Specify number

[Latest](#)

☐ Specify last update

3. Open the context (right-click) menu for **Create custom parameter** and set properties for it:

- Name: report date
- Type: Date
- Date format: yyyy-MM-dd (selected from the predefined formats)
- Conditions (Time range): Past 24 hours
- Add as column: true (checked)

Keep other fields at their default values.

4. Choose **Create**.

After you do, you see the updated path, as in the following screenshot.

Enter your source from S3 [Info](#)

For you to select a folder, all files in the folder need to share the same file type. If there are different schemas, they will be merged.

`s3://databrew-dynamic-datasets/new-cases/US/daily-report-{report date}.csv`

Format is: `s3://bucket/prefix`

Matching files for parameter(s) are selected [Clear parameters](#)

Matching files (6)

6 matching files were found in all records

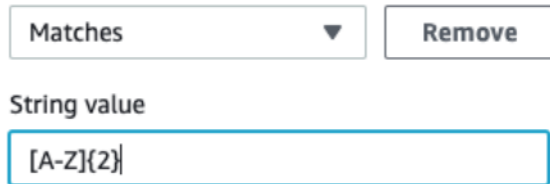
< 1 >

Now you can do the same for the country code and parameterize it as follows:

- Name: country code

- Type: String
- Add as column: true (checked)

You don't have to specify conditions if all values are relevant. In the new-cases folder, for example, we only have subfolders with country codes, so there's no need for conditions. If you had other folders to exclude, you might use the following condition.



Matches ▼ Remove

String value

[A-Z]{2}

This approach limits the subfolders of new cases to contain two capital Latin characters.

After this parameterization, you have only matching files in our dataset and can choose **Create Dataset**.

Note

When you use relative time ranges in conditions, the time ranges are evaluated when the dataset is loaded. This is true whether they are predefined time ranges like "Past 24 hours" or custom time ranges like "5 days ago". This evaluation approach applies whether the dataset is loaded during an interactive session initialization or during a job start.

After you choose **Create Dataset**, your dynamic dataset is ready to use. As an example, you might use it first to create a project and define a cleanup recipe using an interactive DataBrew session. Then you might create a job that is scheduled to run daily. This job might apply the cleanup recipe to the dataset files that meet the conditions of your parameters at the time when the job starts.

Supported conditions for dynamic datasets

You can use conditions for filtering matching S3 files using parameters or the last modified date attribute.

Following, you can find lists of supported conditions for each parameter type.

Conditions used with String parameters

Name in DataBrew SDK	SDK synonyms	Name in DataBrew console	Description
is	eq, ==	Is exactly	The value of the parameter is the same as the value that was provided in the condition.
is not	not eq, !=	Is not	The value of the parameter isn't the same as the value that was provided in the condition.
contains		Contains	The string value of the parameter contains the value that was provided in the condition.
not contains		Does not contain	The string value of the parameter doesn't contain the value that was provided in the condition.
starts_with		Starts with	The string value of the parameter starts with the value that was provided in the condition.
not starts_with		Does not start with	The string value of the parameter doesn't start with

Name in DataBrew SDK	SDK synonyms	Name in DataBrew console	Description
			the value that was provided in the condition.
ends_with		Ends with	The string value of the parameter ends with the value that was provided in the condition.
not ends_with		Does not end with	The string value of the parameter doesn't end with the value that was provided in the condition.
matches		Matches	The value of the parameter matches the regular expression provided in the condition.
not matches		Does not match	The value of the parameter doesn't match the regular expression provided in the condition.

Note

All conditions for String parameters use case-sensitive comparison. If you aren't sure about the case used in an S3 path, you can use the "matches" condition with a regular expression value that starts with `(?i)`. Doing this results in a case-insensitive comparison.

For example, suppose that you want your string parameter to start with abc, but Abc or ABC are also possible. In this case, you can use the "matches" condition with `(?i)^abc` as the condition value.

Conditions used with Number parameters

Name in DataBrew SDK	SDK synonyms	Name in DataBrew console	Description
is	eq, ==	Is exactly	The value of the parameter is the same as the value that was provided in the condition.
is not	not eq, !=	Is not	The value of the parameter isn't the same as the value that was provided in the condition.
less_than	lt, <	Less than	The numeric value of the parameter is less than the value that was provided in the condition.
less_than_equal	lte, <=	Less than or equal to	The numeric value of the parameter is less than or equal to the value that was provided in the condition.
greater_than	gt, >	Greater than	The numeric value of the parameter is greater than

Name in DataBrew SDK	SDK synonyms	Name in DataBrew console	Description
			the value that was provided in the condition.
greater_than_equal	gte, >=	Greater than or equal to	The numeric value of the parameter is greater than or equal to the value that was provided in the condition.

Conditions used with Date parameters

Name in DataBrew SDK	Name in DataBrew console	Condition value format (SDK)	Description
after	Start	ISO 8601 date format like 2021-03-30T01:00:00Z or 2021-03-30T01:00-07:00	The value of the date parameter is after the date provided in the condition.
before	End	ISO 8601 date format like 2021-03-30T01:00:00Z or 2021-03-30T01:00-07:00	The value of the date parameter is before the date provided in the condition.
relative_after	Start (relative)	Positive or negative number of time units, like -48h or +7d.	The value of the date parameter is after the relative date provided in the condition. Relative dates are evaluated when the

Name in DataBrew SDK	Name in DataBrew console	Condition value format (SDK)	Description
			dataset is loaded, either when an interactive session is initialized or when an associated job is started. This is the moment that is called "now" in the examples.
relative_before	End (relative)	Positive or negative number of time units, like -48h or +7d.	The value of the date parameter is before the relative date provided in the condition. Relative dates are evaluated when the dataset is loaded, either when an interactive session is initialized or when an associated job is started. This is the moment that is called "now" in the examples.

If you use the SDK, provide relative dates in the following format: $\pm\{\text{number_of_time_units}\}\{\text{time_unit}\}$. You can use these time units:

- -1h (1 hour ago)
- +2d (2 days from now)
- -120m (120 minutes ago)

- 5000s (5,000 seconds from now)
- -3w (3 weeks ago)
- +4M (4 months from now)
- -1y (1 year ago)

Relative dates are evaluated when the dataset is loaded, either when an interactive session is initialized or when an associated job is started. This is the moment that is called "now" in the examples preceding.

Configuring settings for dynamic datasets

Besides providing a parameterized S3 path, you can configure other settings for datasets with multiple files. These settings are filtering S3 files by their last modified date and limiting the number of files.

Similar to setting a date parameter in a path, you can define a time range when matching files were updated and include only those files into your dataset. You can define these ranges using either absolute dates like "March 30, 2021" or relative ranges like "Past 24 hours".

☒ Specify last updated date range

Past 24 hours ▼

To limit the number of matching files, select a number of files that is greater than 0 and whether you want the latest or the oldest matching files.

Choose filtered files [Info](#)

☒ Specify number of files to include

Latest ▼

10

files

Data types

The data for each column of your dataset are converted to one of the following data types:

- **byte** – 1-byte signed integer numbers. The range of numbers is from -128 to 127.
- **short** – 2-byte signed integer numbers. The range of numbers is from -32768 to 32767.
- **integer** – 4-byte signed integer numbers. The range of numbers is from -2147483648 to 2147483647.

- **long** – 8-byte signed integer numbers. The range of numbers is from -9223372036854775808 to 9223372036854775807.
- **float** – 4-byte single-precision floating point numbers.
- **double** – 8-byte double-precision floating point numbers.
- **decimal** – Signed decimal numbers with up to 38 digits total and 18 digits after the decimal point.
- **string** – Character string values.
- **boolean** – Boolean type has one of two possible values: `true` and `false` or `yes` and `no`.
- **timestamp** – Values comprising fields year, month, day, hour, minute, and second.
- **date** – Values comprising fields year, month and day.

Advanced data types

Advanced data types are data types that DataBrew detects within a string column in a project, and therefore are not part of a dataset. For information about advanced data types, see [Advanced data types](#).

Advanced data types

Advanced data types are data types that DataBrew detects within a string column in a project by means of pattern matching. When you click on a string column, the column is flagged as the corresponding advanced data type if 50% or more of the values in the column meet the criteria for that data type.

The data types DataBrew can detect are:

- Date/timestamp
- SSN
- Phone number
- Email
- Credit card
- Gender
- IP address
- URL

- Zipcode
- Country
- Currency
- State
- City

You can use the following transforms to work with advanced data types:

- [GET_ADVANCED_DATATYPE](#): Given a string column, identifies the advanced data type of the column, if any.
- [EXTRACT_ADVANCED_DATATYPE_DETAILS](#): Extracts details for an advanced data type.
- [ADVANCED_DATATYPE_FILTER](#): Filters a current source column based on advanced data type detection.
- [ADVANCED_DATATYPE_FLAG](#): Creates a new flag column based on the values for the current source column.

Validating data quality in AWS Glue DataBrew

To ensure the quality of your datasets, you can define a list of data quality rules in a ruleset. A *ruleset* is a set of rules that compare different data metrics against expected values. If any of a rule's criteria isn't met, the ruleset as a whole fails validation. You can then inspect individual results for each rule. For any rule that causes a validation failure, you can make the necessary corrections and revalidate.

Examples of rules include the following:

- Value in column "APY" is between 0 and 100
- Number of missing values in column group_name doesn't exceed 5%

You can define each rule for an individual column or independently apply it to several selected columns, for example:

- Max value doesn't exceed 100 for columns "rate", "pay", "increase".

A rule can consist of multiple simple checks. You can define whether all of them should be true or any, for example:

- Value in column "ProductId" should start with "asin-" AND length of value in column "ProductId" is 32.

You can verify rules against either aggregate values such as max, min, or number of duplicate values where there is only one value being compared, or nonaggregate values in each row of a column. In the latter case, you can also define a "passing" threshold such as value in columnA > value in columnB for at least 95% of rows.

As with profile information, you can define column-level data quality rules only for columns of simple types, such as strings and numbers. You can't define data quality rules for columns of complex types, such as arrays or structures. For more details about working with profile information, see [Creating and working with AWS Glue DataBrew profile jobs](#).

Validating data quality rules

After a ruleset is defined, you can add it to a profile job for validation. You can define more than one ruleset for a dataset.

For example, one ruleset might contain rules with minimally acceptable criteria. A validation failure for that ruleset might mean that the data isn't acceptable for further use. An example is missing values in key columns of a dataset used for machine learning training. You can use a second ruleset with stricter rules to verify whether the dataset has such good quality that no cleanup is required.

You can apply one or more rulesets defined for a given dataset in a profile job configuration. When the profile job runs, it produces a validation report in addition to the data profile. The validation report is available at the same location as your profile data. As with profile information, you can explore the results in the DataBrew console. In the **Dataset details** view, choose the **Data Quality** tab to view the results. For more details about working with profile information, see [Creating and working with AWS Glue DataBrew profile jobs](#).

Acting on validation results

When a DataBrew profile job completes, DataBrew sends an Amazon CloudWatch event with the details of that job run. If you also configured your job to validate data quality rules, DataBrew sends an event for each validated ruleset. The event contains its result (SUCCEEDED, FAILED, or ERROR) and a link to the detailed data quality validation report. You can then automate further action by invoking **next action** depending on the status of validation. For more information on connecting events to target actions, such as Amazon SNS notification, AWS Lambda function invocations and others, see [Getting started with Amazon EventBridge](#).

Following is an example of a DataBrew Validation Result event:

```
{
  "version": "0",
  "id": "fb27348b-112d-e7c2-560d-85e7c2c09964",
  "detail-type": "DataBrew Ruleset Validation Result",
  "source": "aws.databrew",
  "account": "123456789012",
  "time": "2021-11-18T13:15:46Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "datasetName": "MyDataset",
```

```
"jobName": "MyProfileJob",
"jobRunId": "db_f07954d20d083de0c1fc1eee11498d8635ee5be4ca416af27d33933e91ff4e6e",
"rulesetName": "MyRuleset",
"validationState": "FAILED",
"validationReportLocation": "s3://MyBucket/MyKey/
MyDataset_f07954d20d083de0c1fc1eee11498d8635ee5be4ca416af27d33933e91ff4e6e_dq-
validation-report.json"
}
}
```

You can use attributes of events such as `detail-type`, `source` and nested properties of the `detail` attribute to [create event patterns](#) in Amazon Eventbridge. For example an event pattern to match all failed validations from any DataBrew job would look like this:

```
{
  "source": ["aws.databrew"],
  "detail-type": ["DataBrew Ruleset Validation Result"],
  "detail": {
    "validationState": ["FAILED"]
  }
}
```

For an example of creating a ruleset and validating its rules, see [Creating a ruleset with data quality rules](#). For more information about working with CloudWatch events in DataBrew, see [Automating DataBrew with CloudWatch Events](#)

Creating a ruleset with data quality rules

In the following procedure, you can find an example of creating a ruleset and applying it to a dataset. A *ruleset* is a set of rules that compare different data metrics against expected values. You then can use this ruleset in a profile job to validate the data quality rules that it includes.

To create an example ruleset with data quality rules

1. Sign in to the AWS Management Console and open the DataBrew console at <https://console.aws.amazon.com/databrew/>.
2. Choose **DQ RULES** from the navigation pane, and then choose **Create data quality ruleset**.
3. Enter a name for your ruleset. Optionally, enter a description for your ruleset.
4. Under **Associated dataset**, choose a dataset to associate with the ruleset.

After you select a dataset, you can view the **Dataset preview** pane at right.

5. Use the preview in the **Dataset preview** pane to explore the values and schema for the dataset as you determine the data quality rules to create. The preview can give you insight about potential issues that you might have with the data.

Some data sources, such as databases, don't support data preview. In that case, you can run a profile job without validating the data quality rules first. Then you can get information about the data schema and values distribution by using the data profile.

6. Check the **Recommendations** tab, which lists some rule suggestions that you can use when creating your ruleset. You can select all, some, or none of the recommendations.

After selecting relevant recommendations, choose **Add to ruleset**.

This will add rules to your ruleset. Inspect and modify parameters if needed. Note that only columns of simple types such as *string*, *numbers* and *boolean* can be used in data quality rules.

7. Choose **Add another rule** to add a rule not covered by recommendations. You can change rule names to make it easier to interpret validation results later.
8. Use **Data quality check scope** to choose whether individual columns will be selected per each check in this rule or whether they should be applied to a group of columns you select. For example, if your dataset has several numeric columns that should have values between 0 and 100, you can define the rule once and select all these columns to be checked by this rule.
9. If your rule will have more than one check, then in the **Rule success criteria** dropdown, choose whether all checks should be met or which ones meet the criteria.
10. Select a check that will be performed to verify this rule in the **Data quality check** dropdown. For more information about available checks, see [Available checks](#).
11. If you chose **Individual check for each column** in the **Data quality check scope**, choose a column. Select or type the column name for this check.
12. Select parameters depending on the check. Some conditions accept only provided custom values and some also support reference to another column.
13. If you choose checks for **Column values** such as *Contains* condition for string values, then you can specify "passing" threshold. For example, if you want at least 95 percent of values to satisfy the condition, you need to choose *Greater than equals* as a threshold's **Condition**, enter 95 as a **Threshold** and leave *"%(percent) rows"* in the next dropdown in the **Threshold** section. Or if you want no more than 10 rows where *value is missing* condition is true, then you can select *Less than equals* as a **Condition**, enter 10 for **Threshold** and choose **rows** in the next

dropdown. Please note that you might get different results if you're using samples of different size during validation.

14. Add more rules if needed.
15. Choose **Create ruleset**.

Creating a profile job using a ruleset

After you create a ruleset as described preceding, you are directed to the **Data quality rules** page, which displays all rulesets in your account.

To create a profile job including a ruleset

1. Choose the name of the ruleset that you previously created to view its details.
2. Choose **Create profile job with ruleset**.

The **Job name** is automatically filled, but you can change it as needed.

3. For **Job run sample**, you can choose to run the entire dataset or a limited number of rows.

If you choose to run a limited sample size, be aware that for certain rules, results might differ compared to the full dataset.

4. For **Job output settings**, choose an **S3** location for the job output. Choose any folder in a named Amazon S3 bucket that you have access to. If you enter a folder name for this bucket that doesn't exist, this folder is created.

Upon successful completion of the profile job, this folder will contain profiles of the data and data quality rules validation report in JSON format.

5. Under **Data quality rules**, note your ruleset is listed under **Data quality ruleset name**.
6. Under **Permissions**, select or create a role to grant DataBrew access to read from the input Amazon S3 location and write to the job output location. If you don't have a role ready, select **Create new IAM role**.
7. Modify any other optional settings as described in [Creating and working with AWS Glue DataBrew profile jobs](#), if needed.
8. Choose **Create and run job**.

Inspecting validation results for and updating data quality rules

After your profile job completes, you can view the validation results for your data quality rules and as needed update your rules.

To view validation data for your data quality rules

1. On the DataBrew console, choose **View data profile**. Doing this displays the **Data profile overview** tab for your dataset.
2. Choose the **Data quality rules** tab. On this tab, you can view the results for all of your data quality rules.
3. Select an individual rule for more details about that rule.

For any rule that failed validation, you can make the necessary corrections.

To update your data quality rules

1. On the navigation pane, choose **DQ RULES**.
2. Under **Data quality ruleset name**, choose the dataset that contains the rules that you plan to edit.
3. Choose the rule that you want to change, and then choose **Edit**.
4. Make the necessary corrections, and then choose **Update ruleset**.
5. Rerun the job. Repeat this process until all validations pass.

Available checks

The following table lists references for all available conditions that can be used in your rules. Note that aggregated conditions cannot be combined with non-aggregated conditions in the same rule.

Note

For SDK users, to apply the same rule to multiple columns use the [ColumnSelectors](#) attribute of a [Rule](#) and specify validated columns using either their names or a regular expression. In this case, you should use implicit *CheckExpression*. For example, `"> :val"` to compare values in each of the selected columns with the provided value. DataBrew uses

implicit syntax for defining [FilterExpression](#) in dynamic datasets. If you want to specify column(s) for each check individually, don't set the *ColumnSelectors* attribute. Instead, provide an explicit expression. For example, “:col > :val” as a *CheckExpression* in a *Rule*.

Condition type	Data quality check	Additional parameters	Comparison type	SDK syntax example
Aggregate dataset conditions	Number of rows		Numeric comparison against custom value	"CheckExpression": "AGG(ROWS_COUNT) > :val", "SubstitutionMap": {":val", "10000"}
	Number of columns		Numeric comparison against custom value	"CheckExpression": "AGG(COLUMNS_COUNT) == :val", "SubstitutionMap": {":val", "20"}
	Duplicate rows		Numeric comparison against custom value	"CheckExpression": "AGG(DUPLICATE_ROWS_COUNT) < :val", "SubstitutionMap":

Condition type	Data quality check	Additional parameters	Comparison type	SDK syntax example
				<pre>":{"val", "100"} or "CheckExpression": "AGG(DUPLICATE_ROWS_PERCENTAGE) < :val", "SubstitutionMap": ":{"val", "5"}</pre>

Condition type	Data quality check	Additional parameters	Comparison type	SDK syntax example
Aggregate column statistics conditions	Missing values		Numeric comparison against custom value	<pre>"CheckExpression": "AGG(MISSING_VALUE S_COUNT) < :val", "SubstitutionMap": {":val", "100"} or "CheckExpression": "AGG(MISSING_VALUE S_PERCENT AGE) < :val", "SubstitutionMap": {":val", "5"}</pre>

Condition type	Data quality check	Additional parameters	Comparison type	SDK syntax example
	Duplicate values		Numeric comparison against custom value	<pre>"CheckExpression": "AGG(DUPLICATE_VALUES_COUNT) < :val", "SubstitutionMap": {":val", "100"} or "CheckExpression": "AGG(DUPLICATE_VALUES_PERCENTAGE) < :val", "SubstitutionMap": {":val", "5"}</pre>

Condition type	Data quality check	Additional parameters	Comparison type	SDK syntax example
	Valid values		Numeric comparison against custom value	<pre>"CheckExpression": "AGG(VALID_VALUES_COUNT) > :val", "SubstitutionMap": {":val", "10000"} or "CheckExpression": "AGG(VALID_VALUES_PERCENTAGE) > :val", "SubstitutionMap": {":val", "95"}</pre>

Condition type	Data quality check	Additional parameters	Comparison type	SDK syntax example
	Distinct values		Numeric comparison against custom value	<pre>"CheckExpression": "AGG(DISTINCT_VALUES_COUNT) > :val", "SubstitutionMap": {":val", "1000"} or "CheckExpression": "AGG(DISTINCT_VALUES_PERCENTAGE) >= :val", "SubstitutionMap": {":val", "50"}</pre>

Condition type	Data quality check	Additional parameters	Comparison type	SDK syntax example
	Unique values		Numeric comparison against custom value	<pre>"CheckExpression": "AGG(UNIQUE_VALUES_COUNT) > :val", "SubstitutionMap": {":val", "100"} or "CheckExpression": "AGG(UNIQUE_VALUES_PERCENTAGE) > :val", "SubstitutionMap": {":val", "20"}</pre>

Condition type	Data quality check	Additional parameters	Comparison type	SDK syntax example
	Outliers	Z-score threshold	Numeric comparison against custom value	<pre>"CheckExpression": "AGG(Z_SCORE_OUTLIERS_COUNT , :zscore_dev) < :val", "SubstitutionMap": {":zscore_dev": "4", ":val", "100"} or "CheckExpression": "AGG(Z_SCORE_OUTLIERS_PERCENTAGE) < :val", "SubstitutionMap": {":val", "5"}</pre>

Condition type	Data quality check	Additional parameters	Comparison type	SDK syntax example
	Value distribution statistics	Statistics name (see next table)	Numeric comparison against custom value	<pre>"CheckExpression": "AGG(<STAT_NAME> < :val", "SubstitutionMap": {":val", "100"}</pre> or <pre>"CheckExpression": "AGG(<STAT_NAME>, :param) < :val", "SubstitutionMap": {":param": "0.25", :val", "5"}</pre>  Note See next table for possible STAT_NAME values

Condition type	Data quality check	Additional parameters	Comparison type	SDK syntax example
	Numerical statistics	Statistics name (see next table)	Numeric comparison against custom value	<pre>"CheckExpression": "AGG(<STAT_NAME> < :val", "SubstitutionMap": {":val", "100"} or "CheckExpression": "AGG(<STAT_NAME>, :param) < :val", "SubstitutionMap": {":param": "0.25", :val", "5"}</pre>  Note See next table for possible STAT_NAME values

Condition type	Data quality check	Additional parameters	Comparison type	SDK syntax example
Non aggregate (accepts threshold)	Value is exactly		Exact comparison against a list of values	<pre>"CheckExpression": ":col IN :list", "SubstitutionMap": {":col": "`size`", ":list": "[\"S\", \"M\", \"L\", \"XL\"]"}</pre>
	Value is not exactly		Value shouldn't exactly match any value from a list	<pre>"CheckExpression": ":col NOT IN :list", "SubstitutionMap": {":col": "`domain`", ":list": "[\"GOV\", \"ORG\"]"}</pre>

Condition type	Data quality check	Additional parameters	Comparison type	SDK syntax example
	String values		String comparison against custom value or other string column	<pre>"CheckExpression": ":col STARTS_WITH :val", "SubstitutionMap": {":col": "`url`", ":val": "http"} or "CheckExpression": ":col1 contains :col2", "SubstitutionMap": {":col1": "`url`", ":col2": "`company_name`"}</pre>

Condition type	Data quality check	Additional parameters	Comparison type	SDK syntax example
	Numeric values		Numeric comparison against custom value or other numeric column	<pre>"CheckExpression": ":col IS_BETWEEN :val1 and :val2", "SubstitutionMap": {":col": "`APY`", ":val1": "0", ":val2": "10"} or "CheckExpression": ":col1 <= :col2", "SubstitutionMap": {":col1": "`bank_rate`", ":col2": "`fed_rate`"}</pre>

Condition type	Data quality check	Additional parameters	Comparison type	SDK syntax example
	Value string length		Numeric comparison against custom value or other numeric column	<pre>"CheckExpression": "length(:col) IS_BETWEEN :val1 and :val2", "SubstitutionMap": {":col": "`identifier`", ":val1": "8", ":val2": "12"} or "CheckExpression": "length(:col1) <= :col2", "SubstitutionMap": {":col1": "`name`", ":col2": "`max_name_len`"}</pre>

Numeric comparisons

DataBrew supports the following operations for numeric comparison: *Is equals* (`==`), *Is not equals* (`!=`), *Less than* (`<`), *Less than equals* (`<=`), *Greater than* (`>`), *Greater than equals* (`>=`) and *Is between* (*is_between* :val1 and :val2).

String comparisons

The following string comparisons are supported: *Starts with*, *Doesn't start with*, *Ends with*, *Doesn't end with*, *Contains*, *Doesn't contain*, *Is equals*, *Is not equals*, *Matches*, *Doesn't match*.

The following table displays available statistics that you can use for Value distribution statistics and Numerical statistics:

Data quality check	Statistics name	Additional parameters	SDK syntax
Value distribution statistics	Min		"CheckExpression": "AGG(MAX) < :val", "SubstitutionMap": {":val", "100"}
	Max		"CheckExpression": "AGG(MIN) > :val", "SubstitutionMap": {":val", "0"}
	Median		"CheckExpression": "AGG(MEDIAN) >= :val", "SubstitutionMap": {":val", "50"}

Data quality check	Statistics name	Additional parameters	SDK syntax
	Mean		"CheckExpression": "AGG(MEAN) <= :val", "SubstitutionMap": {":val", "10"}
	Mode		"CheckExpression": "AGG(MODE) > :val", "SubstitutionMap": {":val", "0"}
	Standard deviation		"CheckExpression": "AGG(STANDARD_DEVIATION) > :val", "SubstitutionMap": {":val", "0"}
	Entropy		"CheckExpression": "AGG(ENTROPY) > :val", "SubstitutionMap": {":val", "0"}

Data quality check	Statistics name	Additional parameters	SDK syntax
Numerical statistics	Sum		"CheckExpression": "AGG(SUM) > :val", "SubstitutionMap": {":val", "0"}
	Kurtosis		"CheckExpression": "AGG(KURTOSIS) > :val", "SubstitutionMap": {":val", "0"}
	Skewness		"CheckExpression": "AGG(SKEWNESS) > :val", "SubstitutionMap": {":val", "0"}
	Variance		"CheckExpression": "AGG(VARIANCE) > :val", "SubstitutionMap": {":val", "0"}

Data quality check	Statistics name	Additional parameters	SDK syntax
	Absolute deviation		<code>"CheckExpression": "AGG(MEDIAN_ABSOLUTE_DEVIATION) > :val", "SubstitutionMap": {":val", "0"}</code>
	Quantile	Quantile: one of '0.25', '0.5', '0.75'	<code>"CheckExpression": "AGG(QUANTILE, :pct) > :val", "SubstitutionMap": {":pct": "0.25", ":val", "0"}</code>

Creating and using AWS Glue DataBrew projects

In AWS Glue DataBrew, a *project* is the centerpiece of your data analysis and transformation efforts.

When you create a project, you bring together two fundamental components:

- A dataset, to provide read-only access to your source data. For more information, see [Connecting to data with AWS Glue DataBrew](#).
- A recipe, to apply DataBrew data transformations to the dataset. For more information, see [Creating and using AWS Glue DataBrew recipes](#).

The DataBrew console presents your project in a highly interactive, intuitive user interface. It encourages you to experiment with hundreds of data transformations, so you can learn how they work and what effect they have on your data.

The data that you see in project view is a sample of your dataset. Because datasets can be very large, with thousands or even millions of rows, using a sample helps ensure that the DataBrew console remains responsive while you transform the sample data in various ways. By default, the sample consists of the first 500 rows of data from the dataset. You can choose different settings for the sample size, and which rows are chosen.

As you transform the sample data, DataBrew helps you build and refine the project recipe—a step-by-step series of the transformations that you applied thus far. Your work-in-progress recipe is saved automatically, so you can leave the project view at any time, return later, and pick up where you left off.

When your recipe is ready for use you can publish it. Publishing a recipe makes it available to the DataBrew job subsystem, where you can apply the recipe to your entire dataset, or create an extensive data profile that lets you understand the structure, content, and statistical characteristics of your data.

Topics

- [Creating a project](#)
- [Overview of a DataBrew project session](#)
- [Deleting a project](#)

Creating a project

Use the following procedure to create a project.

To create a project

1. Sign in to the AWS Management Console and open the DataBrew console .
2. On the navigation pane, choose **PROJECTS**. Then choose **Create project**.
3. Enter a name for your project. Then choose a recipe to attach to your project:
 - Choose **Create new recipe** if you are starting from the beginning. Doing this creates a new, empty recipe and attaches it to your project.
 - Choose **Edit existing recipe** if you have a previously published recipe that you want to use for this project. If the recipe is currently attached to another project, or has any jobs defined for it, then you can't use it in your new project. Choose **Browse recipes** to see what recipes are available.
 - Choose **Import steps from recipe** if you have an existing recipe that's been published previously and want to import its steps, and then do the following:
 1. Choose **Browse recipes** to see what recipes are available.
 2. Choose the published version of the recipe that you want to use. A recipe can have multiple versions, depending on how often you published it while working in project view.
 3. Choose **View recipe steps** to examine the data transformations in the recipe.
4. After you have a recipe, choose the dataset that you want to work with on the **Select a dataset** pane:
 - **My datasets** – Choose a dataset that you created previously. For more information, see [Creating a project](#).)
 - **Sample files** – Create a new dataset based on sample data maintained by AWS. This sample data is a great way to explore what DataBrew can do, without having to provide your own data. Make sure to enter a name for your dataset.
 - **New dataset** – Create a new dataset. For more information, see [Creating a project](#).
5. For **Access permissions**, choose an AWS Identity and Access Management (IAM) role that allows DataBrew to read from your Amazon S3 input location. For an S3 location owned by your AWS account, you can choose the `AwsGlueDataBrewDataAccessRole` service-managed role. Doing this allows DataBrew to access S3 resources that you own.

6. On the **Sampling** pane, you can find options for DataBrew to build a sample of data from your dataset.

For **Type**, choose how DataBrew should get rows from your dataset:

- Use **First n rows** to create a sample based on the first rows in the dataset.
- Use **Random rows** to create a sample based on a random selection of rows in the dataset.
- Choose the number of rows to appear in the sample: 500, 1,000, 2,500, or a custom sample size, up to a maximum of 5,000 rows. A smaller sample size allows DataBrew to perform transformations faster, saving you time as you develop your recipe. A larger sample size more accurately reflects the makeup of the underlying source data. However, project session initialization and interactive transformations are slower.

7. (Optional) Choose **Tags** to attach tags to your dataset.

Tags are simple labels consisting of a user-defined key and an optional value that can make it easier to manage, search for, and filter DataBrew projects by purpose, owner, environment, or other criteria.

8. When the settings are as you want them, choose **Create job**.

DataBrew creates a new dataset if needed, creates a new recipe if needed, builds the data sample, and creates an interactive project session. This process can take a couple of minutes to complete. When the project is ready for use, you can begin working with the data sample.

Overview of a DataBrew project session

In a DataBrew project session, you work within an interactive workspace.

The screenshot displays the AWS Glue DataBrew workspace for a project named 'baby-names'. The left sidebar contains navigation icons for DATASETS, PROJECTS, RECIPES, JOBS, and COMMUNITY. The main workspace is divided into two panes. The left pane shows the 'baby-names' dataset in 'Grid' view, displaying a sample of 500 rows. Above the grid, there are summary statistics for the '# count' and 'gender' columns, including unique counts, total counts, and a distribution bar chart for the count column. The right pane shows a 'Recipe (0)' section for 'baby-names-recipe', Version 0.1, which is currently empty. A 'Build your recipe' section prompts the user to start applying transformation steps, with an 'Add step' button. The top of the interface includes a 'Create job' button, 'LINEAGE' and 'ACTIONS' links, and a toolbar with various data manipulation tools like Filter, Column, Format, Clean, Extract, Missing, Invalid, Duplicates, Split, Merge, Create, Functions, and More. The bottom of the grid view has a zoom slider set to 100%.

The left pane shows the current view of your data. The right pane shows the project's transformation recipe, which is currently empty.

In the upper-right corner of the data grid, there are three tabs: GRID, SCHEMA, and PROFILE. Choosing one of these tabs displays a corresponding view in the workspace; these views are described next.

Grid view

Grid view is the default view, where the sample is shown in tabular format. Use the following procedure for a short walkthrough of grid view.

To take a walkthrough of grid view

1. Start by viewing the entire space:

- a. Scroll left and right to see all of the columns.
 - b. Scroll up and down to see all of the data values.
 - c. Use the zoom control at the bottom of the workspace to adjust the magnification level of the grid.
2. At upper-right, view how many of the sample's columns are shown and the current number of rows in the sample.

To change which columns are shown, choose the **N columns** link (where **N** is the number of columns currently displayed). Choose the columns that you want, and choose **Show selected columns**.

3. Now you can start experimenting with DataBrew transformations. Try the following:
 - a. From the transformation toolbar, choose **Choose Format, Change to uppercase**.
 - b. For **Source column**, choose a column that contains character data.
 - c. Leave the other settings at their defaults.
 - d. To see what the transformed data will look like, choose **Preview changes**. Then, to add this transformation to your recipe, choose **Apply**.

Whenever you apply a data transformation, DataBrew adds it to the working copy of your recipe. This appears at the right side of your workspace.

4. Try the following:
 - a. From the transformation toolbar, choose **Create, Based on a function**.
 - b. For **Select a function**, choose **SQUARE ROOT**.
 - c. For **Source column**, choose a column that contains numeric data.
 - d. Leave the other settings at their defaults.
 - e. Choose **Preview changes** to see what the transformed data looks like. Then, to add this transformation to your recipe, choose **Apply**.
5. Collapse the recipe pane at upper right by choosing **RECIPE**. To expand the recipe pane, choose **RECIPE** again.

Publishing a new version of your recipe

As you continue applying transformations, the number of steps in the recipe increases. At any time, you can publish a new version of your recipe. *Publishing* a recipe makes it available elsewhere in DataBrew. By doing this, you can run a recipe job to transform your entire dataset, as opposed to transforming only the project data sample.

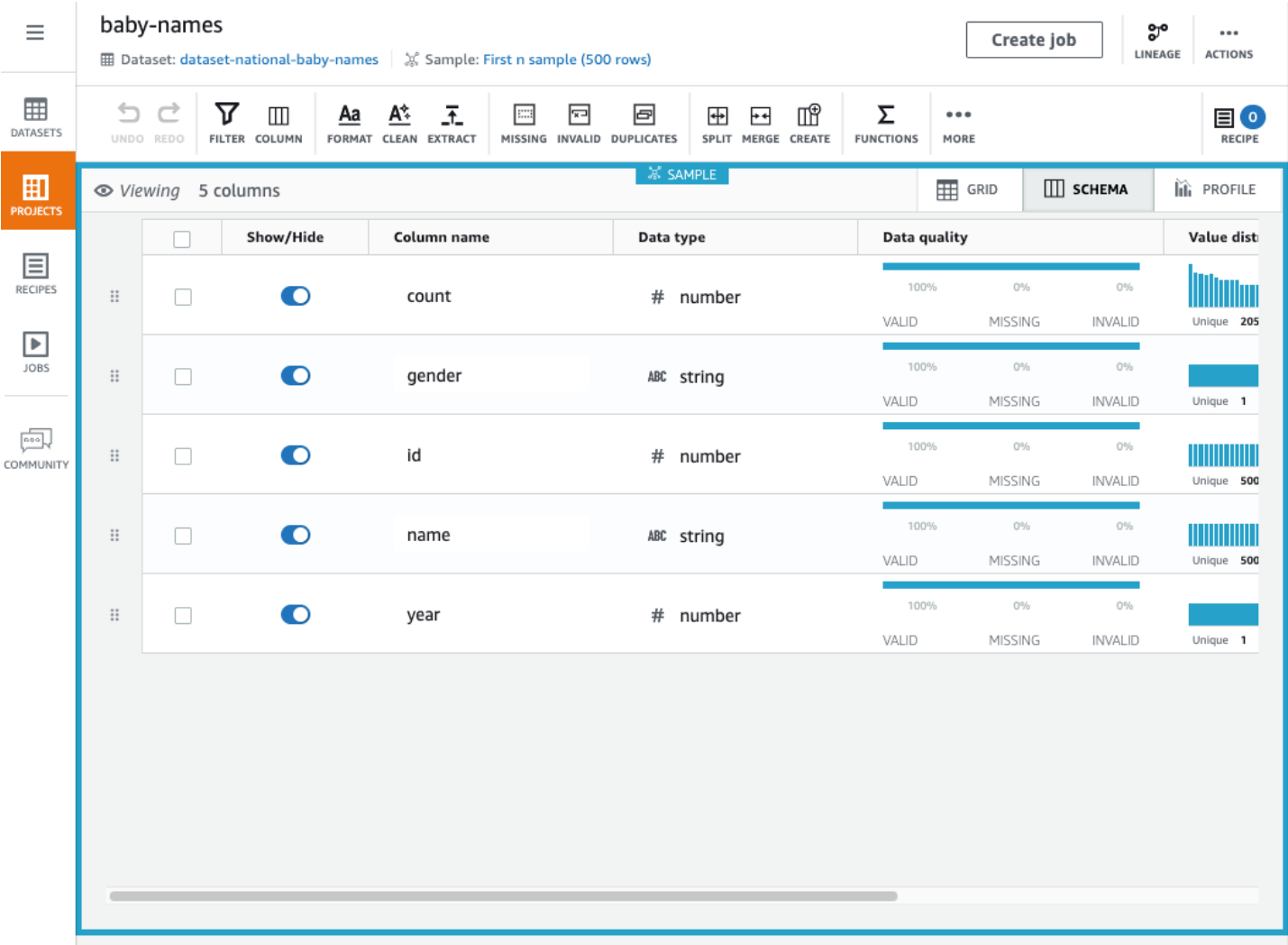
Publishing recipes also encourages an incremental, iterative approach to recipe development: You can publish new versions of your recipe as you go, so you can fall back to a "last known good" recipe version if needed.

To publish a new version of a recipe

- In the recipe pane, choose **Publish**. Enter a description for this version of the recipe, and choose **Publish**.

Schema view

If you choose the **SCHEMA** tab, the view changes, as shown in the screenshot following.



In schema view, you can see statistics about the data values in each column.

In the far left column, next to **Show/Hide**, choose any of the data columns. The **Column details** pane appears at right. This pane shows a summary of statistics for the column values.

You can rename a column by entering a new name for **Column name**.

You can rearrange the column order by dragging and dropping the columns.

Profile view

If you choose the **PROFILE** tab, you can see detailed volumetric information about your project. Before doing so, you run a DataBrew job to create the profile.

To take a walkthrough of profile view

1. Choose **Create job**, and enter a name for your job.
2. For **Job output**, choose **CSV** for the file type.
3. Find or create an Amazon S3 bucket and folder in your AWS account where you want the job output from DataBrew to be written:
 - If you already have this Amazon S3 bucket and folder, choose **Browse** and locate them. Make sure that you have write permissions for both.
 - If you don't have this Amazon S3 bucket and folder, create them:
 1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
 2. If you don't have an Amazon S3 bucket, choose **Create bucket**. For **Bucket name**, enter a unique name for your new bucket. Choose **Create bucket**.
 3. From the list of buckets, choose the one that you want to use.
 4. Choose **Create folder**. For **Folder name**, enter databrew-output, and choose **Create folder**.
4. For **Access permissions**, choose an IAM role that allows DataBrew to write to your Amazon S3 output location.

For an S3 location owned by your AWS account, you can choose the `AwsGlueDataBrewDataAccessRole` service-managed role. Doing this allows DataBrew to access S3 resources that you own.

5. Leave the other settings at their defaults, and choose **Create and run job**.
6. After the job runs to completion, the workspace displays a graphical summary of the data profile.

The **Data profile** overview tab shows a high-level summary of your data's characteristics, as shown in the screenshot following.

Summary

TOTAL ROWS
20,000

TOTAL COLUMNS
5

DATA TYPES

BIG INTEGER ABC STRING
3 columns 2 columns

MISSING CELLS

VALID CELLS MISSING CELLS
100,000 100% 0 0%

Correlations

Correlation coefficient (r) defines how closely two variables are related, ranging from -1.0 to +1.0, where 0 means there is no relationship between the variables.

count

id

The **Column statistics** tab shows a column-by-column breakdown of the data values:

baby-names Create job LINEAGE ACTIONS

Dataset: [dataset-national-baby-names](#) | Sample: First n sample (500 rows)

dataset-national-baby-names (Input) View dataset

S3 dataset-national-baby-names.json 3.8 MB

PROJECTS GRID SCHEMA PROFILE

Data profile overview **Column statistics**

Rerun profile Last job run Succeeded an hour ago, no job runs scheduled Select profile to view Job run 1 | November 10, 2020, 11:30:04 am ▼

Data profile is run on first 20,000 rows of a dataset

Columns (5)

ALL (5) **# BIG INTEGER (3)** **ABC STRING (2)**

count

ABC gender

id

ABC name

year

Big integer | **count**

Data quality

VALID VALUES MISSING VALUES

20000 100% 0 0%

Value distribution

Unique **1,157** Total **20,000**

Data insight

Cardinality

Missing

Correlation

Correlation coefficient related. It represents the relationship between two variables.

TOP

Deleting a project

If you no longer need a project, you can delete it.

To delete a project

1. On the navigation pane, choose **PROJECTS**.
2. Choose the project that you want to delete, and then for **Actions**, choose **Delete**.

Creating and using AWS Glue DataBrew recipes

In DataBrew, a *recipe* is a set of data transformation steps. You can apply these steps to a sample of your data, or apply that same recipe to a dataset.

The easiest way to develop a recipe is to create a DataBrew project, where you can work interactively with a sample of your data—for more information, see [Creating and using AWS Glue DataBrew projects](#). As part of the project creation workflow, a new (empty) recipe is created and attached to the project. You can then start building your recipe by adding data transformations.

Note

You can include up to 100 data transformations in a single DataBrew recipe.

As you proceed with developing your recipe, you can save your work by *publishing* the recipe. DataBrew maintains a list of published versions for your recipe. You can use any published version in a recipe job, to run the recipe (in a recipe job) to transform your dataset. You can also download a copy of the recipe steps, so that you can reuse the recipe in other projects or other dataset transformations.

You can also develop DataBrew recipes programmatically, using the AWS Command Line Interface (AWS CLI) or one of the AWS SDKs. In the DataBrew API, transformations are known as *recipe actions*.

Note

In an interactive DataBrew project session, each data transformation that you apply results in a call to the DataBrew API. These API calls occur automatically, without you having to know the behind-the-scenes details.

Even if you're not a programmer, it's helpful to understand the structure of a recipe and how DataBrew organizes the recipe actions.

Topics

- [Publishing a new recipe version](#)
- [Defining a recipe structure](#)

Publishing a new recipe version

You publish new versions of a recipe in an interactive DataBrew project session.

To publish a new recipe version

1. In the recipe pane, choose **Publish**.
2. Enter a description for this version of the recipe, and choose **Publish**.

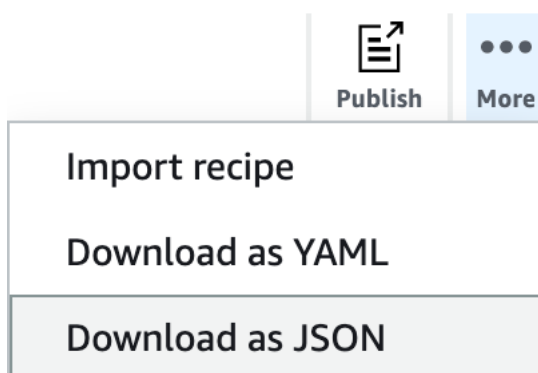
You can view all your published recipes, and their versions, by choosing **PROJECTS** from the navigation pane.

Defining a recipe structure

When you first create a project using the DataBrew console, you define a recipe to be associated with that project. If you don't have an existing recipe, the console creates one for you.

As you work with your project in the console, you use the transformation toolbar to apply actions to the sample data from your dataset. The console shows the recipe steps, and the order of those steps, as you continue building the recipe. You can iterate and refine the recipe until you are satisfied with the steps.

In [Getting started with AWS Glue DataBrew](#), you build a recipe to transform a dataset of famous chess games. You can download a copy of the recipe steps, by choosing **Download as JSON** or **Download as YAML** as shown in the following screenshot.



The downloaded JSON file contains recipe actions corresponding to the transformations that you added to your recipe.

A new recipe doesn't have any steps. You can represent a new recipe as an empty JSON list, as shown following.

```
[ ]
```

Following is an example of such a file, for chess-project-recipe. The JSON list contains several objects that describe the recipe steps. Each object in the JSON list is enclosed in curly braces ({ }). The JSON lines are delimited by commas.

```
[
  {
    "Action": {
      "Operation": "REMOVE_VALUES",
      "Parameters": {
        "sourceColumn": "black_rating"
      }
    },
    "ConditionExpressions": [
      {
        "Condition": "LESS_THAN",
        "Value": "1800",
        "TargetColumn": "black_rating"
      }
    ]
  },
  {
    "Action": {
      "Operation": "REMOVE_VALUES",
      "Parameters": {
        "sourceColumn": "white_rating"
      }
    },
    "ConditionExpressions": [
      {
        "Condition": "LESS_THAN",
        "Value": "1800",
        "TargetColumn": "white_rating"
      }
    ]
  },
  {
    "Action": {
```

```

        "Operation": "GROUP_BY",
        "Parameters": {
            "groupByAggFunctionOptions": "[{\"sourceColumnName\":\"winner\",
            \"targetColumnName\":\"winner_count\", \"targetColumnDataType\":\"int\", \"functionName
            \":\"COUNT\"}]\"",
            "sourceColumns": "[\"winner\", \"victory_status\"]",
            "useNewDataFrame": "true"
        }
    },
    {
        "Action": {
            "Operation": "REMOVE_VALUES",
            "Parameters": {
                "sourceColumn": "winner"
            }
        },
        "ConditionExpressions": [
            {
                "Condition": "IS",
                "Value": "[\"draw\"]",
                "TargetColumn": "winner"
            }
        ]
    },
    {
        "Action": {
            "Operation": "REPLACE_TEXT",
            "Parameters": {
                "pattern": "mate",
                "sourceColumn": "victory_status",
                "value": "checkmate"
            }
        }
    },
    {
        "Action": {
            "Operation": "REPLACE_TEXT",
            "Parameters": {
                "pattern": "resign",
                "sourceColumn": "victory_status",
                "value": "other player resigned"
            }
        }
    }
}

```

```

    },
    {
      "Action": {
        "Operation": "REPLACE_TEXT",
        "Parameters": {
          "pattern": "outoftime",
          "sourceColumn": "victory_status",
          "value": "ran out of time"
        }
      }
    }
  ]

```

It's easier to see each that each action is an individual line if we only add new lines for new actions, as shown following.

```

[
  { "Action": { "Operation": "REMOVE_VALUES", "Parameters": { "sourceColumn":
    "black_rating" } }, "ConditionExpressions": [ { "Condition": "LESS_THAN", "Value":
    "1800", "TargetColumn": "black_rating" } ] },
  { "Action": { "Operation": "REMOVE_VALUES", "Parameters": { "sourceColumn":
    "white_rating" } }, "ConditionExpressions": [ { "Condition": "LESS_THAN", "Value":
    "1800", "TargetColumn": "white_rating" } ] },
  { "Action": { "Operation": "GROUP_BY", "Parameters": { "groupByAggFunctionOptions":
    "[{\"sourceColumnName\":\"winner\",\"targetColumnName\":\"winner_count\",
    \"targetColumnDataType\":\"int\",\"functionName\":\"COUNT\"]", "sourceColumns":
    "[\"winner\",\"victory_status\"]", "useNewDataFrame": "true" } } },
  { "Action": { "Operation": "REMOVE_VALUES", "Parameters": { "sourceColumn":
    "winner" } }, "ConditionExpressions": [ { "Condition": "IS", "Value": "[\"draw\"]",
    "TargetColumn": "winner" } ] },
  { "Action": { "Operation": "REPLACE_TEXT", "Parameters": { "pattern": "mate",
    "sourceColumn": "victory_status", "value": "checkmate" } } },
  { "Action": { "Operation": "REPLACE_TEXT", "Parameters": { "pattern": "resign",
    "sourceColumn": "victory_status", "value": "other player resigned" } } },
  { "Action": { "Operation": "REPLACE_TEXT", "Parameters": { "pattern": "outoftime",
    "sourceColumn": "victory_status", "value": "ran out of time" } } }
]

```

The actions are performed sequentially, in the same order as in the file:

- **REMOVE_VALUES** – To filter out all of the games where a player's rating is less than 1,800, the minimum rating required to be a Class A chess player. There are two occurrences of this action

—one to remove players on the black side who aren't at least Class A players, and another to remove players on the white side who aren't at this level.

- **GROUP_BY** – To summarize the data. In this case, **GROUP_BY** sorts the rows into groups based on the values of **winner** (black and white). Each of those groups is then broken down further, sorting the rows into subgroups based on the values of **victory_status** (mate, resign, outoftime, and draw). Finally, the number of occurrences for each subgroup is counted. The resulting summary then replaces the original data sample.
- **REMOVE_VALUES** – To delete the results of games that ended with draw.
- **REPLACE_TEXT** – To modify the values for **victory_status**. There are three occurrences of this action—one each for mate, resign, and oufoftime.

In an interactive DataBrew project session, each **RecipeAction** corresponds to a data transformation that you apply to a data sample.

DataBrew provides over 200 recipe actions. For more information, see [Recipe step and function reference](#).

Using conditions

You can use *conditions* to narrow the scope of a recipe action. Conditions are used in transformations that filter the data—for example, removing unwanted rows based on a particular column value.

Let's take a closer look at a recipe actions from **chess-project-recipe**.

```
{
  "Action": {
    "Operation": "REMOVE_VALUES",
    "Parameters": {
      "sourceColumn": "black_rating"
    }
  },
  "ConditionExpressions": [
    {
      "Condition": "LESS_THAN",
      "Value": "1800",
      "TargetColumn": "black_rating"
    }
  ]
}
```

```
}
```

This transformation reads the values in the `black_rating` column. The `ConditionExpressions` list determines the filtering criteria: Any row that has a `black_rating` value of less than 1,800 is removed from the dataset.

A follow-up transformation in the recipe does the same thing, for `white_rating`. In this way, the data is limited to games where each player (black or white) is rated at Class A or above.

Here's another example of a condition, applied to a column of character data.

```
{
  "Action": {
    "Operation": "REMOVE_VALUES",
    "Parameters": {
      "sourceColumn": "winner"
    }
  },
  "ConditionExpressions": [
    {
      "Condition": "IS",
      "Value": "[\"draw\"]",
      "TargetColumn": "winner"
    }
  ]
}
```

This transformation reads the values in the `winner` column, looking for the value `draw` and removing those rows. In this way, the data is limited to only those games where there was a clear winner.

DataBrew supports the following conditions:

- **IS** – The value in the column is the same as the value that was provided in the condition.
- **IS_NOT** – The value in the column isn't the same as the value that was provided in the condition.
- **IS_BETWEEN** – The value in the column is between the `GREATER_THAN_EQUAL` and `LESS_THAN_EQUAL` parameters.
- **CONTAINS** – The string value in the column contains the value that was provided in the condition.

- NOT_CONTAINS – The value in the column does not contain the character string that was provided in the condition.
- STARTS_WITH – The value in the column starts with the character string that was provided in the condition.
- NOT_STARTS_WITH – The value in the column doesn't start with the character string that was provided in the condition.
- ENDS_WITH – The value in the column ends with the character string that was provided in the condition.
- NOT_ENDS_WITH – The value in the column doesn't end with the character string that was provided in the condition.
- LESS_THAN – The value in the column is less than the value that was provided in the condition.
- LESS_THAN_EQUAL – The value in the column is less than or equal to the value that was provided in the condition.
- GREATER_THAN – The value in the column is greater than value that was provided in the condition.
- GREATER_THAN_EQUAL – The value in the column is greater than or equal to the value that was provided in the condition.
- IS_INVALID – The value in the column has an incorrect data type.
- IS_MISSING – There is no value in the column.

Creating, running, and scheduling AWS Glue DataBrew jobs

AWS Glue DataBrew has a job subsystem that serves two purposes:

1. Applying a data transformation recipe to a DataBrew dataset. You do this with a DataBrew recipe job.
2. Analyzing a dataset to create a comprehensive profile of the data. You do this with a DataBrew profile job.

Topics

- [Creating and working with AWS Glue DataBrew recipe jobs](#)
- [Creating and working with AWS Glue DataBrew profile jobs](#)

Creating and working with AWS Glue DataBrew recipe jobs

Use a DataBrew *recipe job* to clean and normalize the data in a DataBrew dataset and write the result to an output location of your choice. Running a recipe job doesn't affect the dataset or the underlying source data. When a job runs, it connects to the source data in a read-only fashion. The job output is written to an output location that you define in Amazon S3, the AWS Glue Data Catalog, or a supported JDBC database.

Use the following procedure to create a DataBrew recipe job.

To create a recipe job

1. Sign in to the AWS Management Console and open the DataBrew console at <https://console.aws.amazon.com/databrew/>.
2. Choose **JOBS** from the navigation pane, choose the **Recipe jobs** tab, and then choose **Create job**.
3. Enter a name for your job, and then choose **Create a recipe job**.
4. For **Job input**, enter details on the job that you want to create: the name of the dataset to be processed, and the recipe to use.

A recipe job uses a DataBrew recipe to transform a dataset. To use a recipe, make sure to publish it first.

5. Configure your job output settings.

Provide a destination for your job output. If you don't have a DataBrew connection configured for your output destination, configure it first on the **DATASETS** tab as described in [Supported connections for data sources and outputs](#). Choose one of the following output destinations:

- Amazon S3, with or without AWS Glue Data Catalog support
- Amazon Redshift, with or without AWS Glue Data Catalog support
- JDBC
- Snowflake tables
- Amazon RDS database tables with AWS Glue Data Catalog support. Amazon RDS database tables support the following database engines:
 - Amazon Aurora
 - MySQL
 - Oracle
 - PostgreSQL
 - Microsoft SQL Server
- Amazon S3 with AWS Glue Data Catalog support.

For AWS Glue Data Catalog output based on AWS Lake Formation, DataBrew supports only replacing existing files. In this approach, the files are replaced to keep your existing Lake Formation permissions intact for your data access role. Also, DataBrew gives precedence to the Amazon S3 location from the AWS Glue Data Catalog table. Thus, you can't override the Amazon S3 location when creating a recipe job.

In some cases, the Amazon S3 location in the job output differs from the Amazon S3 location in the Data Catalog table. In these cases, DataBrew updates the job definition automatically with the Amazon S3 location from the catalog table. It does this when you update or start your existing jobs.

6. For Amazon S3 output destinations only, you have further choices:

- a. Choose one of the available data output formats for Amazon S3, optional compression, and an optional custom delimiter. Supported delimiters for output files are the same as those for input: comma, colon, semicolon, pipe, tab, caret, backslash, and space. For formatting details, see the following table.

Format	File extension (uncompressed)	File extensions (compressed)
Comma-separated values	.csv	.csv.snappy , .csv.gz, .csv.lz4, csv.bz2, .csv.deflate , csv.br
Tab-separated values	.csv	.tsv.snappy , .tsv.gz, .tsv.lz4, tsv.bz2, .tsv.deflate , tsv.br
Apache Parquet	.parquet	.parquet.snappy , .parquet.gz , .parquet.lz4 , .parquet.lzo , .parquet.br
AWS Glue Parquet	Not supported	.glue.parquet.snappy
Apache Avro	.avro	.avro.snappy , .avro.gz, .avro.lz4 , .avro.bz2 , .avro.deflate , .avro.br
Apache ORC	.orc	.orc.snappy , .orc.lzo, .orc.zlib
XML	.xml	.xml.snappy , .xml.gz, .xml.lz4, .xml.bz2, .xml.deflate , .xml.br

Format	File extension (uncompressed)	File extensions (compressed)
JSON (JSON Lines format only)	.json	.json.snappy , .json.gz, .json.lz4 , json.bz2, .json.deflate , .json.br
Tableau Hyper	Not supported	Not applicable

b.

Choose whether to output a single file or multiple files. There are three options for file output with Amazon S3:

- **Autogenerate files (recommended)** – Has DataBrew determine the optimal number of output files.
- **Single file output** – Causes a single output file to be generated. This option might result in additional job execution time because post-processing is required.
- **Multiple file output** – Has you specify the number of files for your job output. Valid values are 2–999. Fewer files than you specify might be output if column partitioning is used or if the number of rows in the output is fewer than the number of files you specify.

c.

(Optional) Choose column partitioning for recipe job output.

Column partitioning provides another way to partition your recipe job output into multiple files. Column partitioning can be used with new or existing Amazon S3 output or with new Data Catalog Amazon S3 output. It cannot be used with existing Data Catalog Amazon S3 tables. The output files are based on the values of column names that you specify. If the column names you specify are unique, the resulting Amazon S3 folder paths are based on the order of the column names.

For an example of column partitioning, see [Example of column partitioning](#), following.

7. (Optional) Choose **Enable encryption for job output** to encrypt the job output that DataBrew writes to your output location, and then choose the encryption method:
 - **Use SSE-S3 encryption** – The output is encrypted using server-side encryption with Amazon S3–managed encryption keys.

- **Use AWS Key Management Service (AWS KMS)** – The output is encrypted using AWS KMS. To use this option, choose the Amazon Resource Name (ARN) of the AWS KMS key that you want to use. If you don't have an AWS KMS key, you can create one by choosing **Create an AWS KMS key**.
8. For **Access permissions**, choose an AWS Identity and Access Management (IAM) role that allows DataBrew to write to your output location. For a location owned by your AWS account, you can choose the `AwsGlueDataBrewDataAccessRole` service-managed role. Doing this allows DataBrew to access AWS resources that you own.
 9. On the **Advanced job settings** pane, you can choose more options for how your job is to run:
 - **Maximum number of units** – DataBrew processes jobs using multiple compute nodes, running in parallel. The default number of nodes is 5. The maximum number of nodes is 149.
 - **Job timeout** – If a job takes more than the number of minutes that you set here to run, it fails with a timeout error. The default value is 2,880 minutes, or 48 hours.
 - **Number of retries** – If a job fails while running, DataBrew can try to run it again. By default, the job isn't retried.
 - **Enable Amazon CloudWatch Logs for job** – Allows DataBrew to publish diagnostic information to CloudWatch Logs. These logs can be useful for troubleshooting purposes, or for more details on how the job is processed.
 10. For **Schedule jobs**, you can apply a DataBrew job schedule so that your job runs at a particular time, or on a recurring basis. For more information, see [Automating job runs with a schedule](#).
 11. When the settings are as you want them, choose **Create job**. Or, if you want to run the job immediately, choose **Create and run job**.

You can monitor your job's progress by checking its status while the job is running. When the job run is complete, the status changes to **Succeeded**. The job output is now available at your chosen output location.

DataBrew saves your job definition, so that you can run the same job later. To rerun a job, choose **Jobs** from the navigation pane. Choose the job that you want to work with, and then choose **Run job**.

Example of column partitioning

As an example of column partitioning, assume that you specify three columns, each row of which contains one of two possible values. The Dept column can have the value Admin or Eng. The Staff-type column can have the value Part-time or Full-time. The Location column can have the value Office1 or Office2. The Amazon S3 buckets for your job output look something like the following.

```
s3://bucket/output-folder/Dept=Admin/Staff-type=Part-time/Location=Office1/
jobId_timestamp_part0001.csv
s3://bucket/output-folder/Dept=Admin/Staff-type=Part-time/Location=Office2/
jobId_timestamp_part0002.csv
s3://bucket/output-folder/Dept=Admin/Staff-type=Full-time/Location=Office1/
jobId_timestamp_part0003.csv
s3://bucket/output-folder/Dept=Admin/Staff-type=Full-time/Location=Office2/
jobId_timestamp_part0004.csv
s3://bucket/output-folder/Dept=Eng/Staff-type=Part-time/Location=Office1/
jobId_timestamp_part0005.csv
s3://bucket/output-folder/Dept=Eng/Staff-type=Part-time/Location=Office2/
jobId_timestamp_part0006.csv
s3://bucket/output-folder/Dept=Eng/Staff-type=Full-time/Location=Office1/
jobId_timestamp_part0007.csv
s3://bucket/output-folder/Dept=Eng/Staff-type=Full-time/Location=Office2/
jobId_timestamp_part0008.csv
```

Automating job runs with a schedule

You can rerun DataBrew jobs at any time and also automate DataBrew job runs with a schedule.

To rerun a DataBrew job

1. Sign in to the AWS Management Console and open the DataBrew console at <https://console.aws.amazon.com/databrew/>.
2. On the navigation pane, choose **Jobs**. Choose the job that you want to run, and then choose **Run job**.

To run a DataBrew job at a particular time, or on a recurring basis, create a DataBrew job schedule. You can then set up your job to run according to the schedule.

To create a DataBrew job schedule

1. On the DataBrew console's navigation pane, choose **Jobs**. Choose the **Schedules** tab, and choose **Add schedule**.
2. Enter a name for your schedule, and then choose a value for **Run frequency**:
 - **Recurring** – Choose how frequently that you want the job to run (for example, every 12 hours). Then choose which day or days to run the job on. Optionally, you can enter the time of day when the job runs.
 - **At a particular time** – Enter the time of day when you want the job to run. Then choose which day or days to run the job on.
 - **Enter CRON** – Define the job schedule by entering a valid cron expression. For more information, see [Working with cron expressions for recipe jobs](#).
3. When the settings are as you want them, choose **Save**.

To associate a job with a schedule

1. On the navigation pane, choose **Jobs**.
2. Choose the job that you want to work with, and then for **Actions**, choose **Edit**.
3. On the **Schedule jobs** pane, choose **Associate schedule**. Choose the name of the schedule that you want to use.
4. When the settings are as you want them, choose **Save**.

Working with cron expressions for recipe jobs

Cron expressions have six required fields, which are separated by white space. The syntax is as follows.

Minutes Hours Day-of-month Month Day-of-week Year

In the preceding syntax, the following values and wildcards are used for the indicated fields.

Fields	Values	Wildcards
Minutes	0–59	, - * /

Fields	Values	Wildcards
Hours	0–23	, - * /
Day-of-month	1–31	, - * ? / L W
Month	1–12 or JAN-DEC	, - * /
Day-of-week	1–7 or SUN-SAT	, - * ? / L
Year	1970–2199	, - * /

Use these wildcards as follows:

- The , (comma) wildcard includes additional values. In the Month field, JAN, FEB, MAR includes January, February, and March.
- The - (en dash) wildcard specifies ranges. In the Day field, 1–15 includes days 1 through 15 of the specified month.
- The * (asterisk) wildcard includes all values in the field. In the Hours field, * includes every hour.
- The / (slash) wildcard specifies increments. In the Minutes field, you can enter **1/10** to specify every 10th minute, starting from the first minute of the hour (for example, the 11th, 21st, and 31st minute).
- The ? (question mark) wildcard specifies one or another. For example, suppose that in the Day-of-month field you enter **7**. If you didn't care what day of the week the seventh was, you can then enter **?** in the Day-of-week field.
- The **L** wildcard in the Day-of-month or Day-of-week field specifies the last day of the month or week.
- The **W** wildcard in the Day-of-month field specifies a weekday. In the Day-of-month field, **3W** specifies the day closest to the third weekday of the month.

These fields and values have the following limitations:

- You can't specify the Day-of-month and Day-of-week fields in the same cron expression. If you specify a value in one of the fields, you must use a ? (question mark) in the other.
- Cron expressions that lead to rates faster than 5 minutes aren't supported.

When creating a schedule, you can use the following sample cron strings.

Minutes	Hours	Day of month	Month	Day of week	Year	Meaning
0	10	*	*	?	*	Run at 10:00 AM (UTC) every day
15	12	*	*	?	*	Run at 12:15 PM (UTC) every day
0	18	?	*	MON-FRI	*	Run at 6:00 PM (UTC) every Monday through Friday
0	8	1	*	?	*	Run at 8:00 AM (UTC) every first day of the month
0/15	*	*	*	?	*	Run every 15 minutes
0/10	*	?	*	MON-FRI	*	Run every 10 minutes Monday through Friday

Minutes	Hours	Day of month	Month	Day of week	Year	Meaning
0/5	8–17	?	*	MON-FRI	*	Run every 5 minutes Monday through Friday between 8:00 AM and 5:55 PM (UTC)

For example, you can use the following cron expression to run a job every day at 12:15 UTC.

```
15 12 * * ? *
```

Deleting jobs and job schedules

If you no longer need a job or job schedule, you can delete it.

To delete a job

1. On the navigation pane, choose **Jobs**.
2. Choose the job that you want to delete, and then for **Actions**, choose **Delete..**

To delete a job schedule

1. On the navigation pane, choose **Jobs**, and then choose the **Schedules** tab.
2. Choose the schedule that you want to delete, and then for **Actions**, choose **Delete..**

Creating and working with AWS Glue DataBrew profile jobs

Profile jobs run a series of evaluations on a dataset and output the results to Amazon S3. The information that data profiling gathers helps you understand your dataset and decide what kind of data preparation steps you might want to run in your recipe jobs.

The simplest way to run a profile job is using the default DataBrew settings. You can configure your profile job before running it so that it returns just the information that you want.

Use the following procedure to create a DataBrew profile job.

To create a profile job

1. Sign in to the AWS Management Console and open the DataBrew console at <https://console.aws.amazon.com/databrew/>.
2. Choose **JOBS** from the navigation pane, choose the **Profile jobs** tab, and then choose **Create job**.
3. Enter a name for your job, and then choose **Create a profile job**.
4. For **Job input**, provide the name of the dataset to be profiled.
5. (Optional) Configure the following on the **Data profile configurations** pane:

- **Dataset level configurations** – Configure details of your profile job for all columns in your dataset.

Optionally, you can turn on the ability to detect and count duplicate rows in the dataset. You can also choose **Enable correlations matrix** and select columns to see how closely the values in multiple columns are related. For details of the statistics that you can configure at the dataset level, see [Configurable statistics at the dataset level](#). You can configure statistics on the DataBrew console, or using the DataBrew API or AWS SDKs.

- **Column level configurations** – Using **Default profile configuration settings**, you can select the columns to include in your profile job. Use **Add configuration override** to select the columns for which to limit the number of statistics gathered, or override the default configuration of certain statistics. For details of the statistics that you can configure at the column level, see [Configurable statistics at the column level](#). You can configure statistics on the DataBrew console, or using the DataBrew API or AWS SDKs.

Be sure that any configuration overrides that you specify apply to columns that you included in your profile job. If there are conflicts between different overrides that you configured for a column, the last conflicting override has priority.

6. (Optional) You can create **Data quality rules** and apply additional rulesets associated with this dataset or remove already applied ones. For more information on data quality validation, see [Validating data quality in AWS Glue DataBrew](#).
7. On the **Advanced job settings** pane, you can choose more options for how your job is to run:

- **Maximum number of units** – DataBrew processes jobs using multiple compute nodes, running in parallel. The default number of nodes is 5. The maximum number of nodes is 149.
 - **Job timeout** – If a job takes more than the number of minutes that you set here to run, it fails with a timeout error. The default value is 2,880 minutes, or 48 hours.
 - **Number of retries** – If a job fails while running, DataBrew can try to run it again. By default, the job isn't retried.
 - **Enable Amazon CloudWatch Logs for job** – Allows DataBrew to publish diagnostic information to CloudWatch Logs. These logs can be useful for troubleshooting purposes, or for more details on how the job is processed.
8. For **Associated Schedule**, you can apply a DataBrew job schedule so that your job runs at a particular time, or on a recurring basis. For more information, see [Automating job runs with a schedule](#).
 9. When the settings are as you want them, choose **Create job**. Or, if you want to run the job immediately, choose **Create and run job**.

Building a profile job configuration programmatically in AWS Glue DataBrew

In this section, you can find descriptions of profile job steps and functions that you can use programmatically. You can use them either from the AWS Command Line Interface (AWS CLI) or by using one of the AWS SDKs.

In a profile job, you can customize a configuration to control how DataBrew evaluates your dataset. You can apply the configuration to a dataset or apply it to particular columns. You can build the configuration when creating a profile job, and then update it anytime.

A profile configuration structure includes four parts:

- [ProfileColumns section](#)
- [DatasetStatisticsConfiguration section](#)
- [ColumnStatisticsConfigurations section](#)
- [EntityDetectorConfiguration section for configuring PII](#)

Following is an example.

```

{
  "ProfileColumns": [
    {
      "Name": "example"
    },
    {
      "Regex": "example.*"
    }
  ],
  "DatasetStatisticsConfiguration": {
    "IncludedStatistics": [
      "CORRELATION"
    ],
    "Overrides": [
      {
        "Statistic": "CORRELATION",
        "Parameters": {
          "columnSelectors": "[{\"name\":\"example\"}, {\"regex\":\"example.*"
        }
      }
    ]
  },
  "ColumnStatisticsConfigurations": [
    {
      "Selectors": [
        {
          "Name": "example"
        }
      ],
      "Statistics": {
        "IncludedStatistics": [
          "CORRELATION",
          "DUPLICATE_ROWS_COUNT"
        ],
        "Overrides": [
          {
            "Statistic": "VALUE_DISTRIBUTION",
            "Parameters": {
              "binNumber": "10"
            }
          }
        ]
      }
    }
  ]
}

```

```

    }
  }
]
}

```

ProfileColumns section

In the ProfileColumns section of your structure, set the columns from your dataset that you want to evaluate in your profile job. ProfileColumns is a list of column selectors (Selectors). You can specify either a column name or a regular expression in a column selector. An example follows.

```
"ProfileColumns": [{"Name": "example"}, {"Regex": "example.*"}]
```

When ProfileColumns is specified, only columns whose names match a name or regular expression in ProfileColumns are included in the profile job. If the profile job doesn't support a selected column's data type, DataBrew skips the selected column during the job run.

If ProfileColumns is undefined, the profile job evaluates all supported columns. Supported columns are columns containing data of a supported data type: ByteType, ShortType, IntegerType, LongType, FloatType, DoubleType, String, or Boolean.

DatasetStatisticsConfiguration section

In the DatasetStatisticsConfiguration section of your structure, you can build a configuration for intercolumn evaluations. The configuration includes IncludedStatistics and Overrides. An example follows.

```

"DatasetStatisticsConfiguration": {
  "IncludedStatistics": ["CORRELATION"],
  "Overrides": [
    {
      "Statistic": "CORRELATION",
      "Parameters": {
        "columnSelectors": "[{"name\":\"example\"}, {"regex\":\"example.*
\"}]\"
      }
    }
  ]
}

```

```
}

```

You can select evaluations that you want to have by adding evaluation names to `IncludedStatistics`. An example follows.

```
"IncludedStatistics": ["CORRELATION", "DUPLICATE_ROWS_COUNT"]

```

When you specify `IncludedStatistics`, only evaluations in the list are included in the profile job. If `IncludedStatistics` is undefined, the profile job runs all supported evaluations with default settings. You can exclude all evaluations by adding `NONE` to `IncludedStatistics`. An example follows.

```
"IncludedStatistics": ["NONE"]

```

Configurable statistics at the dataset level

In the `DatasetStatisticsConfiguration` section of your structure, a profile job supports the evaluations shown in the table following.

Statistic name	Description	Supported data types	Default status	Attributes of profile result	Type of profile result
DUPLICATE_ROWS_COUNT	Count of duplicate rows in the dataset	all	Enable	duplicate RowCount	Int
CORRELATION	Pearson Correlation Coefficient between two columns	number	Enable	correlations (in each selected column)	Object

In `IncludedStatistics`, you can override each evaluation's default settings by adding an override. Each override includes the name of a particular evaluation and a parameter map.

In `DatasetStatisticsConfiguration`, a profile job supports the `CORRELATION` override. This override calculates the Pearson Correlation Coefficient between two columns from a list of selected columns. The default setting is selecting the first 10 numeric columns. You can specify either a number of columns or a list of column selectors to override the default setting.

`CORRELATION` takes these parameters:

- `columnNumber` – The number of numeric columns. The profile job selects the first n columns from the dataset. This value should be greater than 1. Use "ALL" to select all numeric columns.
- `columnSelectors`: – List of column selectors. Each selector can have either a column name or a regular expression.

An example follows.

```
{
  "Statistic": "CORRELATION",
  "Parameters": {
    "columnSelectors": "[{\"name\":\"example\"}, {\"regex\":\"example.*\"}]"
  }
}
```

ColumnStatisticsConfigurations section

In the `ColumnStatisticsConfigurations` section of your structure, you can build configurations for particular columns. `ColumnStatisticsConfigurations` is a list of `ColumnStatisticsConfiguration` settings. In `ColumnStatisticsConfiguration`, there are `Selectors`, a list of column selectors, and `Statistics` for the configuration of statistics. An example follows.

```
{
  "Selectors": [{"Name": "example"}],
  "Statistics": {
    "IncludedStatistics": ["CORRELATION", "DUPLICATE_ROWS_COUNT"],
    "Overrides": [
      {
        "Statistic": "VALUE_DISTRIBUTION",
        "Parameters": {
          "binNumber": "10"
        }
      }
    ]
  }
}
```

```
    }  
  ]  
}  
}
```

`Selectors` is a list of column selectors. As with `ProfileColumns`, you can specify either a column name or a regular expression in each column selector. When you specify `Selectors`, the column configuration is applied to columns that match any column selector in `Selectors`. Otherwise, the configuration is applied to all supported columns.

In `Statistics`, you can override settings of selected columns. As with `DatasetStatisticsConfiguration`, `Statistics` has `IncludedStatistics` and `Overrides`.

To select the evaluations that you want, add evaluation names to `IncludedStatistics`.

```
"IncludedStatistics": ["CORRELATION", "DUPLICATE_ROWS_COUNT"]
```

When you specify `IncludedStatistics`, only evaluations in the list are included in the profile job. Otherwise, the profile job runs all supported evaluations with default settings.

You can exclude all evaluations by adding `NONE` to `IncludedStatistics`.

```
"IncludedStatistics": ["NONE"]
```

In some cases, there might be multiple configurations in `ColumnStatisticsConfigurations` that have different `IncludedStatistics` that you can apply to the same column. In these cases, the profile job picks the last configuration in `ColumnStatisticsConfigurations` and applies its `IncludedStatistics` to the selected column. A new configuration overrides older configurations.

Configurable statistics at the column level

In `ColumnStatisticsConfigurations`, a profile job supports the evaluations shown in the table following.

A supported data type of `number` in this table means that the attribute's data type is one of the following: `ByteType`, `ShortType`, `IntegerType`, `LongType`, `FloatType`, or `DoubleType`.

Statistic name	Description	Supported data types	Default status	Attributes of profile result	Type of profile result
–	Name of the column.	all	–	name	string
–	Data type of the column.	all	–	type	string
DISTINCT_VALUES_COUNT	Number of distinct values. A <i>distinct value</i> is value that appears at least once.	number/boolean/string	Enabled	distinctValuesCount	Int
ENTROPY	Entropy (information theory).	number/boolean/string	Enabled	entropy	Double
INTER_QUANTILE_RANGE	Range between the 25th percent and 75th percent of numbers.	number	Enabled	interquartileRange	Double
KURTOSIS	Kurtosis of the column.	number	Enabled	kurtosis	Double
MAX	Maximum value in the column.	number/string length	Enabled	max	Int/Double
MAXIMUM_VALUES	List of the maximum values in the column and their counts.	number	Enabled	maximumValues	List

Statistic name	Description	Supported data types	Default status	Attributes of profile result	Type of profile result
MEAN	Mean value of values in the column.	number/string length	Enabled	mean	Double
MEDIAN	Median of values in the column.	number/string length	Enabled	median	Double
MEDIAN_ABSOLUTE_DEVIATION	The median of the absolute differences between each data point and the median of a numeric column.	number	Enabled	medianAbsoluteDeviation	Double
MIN	Minimum value in the column.	number/string length	Enabled	min	Int/Double
MINIMUM_VALUES	List of the minimum values in the column and their counts.	number	Enabled	minimumValues	List
MISSING_VALUES_COUNT	Number of missing values in the column. Null and empty strings are considered as missing.	all	Enabled	missingValuesCount	Int

Statistic name	Description	Supported data types	Default status	Attributes of profile result	Type of profile result
MODE	The most frequently occurring value in the column. If several values appear that often, the mode is one of those values.	number/string length	Enabled	mode	Int/Double
MOST_COMMON_VALUES	List of the most common values in the column.	number/boolean/string	Enabled	mostCommonValues	List
OUTLIER_DETECTION	Detect outliers in the column by Z_score algorithm. Count the number of outliers and extract a list of samples from detected outliers.	number/string length	Enabled	zScoreOutliersCount, zScoreOutliersSample	Int/List
PERCENTILES	Percentile values of numeric column (5%, 25%, 75%, 95%).	number	Enabled	percentile5, percentile25, percentile75, percentile95	Double
RANGE	Range of values in the column.	number	Enabled	range	Int/Double

Statistic name	Description	Supported data types	Default status	Attributes of profile result	Type of profile result
SKEWNESS	Skewness of values in the column.	number	Enabled	skewness	Double
STANDARD_DEVIATION	Unbiased sample standard deviation of values in the column.	number/string length	Enabled	standardDeviation	Double
SUM	Sum of values in the column.	number	Enabled	sum	Int/Double
UNIQUE_VALUES_COUNT	Number of unique values. A unique value means that the value appears only once.	number/boolean/string	Enabled	uniqueValuesCount	Int
VALUE_DISTRIBUTION	Measure of the distribution of values in the column by range.	number/string length	Enabled	valueDistribution	List
VARIANCE	Variance of values in the column.	number	Enabled	variance	Double
Z_SCORE_DISTRIBUTION	Measure of the distribution of data points' z-score values by range.	number	Enabled	zScoreDistribution	List
ZEROS_COUNT	Number of zeroes (0s) in the column.	number	Enabled	zerosCount	Int

In `IncludedStatistics`, you can override each evaluation's default parameters by adding an override. Each override includes the name of a particular evaluation and a parameter map.

Parameters for `ColumnStatisticsConfigurations` columns

In `ColumnStatisticsConfigurations`, a profile job supports the following parameters.

In some cases, there might be multiple configurations in `ColumnStatisticsConfigurations` that have different `IncludedStatistics` that you can apply to the same column. In these cases, the profile job picks the last configuration in `ColumnStatisticsConfigurations` and applies its `IncludedStatistics` to the selected column. A new configuration overrides older configurations.

MAXIMUM_VALUES

Lists the maximum values in the numeric column and their counts. The default list size is 5. You can override the list size by specifying a value for `sampleSize`.

Settings

`sampleSize` – The size of list that includes the maximum number and count of values in the numeric column. This value should be greater than 0. Use "ALL" to list all values.

Example

```
{
  "Statistic": "MAXIMUM_VALUES",
  "Parameters": {
    "sampleSize": "5"
  }
}
```

MINIMUM_VALUES

Lists the minimum values in the numeric column and their counts. The default list size is 5. You can override the list size by specifying a value for `sampleSize`.

Settings

`sampleSize` – The size of list that includes the maximum number and count of values in the numeric column. This value should be greater than 0. Use "ALL" to list all values.

Example

```
{
  "Statistic": "MINIMUM_VALUES",
  "Parameters": {
    "sampleSize": "5"
  }
}
```

MOST_COMMON_VALUES

Lists the most common values in the column and their counts. The default list size is 50. You can override the list size by specifying a value for `sampleSize`.

Settings

`sampleSize` – The size of list that includes the maximum number and count of values in the numeric column. This value should be greater than 0. Use "ALL" to list all values.

Example

```
{
  "Statistic": "MOST_COMMON_VALUES",
  "Parameters": {
    "sampleSize": "50"
  }
}
```

OUTLIER_DETECTION

Detects outliers in the numeric column or string column (based on string length) by Z_score algorithm.

Your profile job counts the number of outliers and generates a sample list of outliers and their z-scores. The sample list is ordered by the z-score's absolute value. The default list size is 50.

The Z_Score algorithm identifies a value as an outlier when it deviates from the mean by more than the standard deviation threshold. The default outlier threshold is 3.

You can provide one more threshold, a mild threshold, to get more information. Your mild threshold should be less than your threshold. This feature is turned off by default. When a mild threshold is specified, your profile job returns one more count, `zScoreMildOutliersCount`. Also, `zScoreOutliersSample` can include a sample of mild threshold outliers in this case.

Settings

- `threshold` – The threshold value to use when detecting outliers. This value should be greater or equal to 0.
- `mildThreshold` – The mild threshold value to use when detecting outliers. This value should be greater or equal to 0 and less than `threshold`.
- `sampleSize` – The size of list that includes outliers in the column. Use "ALL" to list all values.

Example

```
{
  "Statistic": "OUTLIER_DETECTION",
  "Parameters": {
    "threshold": "5",
    "mildThreshold": "3.5",
    "sampleSize": "20"
  }
}
```

VALUE_DISTRIBUTION

Measures the distribution of values in the column by the values' ranges. A profile job groups values from a numeric column or string column (based on string length) into bins by numeric ranges, and generates a list of bins. Bins are consecutive, and the upper bound for a bucket is the lower bound for the next bucket.

Settings

`binNumber` – Number of bins. This value should be greater than 0.

Example

```
{
  "Statistic": "VALUE_DISTRIBUTION",
  "Parameters": {
    "binNumber": "5"
  }
}
```

Z_SCORE_DISTRIBUTION

Measures the distribution of values' z-scores in numeric column. A profile job groups z-scores of values into bins by numeric ranges, and generates a list of bins. Bins are consecutive, and the upper bound for a bucket is the lower bound for the next bucket.

Settings

binNumber – Number of bins. This value should be greater than 0.

Example

```
{
  "Statistic": "Z_SCORE_DISTRIBUTION",
  "Parameters": {
    "binNumber": "5"
  }
}
```

EntityDetectorConfiguration section for configuring PII

In the `EntityDetectorConfiguration` section of your structure, you can configure the entity types in your dataset that you want DataBrew to detect as **personally identifiable information** (PII) for a profile job.

EntityTypes

You configure the entity types you want DataBrew to detect as PII for your profile job. When `EntityDetectorConfiguration` is undefined, entity detection is disabled. The following entity types can be detected in your dataset:

- `USA_SSN`

- EMAIL
- USA_ITIN
- USA_PASSPORT_NUMBER
- PHONE_NUMBER
- USA_DRIVING_LICENSE
- BANK_ACCOUNT
- CREDIT_CARD
- IP_ADDRESS
- MAC_ADDRESS
- USA_DEA_NUMBER
- USA_HCPCS_CODE
- USA_NATIONAL_PROVIDER_IDENTIFIER
- USA_NATIONAL_DRUG_CODE
- USA_HEALTH_INSURANCE_CLAIM_NUMBER
- USA_MEDICARE_BENEFICIARY_IDENTIFIER
- USA_CPT_CODE
- PERSON_NAME
- DATE

The entity type group USA_ALL is also supported, and includes all of the above entity types except PERSON_NAME and DATE.

The type of EntityTypes is an array of strings.

AllowedStatistics

Configure the statistics that are allowed to be run on columns that contain detected entities. If AllowedStatistics is undefined, no statistics will be computed on columns that contain detected entities. See [Configurable statistics at the column level](#) for a list of valid values for the AllowedStatistics parameter.

The type of AllowedStatistics is an array of AllowedStatistics objects.

Security in AWS Glue DataBrew

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The [shared responsibility model](#) describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the [AWS Compliance Programs](#). To learn about the compliance programs that apply to AWS Glue DataBrew, see [AWS services in Scope by Compliance Program](#).
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using AWS Glue DataBrew. The following topics show you how to configure DataBrew to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your DataBrew resources.

Topics

- [Data protection in AWS Glue DataBrew](#)
- [Identity and access management for AWS Glue DataBrew](#)
- [Logging and monitoring in DataBrew](#)
- [Compliance validation for AWS Glue DataBrew](#)
- [Resilience in AWS Glue DataBrew](#)
- [Infrastructure security in AWS Glue DataBrew](#)
- [Configuration and vulnerability analysis in AWS Glue DataBrew](#)

Data protection in AWS Glue DataBrew

DataBrew offers several features that are designed to help protect your data.

Topics

- [Encryption at rest](#)
- [Encryption in transit](#)
- [Key management](#)
- [Identifying and handling personally identifiable information \(PII\)](#)
- [DataBrew dependency on other AWS services](#)

The AWS [shared responsibility model](#) applies to data protection in AWS Glue DataBrew. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see [Data Privacy FAQ](#). For information about data protection in Europe, see the [General Data Protection Regulation \(GDPR\) Center](#).

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see [Working with CloudTrail trails](#) in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see [Federal Information Processing Standard \(FIPS\) 140-3](#).

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with DataBrew or other AWS services using the console, API, AWS CLI, or AWS

SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Encryption at rest

DataBrew supports data encryption at rest for DataBrew projects and jobs. Projects and jobs can read encrypted data, and jobs can write encrypted data by calling [AWS Key Management Service \(AWS KMS\)](#) to generate keys and decrypt data. You can also use KMS keys to encrypt the job logs that are generated by DataBrew jobs. You can specify encryption keys using the DataBrew console or the DataBrew API.

Important

AWS Glue DataBrew supports only symmetric AWS KMS keys. For more information, see [AWS KMS keys](#) in the *AWS Key Management Service Developer Guide*.

When you create jobs in DataBrew with encryption enabled, you can use the DataBrew console to specify S3-managed server-side encryption keys (SSE-S3) or KMS keys stored in AWS KMS (SSE-KMS) to encrypt data at rest.

Important

When you use an Amazon Redshift dataset, objects unloaded to the provided temporary directory are encrypted with SSE-S3.

Encrypting data written by DataBrew jobs

DataBrew jobs can write to encrypted Amazon S3 targets and encrypted Amazon CloudWatch Logs.

Topics

- [Setting up DataBrew to use encryption](#)
- [Creating a route to AWS KMS for VPC jobs](#)
- [Setting up encryption with AWS KMS keys](#)

Setting up DataBrew to use encryption

Follow this procedure to set up your DataBrew environment to use encryption.

To set up your DataBrew environment to use encryption

1. Create or update your AWS KMS keys to give AWS KMS permissions to the AWS Identity and Access Management (IAM) roles that are passed to DataBrew jobs. These IAM roles are used to encrypt CloudWatch Logs and Amazon S3 targets. For more information, see [Encrypt Log Data in CloudWatch Logs Using AWS KMS](#) in the *Amazon CloudWatch Logs User Guide*.

In the following example, *"role1"*, *"role2"*, and *"role3"* are IAM roles that are passed to DataBrew jobs. This policy statement describes a KMS key policy that gives permission to the listed IAM roles to encrypt and decrypt with this KMS key.

```
{
  "Effect": "Allow",
  "Principal": {
    "Service": "logs.region.amazonaws.com",
    "AWS": [
      "role1",
      "role2",
      "role3"
    ]
  },
  "Action": [
    "kms:Encrypt*",
    "kms:Decrypt*",
    "kms:ReEncrypt*",
    "kms:GenerateDataKey*",
    "kms:Describe*"
  ],
  "Resource": "*"
}
```

The Service statement, shown as `"Service": "logs.region.amazonaws.com"`, is required if you use the key to encrypt CloudWatch Logs.

2. Ensure that the AWS KMS key is set to ENABLED before it is used.

For more information about specifying permissions using AWS KMS key policies, see [Using key policies in AWS KMS](#).

Creating a route to AWS KMS for VPC jobs

You can connect directly to AWS KMS through a private endpoint in your virtual private cloud (VPC) instead of connecting over the internet. When you use a VPC endpoint, communication between your VPC and AWS KMS is conducted entirely within the AWS network.

You can create an AWS KMS VPC endpoint within a VPC. Without this step, your DataBrew jobs might fail with a `kms timeout`. For detailed instructions, see [Connecting to AWS KMS Through a VPC Endpoint](#) in the *AWS Key Management Service Developer Guide*.

As you follow these instructions, on the [VPC console](#), make sure to do the following:

- Choose **Enable Private DNS name**.
- For **Security group**, choose the security group (including a self-referencing rule) that you use for your DataBrew job that accesses Java Database Connectivity (JDBC).

When you run a DataBrew job that accesses JDBC data stores, DataBrew must have a route to the AWS KMS endpoint. You can provide the route with a network address translation (NAT) gateway or with an AWS KMS VPC endpoint. To create a NAT gateway, see [NAT Gateways](#) in the *Amazon VPC User Guide*.

Setting up encryption with AWS KMS keys

When you enable encryption on a job, it applies to both Amazon S3 and CloudWatch. The IAM role that is passed must have the following AWS KMS permissions.

For more information, see the following topics in the *Amazon Simple Storage Service User Guide*:

- For information about SSE-S3, see [Protecting Data Using Server-Side Encryption with Amazon S3-Managed Encryption Keys \(SSE-S3\)](#).
- For information about SSE-KMS, see [Protecting Data Using Server-Side Encryption with AWS KMS-Managed Keys \(SSE-KMS\)](#).

Encryption in transit

AWS provides Secure Sockets Layer (SSL) encryption for data in flight.

DataBrew support for JDBC data sources comes through AWS Glue. When connecting to JDBC data sources, DataBrew uses the settings on your AWS Glue connection, including the **Require SSL connection** option. For more information, see [AWS Glue Connection Properties - AWS Glue](#) in the *AWS Glue Developer Guide*.

AWS KMS provides both "bring your own key" encryption and server-side encryption for DataBrew extract, transform, load (ETL) processing and for the AWS Glue Data Catalog.

Key management

You can use IAM with DataBrew to define users, AWS resources, groups, roles, and fine-grained policies regarding access, denial, and more.

You can define the access to the metadata using both resource-based and identity-based policies, depending on your organization's needs. Resource-based policies list the principals that are allowed or denied access to your resources, allowing you to set up policies such as cross-account access. Identity policies are specifically attached to users, groups, and roles within IAM.

DataBrew supports creating your own AWS KMS key "bring your own key" encryption. DataBrew also provides server-side encryption using KMS keys from AWS KMS for DataBrew jobs.

Identifying and handling personally identifiable information (PII)

When you build analytic functions or machine learning models, you need safeguards to prevent exposure of personally identifiable information (PII) data. *PII* is personal data that can be used to identify an individual, such as an address, bank account number, or phone number. For example, when data analysts and data scientists use datasets to discover general demographic information, they should not have access to specific individuals' PII.

DataBrew provides data masking mechanisms to obfuscate PII data during data preparation process. Depending on your organization's needs, there are different PII data redaction mechanisms available. You can obfuscate the PII data so that users can't revert it back, or you can make the obfuscation reversible.

Identifying and masking PII data in DataBrew involves building a set of transforms that customers can use to redact PII data. Part of this process is providing PII data detection and statistics in the **Data Profile overview** dashboard on the DataBrew console.

You can use the following data-masking techniques:

- *Substitution* – Replace PII data with other authentic-looking values.

- *Shuffling* – Shuffle the value from the same column in different rows.
- *Deterministic encryption* – Apply deterministic encryption algorithms to the column values. Deterministic encryption always produces the same ciphertext for a value.
- *Probabilistic encryption* – Apply probabilistic encryption algorithms to the column values. Probabilistic encryption produces different ciphertext each time that it's applied.
- *Decryption* – Decrypt columns based on encryption keys.
- *Nulling out or deletion* – Replace a particular field with a null value or delete the column.
- *Masking out* – Use character scrambling or mask certain portions in the columns.
- *Hashing* – Apply hash functions to the column values.

For more information on using transforms, see [Personally identifiable information \(PII\) recipe steps](#). For more information on using profile jobs to detect PII, including a list of the entity types that can be detected, see [EntityDetectorConfiguration section for configuring PII](#) in *Building a profile job configuration programmatically*.

DataBrew dependency on other AWS services

To work with the DataBrew console, you need a minimum set of permissions to work with the DataBrew resources for your AWS account. In addition to these DataBrew permissions, the console requires permissions from the following services:

- CloudWatch Logs permissions to display logs.
- IAM permissions to list and pass roles.
- Amazon EC2 permissions to list VPCs, subnets, security groups, instances, and other objects. DataBrew uses these permissions to set up Amazon EC2 items such as VPCs when running DataBrew jobs.
- Amazon S3 permissions to list buckets and objects.
- AWS Glue permissions to read AWS Glue schema objects, such as databases, partitions, tables, and connections.
- AWS Lake Formation permissions to work with Lake Formation data lakes.

Identity and access management for AWS Glue DataBrew

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in)

and *authorized* (have permissions) to use DataBrew resources. IAM is an AWS service that you can use with no additional charge.

Topics

- [Authenticating with identities](#)
- [Managing access using policies](#)
- [AWS Glue DataBrew and AWS Lake Formation](#)
- [How AWS Glue DataBrew works with IAM](#)
- [Identity-based policy examples for AWS Glue DataBrew](#)
- [AWS managed policies for AWS Glue DataBrew](#)
- [Troubleshooting identity and access in AWS Glue DataBrew](#)

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see [How to sign in to your AWS account](#) in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see [AWS Signature Version 4 for API requests](#) in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see [Tasks that require root user credentials](#) in the *IAM User Guide*.

Users and groups

An [IAM user](#) is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more

information, see [Require human users to use federation with an identity provider to access AWS using temporary credentials](#) in the *IAM User Guide*.

An [IAM group](#) specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see [Use cases for IAM users](#) in the *IAM User Guide*.

IAM roles

An [IAM role](#) is an identity with specific permissions that provides temporary credentials. You can assume a role by [switching from a user to an IAM role \(console\)](#) or by calling an AWS CLI or AWS API operation. For more information, see [Methods to assume a role](#) in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see [Overview of JSON policies](#) in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Define custom IAM permissions with customer managed policies](#) in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between

managed and inline policies, see [Choose between managed policies and inline policies](#) in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples include *IAM role trust policies* and *Amazon S3 bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. You must [specify a principal](#) in a resource-based policy.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

DataBrew does not support resource-based policies.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see [Access control list \(ACL\) overview](#) in the *Amazon Simple Storage Service Developer Guide*.

DataBrew does not support ACLs.

Other policy types

AWS supports additional policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries** – Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see [Permissions boundaries for IAM entities](#) in the *IAM User Guide*.
- **Service control policies (SCPs)** – Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see [Service control policies](#) in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see [Resource control policies \(RCPs\)](#) in the *AWS Organizations User Guide*.

- **Session policies** – Advanced policies passed as a parameter when creating a temporary session for a role or federated user. For more information, see [Session policies](#) in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see [Policy evaluation logic](#) in the *IAM User Guide*.

AWS Glue DataBrew and AWS Lake Formation

AWS Glue DataBrew supports AWS Lake Formation permissions for AWS Glue Data Catalog tables. When a dataset uses an AWS Glue Data Catalog table that is registered with Lake Formation, the IAM role provided to projects or jobs must have [DESCRIBE](#) and [SELECT](#) Lake Formation permissions on the table.

AWS Glue DataBrew supports writing to AWS Glue Data Catalog tables based on AWS Lake Formation. When a DataBrew job uses a Data Catalog that is registered with Lake Formation, the IAM role provided to the jobs must have [INSERT](#), [ALTER](#), and [DELETE](#) permissions from Lake Formation for the tables involved. The IAM role must have `glue:UpdateTable` permissions, and also permissions to the data location associated with the Data Catalog table.

How AWS Glue DataBrew works with IAM

Before you use IAM to manage access to DataBrew, you should understand what IAM features are available to use with DataBrew. To get a high-level view of how DataBrew and other AWS services work with IAM, see [AWS Services That Work with IAM](#) in the *IAM User Guide*.

Topics

- [DataBrew identity-based policies](#)
- [Resource-based policies in DataBrew](#)
- [DataBrew IAM Roles](#)

DataBrew identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources, and also the conditions under which actions are allowed or denied. DataBrew supports specific actions,

resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see [IAM JSON Policy Elements Reference](#) in the *IAM User Guide*.

Actions

Administrators can use AWS JSON policies to specify who has access to what. That is, an AWS JSON policy can specify which principal can perform actions on what resources, and under what conditions.

The Action element of a JSON policy describes the actions to which you can allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as permission-only actions that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called dependent actions.

Include actions in a policy to grant permissions to perform the associated operation.

Policy actions in DataBrew use the following prefix before the action: `databrew:`. For example, to grant someone permission to run an Amazon EC2 instance with the Amazon EC2 `RunInstances` API operation, you include the `ec2:RunInstances` action in their policy. Policy statements must include either an `Action` or `NotAction` element. DataBrew defines its own set of actions that describe tasks that you can perform with it.

To specify multiple actions in a single statement, separate them with commas as follows.

```
"Action": [
    "databrew:CreateRecipeJob",
    "databrew:UpdateSchedule"
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action.

```
"Action": "databrew:Describe*"
```

To see a list of DataBrew actions, see [Actions Defined by AWS Glue DataBrew](#) in the *IAM User Guide*.

Resources

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Resource JSON policy element specifies the object or objects to which the action applies. As a best practice, specify a resource using its [Amazon Resource Name \(ARN\)](#). For actions that don't support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": ""
```

The following are the DataBrew APIs that don't support resource level permissions:

- ListDatasets
- ListJobs
- ListProjects
- ListRecipes
- ListRulesets
- ListSchedules

The DataBrew dataset resource has the following Amazon Resource Name (ARN).

```
arn:${Partition}:databrew:${Region}:${Account}:dataset/${Name}
```

For more information about the format of ARNs, see [Amazon Resource Names \(ARNs\) and AWS Service Namespaces](#).

For example, to specify the i-1234567890abcdef0 instance in your statement, use the following ARN.

```
"Resource": "arn:aws:databrew:us-east-1:123456789012:dataset/my-chess-dataset"
```

To specify all instances that belong to a specific account, use the wildcard (*).

```
"Resource": "arn:aws:databrew:us-east-1:123456789012:dataset/*"
```

You can't perform some DataBrew actions, such as those for creating resources, on a specific resource. In those cases, you must use the wildcard (*).

```
"Resource": ""
```

To see a list of DataBrew resource types and their ARNs, see [Resources Defined by AWS Glue DataBrew](#) in the *IAM User Guide*. To learn with which actions you can specify the ARN of each resource, see [Actions Defined by AWS Glue DataBrew](#).

Condition keys

DataBrew doesn't provide any service-specific condition keys, but it does support using some global condition keys. To see all AWS global condition keys, see [AWS global condition context keys](#) in the *IAM User Guide*.

Examples

To view examples of DataBrew identity-based policies, see [Identity-based policy examples for AWS Glue DataBrew](#).

Resource-based policies in DataBrew

DataBrew doesn't support resource-based policies.

DataBrew IAM Roles

An [IAM role](#) is an entity within your AWS account that has specific permissions.

Using temporary credentials with DataBrew

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You get temporary security credentials by calling AWS STS API operations such as [AssumeRole](#) or [GetFederationToken](#).

DataBrew supports using temporary credentials.

Service-linked roles

[Service-linked roles](#) allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An administrator can view but not edit the permissions for service-linked roles.

Choosing an IAM role in DataBrew

When you create a dataset resource in DataBrew, you choose an IAM role to allow DataBrew access on your behalf. If you have previously created a service role or service-linked role, then DataBrew provides you with a list of roles to choose from. Make sure to choose a role that allows read access to an Amazon S3 bucket or AWS Glue Data Catalog resource, as appropriate.

Identity-based policy examples for AWS Glue DataBrew

By default, users and roles don't have permission to create or modify DataBrew resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS APIs. An administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see [Creating Policies on the JSON Tab](#) in the *IAM User Guide*.

Topics

- [Policy best practices](#)
- [Using the DataBrew console](#)
- [Allowing users to view their own permissions](#)
- [Managing DataBrew resources based on tags](#)

Policy best practices

Identity-based policies determine whether someone can create, access, or delete DataBrew resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see [AWS managed policies](#) or [AWS managed policies for job functions](#) in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see [Policies and permissions in IAM](#) in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to

service actions if they are used through a specific AWS service, such as CloudFormation. For more information, see [IAM JSON policy elements: Condition](#) in the *IAM User Guide*.

- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see [Validate policies with IAM Access Analyzer](#) in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see [Secure API access with MFA](#) in the *IAM User Guide*.

For more information about best practices in IAM, see [Security best practices in IAM](#) in the *IAM User Guide*.

Using the DataBrew console

To access the AWS Glue DataBrew console, you must have a minimum set of permissions. These permissions must enable you to list and view details about the DataBrew resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console doesn't function as intended for users or roles with that policy.

To ensure that users and roles can use the DataBrew console, also attach the following AWS managed policy to the entities. For more information, see [Adding Permissions to a User](#) in the *IAM User Guide*.

```
AWSDataBrewConsoleAccess
```

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the DataBrew API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

Allowing users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsForUser",
        "iam:ListAttachedUserPolicies",
        "iam:ListUserPolicies",
        "iam:GetUser"
      ],
      "Resource": ["arn:aws:iam::*:user/${aws:username}"]
    },
    {
      "Sid": "NavigateInConsole",
      "Effect": "Allow",
      "Action": [
        "iam:GetGroupPolicy",
        "iam:GetPolicyVersion",
        "iam:GetPolicy",
        "iam:ListAttachedGroupPolicies",
        "iam:ListGroupPolicies",
        "iam:ListPolicyVersions",
        "iam:ListPolicies",
        "iam:ListUsers"
      ],
      "Resource": "*"
    }
  ]
}
```

Managing DataBrew resources based on tags

You can use conditions in your identity-based policy to manage DataBrew resources based on tags, for example, to delete, update, or describe the resources. The following example shows a policy that denies the deletion of a project. However, deletion is denied only if the project tag *Owner* has the value of *admin*. This policy also grants the permissions necessary to deny this action on the console.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DeleteResourceInConsole",
      "Effect": "Allow",
      "Action": "databrew:DeleteProject",
      "Resource": "*"
    },
    {
      "Sid": "DenyDeleteProjectIfAdminTag",
      "Effect": "Deny",
      "Action": "databrew:DeleteProject",
      "Resource": "arn:aws:databrew:*:*:project/*",
      "Condition": {
        "StringEquals": {"aws:ResourceTag/Owner": "admin"}
      }
    }
  ]
}
```

You can attach this policy to the users in your account. If a user named *richard-roe* attempts to delete a DataBrew project, the resource must not be tagged *Owner=admin* or *owner=admin*. Otherwise, the user is denied permission to delete the project. The condition tag key *Owner* matches both *Owner* and *owner* because condition key names are not case-sensitive. For more information, see [IAM JSON Policy Elements: Condition](#) in the *IAM User Guide*.

Note

ListDatasets, ListJobs, ListProjects, ListRecipes, ListRulesets, and ListSchedules do not support tag-based access control.

AWS managed policies for AWS Glue DataBrew

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create [IAM customer managed policies](#) that

provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see [AWS managed policies](#) in the IAM User Guide.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the *ReadOnlyAccess* AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see [AWS managed policies for job functions](#) in the IAM User Guide.

DataBrew updates to AWS managed policies

View details about updates to AWS managed policies for DataBrew since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the DataBrew Document history page. The managed policy can be found on the AWS IAM console at [AwsGlueDataBrewFullAccessPolicy](#).

Change	Description	Date
AWSGlueDataBrewServiceRole – Read permission for AWS Glue was added.	This update adds <code>glue:GetCustomEntityType</code> . This permission is required to execute AWS Glue DataBrew profile jobs with PII-identification enabled.	March 20, 2024
AWSGlueDataBrewServiceRole - Read permission for AWS Glue was added.	This update adds <code>glue:BatchGetCustomEntityTypes</code> . This permission is required to execute AWS Glue	May 9, 2022

Change	Description	Date
	DataBrew profile jobs with PII-identification enabled.	
AwsGlueDataBrewFullAccessPolicy - Read permissions for Amazon Redshift-Data DescribeStatements and Amazon S3 GetLifecycleConfiguration were added.	This update adds <code>redshift-data:DescribeStatement</code> to support validating your SQL when creating an Amazon Redshift-based dataset. It also adds <code>s3:GetLifecycleConfiguration</code> to evaluate whether or not the Amazon S3 bucket prefix you are providing as a temporary directory has the lifecycle configured. Additionally, this change replaces "databrew:*" permissions with an explicit list of permissions including all DataBrew APIs.	February 4, 2022

Change	Description	Date
AwsGlueDataBrewFullAccessPolicy - Read/write permissions for AWS Secrets Manager were added.	This update adds <code>secretsmanager:CreateSecret</code> and <code>secretsmanager:GetSecretValue</code> for a secret named <code>databrew!default</code> , a default secret for use with DataBrew transforms. Additionally, it adds permissions to <code>CreateSecret</code> for secrets prefixed with <code>AwsGlueDataBrew-</code> for creating secrets from the DataBrew console. GenerateRandom , described in the <i>AWS Key Management Service API Reference</i> , is used to generate a random byte string that is cryptographically secure.	November 18, 2021
AWSGlueDataBrewServiceRole - Read/write permissions for AWS Secrets Manager were added.	This update adds <code>secretsmanager:GetSecretValue</code> for a secret named <code>databrew!default</code> , a default secret for use with DataBrew transforms.	November 18, 2021

Change	Description	Date
AwsGlueDataBrewFullAccessPolicy - Read/write permissions for AWS Secrets Manager were added.	This update adds <code>secretsmanager:CreateSecret</code> and <code>secretsmanager:GetSecretValue</code> for a secret named <code>databrew!default</code> , a default secret for use with DataBrew transforms. Additionally, it adds permissions to <code>CreateSecret</code> for secrets prefixed with <code>AwsGlueDataBrew-</code> for creating secrets from the DataBrew console. <code>kms:GenerateRandom</code> (https://docs.aws.amazon.com/kms/latest/APIReference/API_GenerateRandom.html) is used to generate a random byte string that is cryptographically secure.	November 18, 2021
AWSGlueDataBrewServiceRole - Read/write permissions for AWS Secrets Manager were added.	This update adds <code>secretsmanager:GetSecretValue</code> for a secret named <code>databrew!default</code> , a default secret for use with DataBrew transforms.	November 18, 2021

Change	Description	Date
AwsGlueDataBrewFullAccessPolicy - Read permissions for AWS Glue catalog databases and create permissions for AWS Glue catalog table were added.	This update adds permissions to list AWS Glue Catalog databases and create new catalog tables under an existing database as part of configuring output to DataBrew jobs.	June 30, 2021
AwsGlueDataBrewFullAccessPolicy - Read/write permissions for Amazon AppFlow dataset feature were added.	This update adds permissions to read existing Amazon AppFlow flows and flow executions and to create flow executions.	April 28, 2021
AwsGlueDataBrewFullAccessPolicy - Read permissions for database datasets were added.	This update adds permissions to read existing AWS Glue connections and create new AWS Glue connections for use with DataBrew. Also, to make the console experience of creating new connections easier, it allows listing of Amazon VPC resources and Amazon Redshift clusters. It also gives permission to list, but not read, AWS Secrets Manager secrets.	March 30, 2021
DataBrew started tracking changes	DataBrew started tracking changes for its AWS managed policies.	March 30, 2021

Troubleshooting identity and access in AWS Glue DataBrew

Use the following information to help you diagnose and fix common issues that you might encounter when working with DataBrew and IAM.

Topics

- [I am not authorized to perform an action in DataBrew](#)
- [I am not authorized to perform iam:PassRole](#)
- [I want to allow people outside of my AWS account to access my DataBrew resources](#)

I am not authorized to perform an action in DataBrew

If the AWS Management Console tells you that you're not authorized to perform an action, contact your administrator for assistance. Your administrator is the person that provided you with your sign-in credentials.

The following example error occurs when the `mateojackson` user tries to use the console to view details about a project but doesn't have `databrew:DescribeProject` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
databrew:DescribeProject on resource: my-example-project
```

In this case, Mateo asks his administrator to update his policies to allow him to access the *my-example-project* resource using the `databrew:GetProject` action.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to DataBrew.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in DataBrew. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my DataBrew resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether DataBrew supports these features, see [How AWS Glue DataBrew works with IAM](#).
- To learn how to provide access to your resources across AWS accounts that you own, see [Providing access to an IAM user in another AWS account that you own](#) in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see [Providing access to AWS accounts owned by third parties](#) in the *IAM User Guide*.
- To learn how to provide access through identity federation, see [Providing access to externally authenticated users \(identity federation\)](#) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

Logging and monitoring in DataBrew

Monitoring is an important part of maintaining the reliability, availability, and performance of DataBrew and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multipoint failure if one occurs. AWS provides several tools for monitoring your DataBrew resources and responding to potential incidents:

Amazon CloudWatch Alarms

Using Amazon CloudWatch alarms, you watch a single metric over a time period that you specify. If the metric exceeds a given threshold, a notification is sent to an Amazon SNS topic or AWS Auto Scaling policy. CloudWatch alarms don't invoke actions because they are in a particular state. Rather, the state must have changed and been maintained for a specified number of periods.

AWS CloudTrail Logs

CloudTrail provides a record of actions taken by a user, role, or an AWS service in DataBrew. Using the information collected by CloudTrail, you can determine the request that was made to DataBrew, the IP address from which the request was made, who made the request, when it was made, and additional details.

Compliance validation for AWS Glue DataBrew

Third-party auditors assess the security and compliance of AWS Glue DataBrew as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

Our new AWS sign-up experience is not designed for regulated workloads. If you're using our new AWS sign-up experience, but you want to use AWS for regulated workloads, you can [sign up for AWS \(advanced\)](#) or [activate advanced features](#) for your AWS environment.

To learn whether an AWS service is within the scope of specific compliance programs, see [AWS services in Scope by Compliance Program](#) and choose the compliance program that you are interested in. For general information, see [AWS Compliance Programs](#).

You can download third-party audit reports using AWS Artifact. For more information, see [Downloading Reports in AWS Artifact](#).

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. For more information about your compliance responsibility when using AWS services, see [AWS Security Documentation](#).

Resilience in AWS Glue DataBrew

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For AWS Glue DataBrew, we suggest that you configure your jobs to use one or more retries. The number of retries for a job is configured in the DataBrew console under **Advanced job settings**.

For more information about AWS Regions and Availability Zones, see [AWS Global Infrastructure](#).

Infrastructure security in AWS Glue DataBrew

As part of a managed service, AWS Glue DataBrew is protected by the AWS global network security procedures that are described in the [Amazon Web Services: Overview of Security Processes](#) whitepaper.

You use AWS published API calls to access DataBrew through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the [AWS Security Token Service](#) (AWS STS) to generate temporary security credentials to sign requests.

Topics

- [Using AWS Glue DataBrew with your VPC](#)
- [Using AWS Glue DataBrew with VPC endpoints](#)

Using AWS Glue DataBrew with your VPC

If you use Amazon VPC to host your AWS resources, you can configure AWS Glue DataBrew to route traffic through your virtual private cloud (VPC) based on the Amazon VPC service. DataBrew does

this by first provisioning an elastic network interface in the subnet that you specify. DataBrew then attaches the security group that you specify to that network interface to control access. The specified security group must have self-referencing inbound and outbound rules for all traffic. Also, your VPC must have DNS hostnames and resolution turned on. For more information, see [Setting Up a VPC to Connect to JDBC Data Stores](#) in the *AWS Glue Developer Guide*.

For AWS Glue Data Catalog datasets, VPC information is configured when you create an AWS Glue connection in the Data Catalog. To create Data Catalog tables for this connection, run a crawler from the AWS Glue console. For more information, see [Populating the AWS Glue Data Catalog](#) in the *AWS Glue Developer Guide*.

For database datasets, specify your VPC information when you create the connection from the DataBrew console.

To use AWS Glue DataBrew with a VPC subnet without a [NAT](#), you must have a gateway VPC endpoint to Amazon S3 and a VPC endpoint for the AWS Glue interface. For more information, see [Create a gateway endpoint](#) and [Interface VPC endpoints \(AWS PrivateLink\)](#) in the Amazon VPC documentation. The elastic interface provisioned by DataBrew does not have a public IPv4 address, and so it does not support use of a VPC Internet Gateway.

Amazon S3 interface endpoints are not supported at this time. If you are using AWS Secrets Manager to store your secret, you need a route to Secrets Manager. If you are using encryption, you need a route to AWS Key Management Service (AWS KMS).

Using AWS Glue DataBrew with VPC endpoints

If you use Amazon VPC to host your AWS resources, you can establish a private connection between your VPC and DataBrew by provisioning a VPC endpoint. Using this VPC endpoint, you can make DataBrew API calls.

A DataBrew VPC endpoint is not required to use DataBrew with your VPC. For more information, see [Using AWS Glue DataBrew with your VPC](#).

You can use AWS Glue with VPC endpoints in all AWS Regions that support both AWS Glue and VPC endpoints.

For more information, see these topics in the *Amazon VPC User Guide*:

- [What Is Amazon VPC?](#)
- [Creating an Interface Endpoint](#)

Configuration and vulnerability analysis in AWS Glue DataBrew

Configuration and IT controls are a shared responsibility between AWS and you, our customer. For more information, see the AWS [shared responsibility model](#).

Monitoring AWS Glue DataBrew

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS Glue DataBrew and your other AWS solutions. AWS provides the following monitoring tools to watch DataBrew, report when something is wrong, and take automatic actions when appropriate:

- *Amazon CloudWatch* monitors your AWS resources and the applications you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For example, you can have CloudWatch track CPU usage or other metrics of your Amazon EC2 instances and automatically launch new instances when needed. For more information, see the [Amazon CloudWatch User Guide](#).
- *Amazon CloudWatch Events* enables you to set up automatic notifications for specific events in DataBrew. Events from DataBrew are delivered to CloudWatch Events in near-real time. You can configure CloudWatch Events to monitor events and invoke targets in response to events that indicate changes to your resource shares. Changes to a resource share trigger events for both the owner of the resource share and the principals that were granted access to the resource share. For more information, see the [Amazon CloudWatch Events User Guide](#).
- *Amazon CloudWatch Logs* enables you to monitor, store, and access your log files from Amazon EC2 instances, CloudTrail, and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the [Amazon CloudWatch Logs User Guide](#).
- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account. It then delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the [AWS CloudTrail User Guide](#).

Topics

- [Monitoring DataBrew with Amazon CloudWatch](#)
- [Automating DataBrew with CloudWatch Events](#)
- [Monitoring DataBrew with CloudWatch Logs](#)
- [Logging DataBrew API calls with AWS CloudTrail](#)
- [Using AWS User Notifications with AWS Glue Databrew](#)

Monitoring DataBrew with Amazon CloudWatch

You can monitor DataBrew using CloudWatch, which collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months, so that you can access historical information and gain a better perspective on how your web application or service is performing. You can also set alarms that watch for certain thresholds, and send notifications or take actions when those thresholds are met. For more information, see the [Amazon CloudWatch User Guide](#).

AWS Glue DataBrew reports the following metrics in the AWS/DataBrew namespace.

Metric	Description
SessionCount	The total number of DataBrew sessions across the customer's account Valid Dimensions: LogGroupName Valid Statistic: Sum Units: Count

Automating DataBrew with CloudWatch Events

Amazon CloudWatch Events enables you to automate your AWS services and respond automatically to system events such as application availability issues or resource changes. Events from AWS services are delivered to CloudWatch Events in near-real time. You can write simple rules to indicate which events are of interest to you, and what automated actions to take when an event matches a rule. The actions that can be automatically triggered include the following:

- Invoking the Amazon EC2 run command
- Relaying the event to Amazon Kinesis Data Streams
- Activating an AWS Step Functions state machine
- Notifying an Amazon SNS topic or an Amazon SQS queue

DataBrew reports an event to CloudWatch Events whenever the state of a resource in your AWS account changes. Events are emitted on a best effort basis.

Following are examples of several events, showing various states of a DataBrew job: SUCCEEDED, FAILED, TIMEOUT, and STOPPED.

```
{
  "version": "0",
  "id": "abcdef00-1234-5678-9abc-def012345678",
  "detail-type": "DataBrew Job State Change",
  "source": "aws.databrew",
  "account": "123456789012",
  "time": "2017-09-07T18:57:21Z",
  "region": "us-west-2",
  "resources": [],
  "detail": {
    "jobName": "MyJob",
    "severity": "INFO",
    "state": "SUCCEEDED",
    "jobRunId": "db_abcdef0123456789abcdef0123456789abcdef0123456789",
    "message": "Job run succeeded"
  }
}

{
  "version": "0",
  "id": "abcdef01-1234-5678-9abc-def012345678",
  "detail-type": "DataBrew Job State Change",
  "source": "aws.databrew",
  "account": "123456789012",
  "time": "2017-09-07T06:02:03Z",
  "region": "us-west-2",
  "resources": [],
  "detail": {
    "jobName": "MyJob",
    "severity": "ERROR",
    "state": "FAILED",
    "jobRunId": "db_0123456789abcdef0123456789abcdef0123456789abcdef",
    "message": "AnalysisException: 'Path does not exist: s3://MyBucket/MyFile;'"
  }
}

{
  "version": "0",
  "id": "abcdef00-1234-5678-9abc-def012345678",
```

```

    "detail-type": "DataBrew Job State Change",
    "source": "aws.databrew",
    "account": "123456789012",
    "time": "2017-11-20T20:22:06Z",
    "region": "us-east-2",
    "resources": [],
    "detail": {
      "jobName": "MyJob",
      "severity": "WARN",
      "state": "TIMEOUT",
      "jobRunId": "db_abc0123456789abcdef0123456789abcdef0123456789abcdef0123456789def",
      "message": "Job run timed out"
    }
  }

  {
    "version": "0",
    "id": "abcdef00-1234-5678-9abc-def012345678",
    "detail-type": "DataBrew Job State Change",
    "source": "aws.databrew",
    "account": "123456789012",
    "time": "2017-11-20T20:22:06Z",
    "region": "us-east-2",
    "resources": [],
    "detail": {
      "jobName": "MyJob",
      "severity": "INFO",
      "state": "STOPPED",
      "jobRunId": "db_abc0123456789abcdef0123456789abcdef0123456789abcdef0123456789def",
      "message": "Job run stopped"
    }
  }
}

```

For more information, see the [Amazon CloudWatch Events User Guide](#).

Monitoring DataBrew with CloudWatch Logs

You can monitor DataBrew jobs using CloudWatch Logs, which collects detailed information from the DataBrew job subsystem and makes it available for review. These logs can be helpful if you want to gain insight into the resources your profile and recipe jobs are using, or for troubleshooting purposes. For more information, see the [Amazon CloudWatch Logs User Guide](#).

Logging DataBrew API calls with AWS CloudTrail

DataBrew is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in DataBrew. CloudTrail captures all API calls for DataBrew as events. The calls captured include calls from the DataBrew console and code calls to the DataBrew API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for DataBrew. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to DataBrew. You can also determine the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the [AWS CloudTrail User Guide](#).

DataBrew Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in DataBrew, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see [Viewing Events with CloudTrail Event History](#) in the *AWS CloudTrail User Guide*.

For an ongoing record of events in your AWS account, including events for DataBrew, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following in the *AWS CloudTrail User Guide*:

- [Overview for Creating a Trail](#)
- [CloudTrail Supported Services and Integrations](#)
- [Configuring Amazon SNS Notifications for CloudTrail](#)
- [Receiving CloudTrail Log Files from Multiple Regions](#) and [Receiving CloudTrail Log Files from Multiple Accounts](#)

All DataBrew actions are logged by CloudTrail and are documented in the [API Reference](#). For example, calls to the `CreateDataset`, `UpdateRecipe` and `StartJobRun` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the [CloudTrail userIdentity Element](#).

Understanding DataBrew Log File Entries

Again, a CloudTrail *trail* is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An *event* represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the CreateProfileJob operation.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::1234567890:user/joe",
    "accountId": "1234567890",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "joe"
  },
  "eventTime": "2020-11-09T18:54:44Z",
  "eventSource": "databrew.amazonaws.com",
  "eventName": "CreateProfileJob",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.0.2.0",
  "requestParameters": {
    "OutputLocation": {
      "Bucket": "bucketName",
      "Key": "keyName"
    }
  },
}
```

```
    "DatasetName": "my-chess-dataset",
    "RoleArn": "arn:aws:iam::1234567890:role/custom-role",
    "Name": "my-profile-job"
  },
  "responseElements": {
    "Name": "my-profile-job"
  },
  "requestID": "993bc3b8-3980-48dd-961e-c1c8529eb248",
  "eventID": "f8128dfa-df29-458b-a2d5-34805b46eefd",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "recipientAccountId": "1234567890"
}
```

Using AWS User Notifications with AWS Glue Databrew

You can use [AWS User Notifications](#) to set up delivery channels to get notified about AWS Glue Databrew events. You receive a notification when an event matches a rule that you specify. You can receive notifications for events through multiple channels, including email, [Amazon Q Developer in chat applications](#) chat notifications, or [AWS Console Mobile Application](#) push notifications. You can also see notifications in the [Console Notifications Center](#). AWS User Notifications supports aggregation, which can reduce the number of notifications you receive during specific events.

Recipe step and function reference

In this reference, you can find descriptions of the recipe steps and functions that you can use programmatically, either from the AWS CLI or by using one of the AWS SDKs. In DataBrew, a *recipe step* is an action that transforms your raw data into a form that is ready to be consumed by your data pipeline. A DataBrew *function* is a special kind of recipe step that performs a computation based on parameters.

Categories for transformations in the UI include the following:

- Basic column recipe steps
 - Filter
 - Column
- Data cleaning recipe steps
 - Format
 - Clean
 - Extract
- Data quality recipe steps
 - Missing
 - Invalid
 - Duplicates
 - Outliers
- Personally identifiable information (PII) recipe steps
 - Mask personal information
 - Replace personal information
 - Encrypt personal information
 - Shuffle rows
- Column structure recipe steps
 - Split
 - Merge
 - Create
- Column formatting recipe steps

- Decimal precision
- Thousands separator
- Abbreviate numbers
- Data structure recipe steps
 - Nest-Unnest
 - Pivot
 - Group
 - Join
 - Union
- Data science recipe steps
 - Text
 - Scale
 - Mapping
 - Encode
- Functions
 - Mathematical functions
 - Aggregate functions
 - Text functions
 - Date and time functions
 - Window functions
 - Web functions
 - Other functions

For more information about how these recipe steps and functions are used in a recipe (including the use of condition expressions) see [Defining a recipe structure](#).

The following sections describe the recipe steps and functions, organized by what they do.

Topics

- [Basic column recipe steps](#)
- [Data cleaning recipe steps](#)
- [Data quality recipe steps](#)

- [Personally identifiable information \(PII\) recipe steps](#)
- [Outlier detection and handling recipe steps](#)
- [Column structure recipe steps](#)
- [Column formatting recipe steps](#)
- [Data structure recipe steps](#)
- [Data science recipe steps](#)
- [Mathematical functions](#)
- [Aggregate functions](#)
- [Text functions](#)
- [Date and time functions](#)
- [Window functions](#)
- [Web functions](#)
- [Other functions](#)

Basic column recipe steps

Use these basic column recipe actions to perform simple transformations on your data.

Topics

- [CHANGE_DATA_TYPE](#)
- [DELETE](#)
- [DUPLICATE](#)
- [JSON_TO_STRUCTS](#)
- [MOVE_AFTER](#)
- [MOVE_BEFORE](#)
- [MOVE_TO_END](#)
- [MOVE_TO_INDEX](#)
- [MOVE_TO_START](#)
- [RENAME](#)
- [SORT](#)

- [TO_BOOLEAN_COLUMN](#)
- [TO_DOUBLE_COLUMN](#)
- [TO_NUMBER_COLUMN](#)
- [TO_STRING_COLUMN](#)

CHANGE_DATA_TYPE

Changes the data type of an existing column.

If a column value can't be converted to the new type, it will be replaced with NULL. This can happen when a string column is converted to an integer column. For example, string "123" will become integer 123, but string "ABC" cannot become a number, so it will be replaced with a NULL value.

Parameters

- **sourceColumn** – The name of an existing column.
- **columnDataType** – New type of the column. The following data types are supported:
 - **byte**: 1-byte signed integer numbers. The range of numbers is from -128 to 127.
 - **short**: 2-byte signed integer numbers. The range of numbers is from -32768 to 32767.
 - **int**: 4-byte signed integer numbers. The range of numbers is from -2147483648 to 2147483647.
 - **long**: 8-byte signed integer numbers. The range of numbers is from -9223372036854775808 to 9223372036854775807.
 - **float**: 4-byte single-precision floating point numbers.
 - **double**: 8-byte double-precision floating point numbers.
 - **decimal**: Signed decimal numbers with up to 38 digits total and 18 digits after the decimal point.
 - **string**: Character string values.
 - **boolean**: Boolean type has one of two possible values: `true` and `false` or `yes` and `no`.
 - **timestamp**: Values comprising fields year, month, day, hour, minute, and second.
 - **date**: Values comprising fields year, month and day.

Example

```
{
  "RecipeAction": {
    "Operation": "CHANGE_DATA_TYPE",
    "Parameters": {
      "sourceColumn": "columnName",
      "columnDataType": "boolean"
    }
  }
}
```

DELETE

Removes a column from the dataset.

Parameters

- `sourceColumn` – The name of an existing column.

Example

```
{
  "RecipeAction": {
    "Operation": "DELETE",
    "Parameters": {
      "sourceColumn": "extra_data"
    }
  }
}
```

DUPLICATE

Creates a new column with the different name, but with all of the same data. Both the old and new columns are retained in the dataset.

Parameters

- `sourceColumn` – The name of an existing column.
- `targetColumn` – A name for the duplicate column.

Example

```
{
  "RecipeAction": {
    "Operation": "DUPLICATE",
    "Parameters": {
      "sourceColumn": "last_name",
      "targetColumn": "copy_of_last_name"
    }
  }
}
```

JSON_TO_STRUCTS

Converts a JSON string to statically typed structs. During conversion, it detects the schema of every JSON object and merges them in order to get the most generic schema to represent the entire JSON string. The “`unnestLevel`” parameter specifies how many levels of JSON objects to convert to structs.

Parameters

- `sourceColumns` – A list of source columns.
- `regexColumnSelector` – A regular expression to select the columns.
- `removeSourceColumn` – A Boolean value. If `true` then remove the source column; otherwise, keep it.
- `unnestLevel` – The number of levels to unnest.
- `conditionExpressions` – Condition expressions.

Example

```
{
  "RecipeAction": {
    "Operation": "JSON_TO_STRUCTS",
    "Parameters": {
      "sourceColumns": "[\"address\"]",
      "removeSourceColumn": "true",
      "unnestLevel": "2"
    }
  }
}
```

```
    }  
  }  
}
```

MOVE_AFTER

Moves a column to the position immediately after another column.

Parameters

- **sourceColumn** – The name of an existing column.
- **targetColumn** – The name of another column. The column specified by **sourceColumn** will be moved immediately after the column specified by **targetColumn**.

Example

```
{  
  "RecipeAction": {  
    "Operation": "MOVE_AFTER",  
    "Parameters": {  
      "sourceColumn": "rating",  
      "targetColumn": "height_cm"  
    }  
  }  
}
```

MOVE_BEFORE

Moves a column to the position immediately before another column.

Parameters

- **sourceColumn** – The name of an existing column.
- **targetColumn** – The name of another column. The column specified by **sourceColumn** will be moved immediately after the column specified by **targetColumn**.

Example

```
{
  "RecipeAction": {
    "Operation": "MOVE_BEFORE",
    "Parameters": {
      "sourceColumn": "height_cm",
      "targetColumn": "weight_kg"
    }
  }
}
```

MOVE_TO_END

Moves a column to the end position (last column) in the dataset.

Parameters

- `sourceColumn` – The name of an existing column.

Example

```
{
  "RecipeAction": {
    "Operation": "MOVE_TO_END",
    "Parameters": {
      "sourceColumn": "height_cm"
    }
  }
}
```

MOVE_TO_INDEX

Moves a column to a position specified by a number.

Parameters

- `sourceColumn` – The name of an existing column.
- `targetIndex` – The new position for the column. Positions start with 0—so, for example, 1 refers to the second column, 2 refers to the third column, and so on.

Example

```
{
  "RecipeAction": {
    "Operation": "MOVE_TO_INDEX",
    "Parameters": {
      "sourceColumn": "nationality",
      "targetIndex": "5"
    }
  }
}
```

MOVE_TO_START

Moves a column to the beginning position (first column) in the dataset.

Parameters

- `sourceColumn` – The name of an existing column.

Example

```
{
  "RecipeAction": {
    "Operation": "MOVE_TO_START",
    "Parameters": {
      "sourceColumn": "first_name"
    }
  }
}
```

RENAME

Creates a new column with the different name, but with all of the same data. The old column is then removed from the dataset.

Parameters

- `sourceColumn` – The name of an existing column.

- **targetColumn** – A new name for the column.

Example

```
{
  "RecipeAction": {
    "Operation": "RENAME",
    "Parameters": {
      "sourceColumn": "date_of_birth",
      "targetColumn": "birth_date"
    }
  }
}
```

SORT

Sorts the data in one or more columns of a dataset in ascending, descending, or custom order.

Parameters

- **expressions** – A string that contains one or more JSON-encoded strings representing sorting expressions.
 - **sourceColumn** – A string that contains the name of an existing column.
 - **ordering** – Ordering can be either **ASCENDING** or **DESCENDING**.
 - **nullsOrdering** – Nulls ordering can be either **NULLS_TOP** or **NULLS_BOTTOM** to place null or missing values at the beginning or at the bottom of the column.
 - **customOrder** – A list of strings that defines a custom order for the string sorting. By default, strings are sorted alphabetically.
 - **isCustomOrderCaseSensitive** – Boolean. The default value is **false**.

Example

```
{
  "RecipeAction": {
    "Operation": "SORT",
    "Parameters": {
```

```
        "expressions": "[{\"sourceColumn\": \"A\", \"ordering\": \"ASCENDING\",  
        \"nullsOrdering\": \"NULLS_TOP\"}]",  
    }  
}
```

Example of custom sort order

In the following example, the customOrder expression string has the format of a list of objects. Each object describes a sorting expression for one column.

```
[  
  {  
    "sourceColumn": "A",  
    "ordering": "ASCENDING",  
    "nullsOrdering": "NULLS_TOP",  
  },  
  {  
    "sourceColumn": "B",  
    "ordering": "DESCENDING",  
    "nullsOrdering": "NULLS_BOTTOM",  
    "customOrder": ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"],  
    "isCustomOrderCaseSensitive": false,  
  }  
]
```

TO_BOOLEAN_COLUMN

Changes the data type of an existing column to BOOLEAN.

Note

We recommend using CHANGE_DATA_TYPE recipe action rather than TO_BOOLEAN_COLUMN.

Parameters

- `sourceColumn` – The name of an existing column.

- `columnDataType` – A value that must be boolean.

Example

```
{
  "RecipeAction": {
    "Operation": "TO_BOOLEAN_COLUMN",
    "Parameters": {
      "columnDataType": "boolean",
      "sourceColumn": "is_present"
    }
  }
}
```

TO_DOUBLE_COLUMN

Changes the data type of an existing column to DOUBLE.

Note

We recommend using `CHANGE_DATA_TYPE` recipe action rather than `TO_DOUBLE_COLUMN`.

Parameters

- `sourceColumn` – The name of an existing column.
- `columnDataType` – A value that must be number.

Example

```
{
  "RecipeAction": {
    "Operation": "TO_DOUBLE_COLUMN",
    "Parameters": {
      "columnDataType": "number",
      "sourceColumn": "hourly_rate"
    }
  }
}
```

```
}  
}
```

TO_NUMBER_COLUMN

Changes the data type of an existing column to NUMBER.

Note

We recommend using CHANGE_DATA_TYPE recipe action rather than TO_NUMBER_COLUMN.

Parameters

- `sourceColumn` – The name of an existing column.
- `columnDataType` – A value that must be number.

Example

```
{  
  "RecipeAction": {  
    "Operation": "TO_NUMBER_COLUMN",  
    "Parameters": {  
      "columnDataType": "number",  
      "sourceColumn": "hours_worked"  
    }  
  }  
}
```

TO_STRING_COLUMN

Changes the data type of an existing column to STRING.

Note

We recommend using CHANGE_DATA_TYPE recipe action rather than TO_STRING_COLUMN.

Parameters

- `sourceColumn` – The name of an existing column.
- `columnDataType` – A value that must be `string`.

Example

```
{
  "RecipeAction": {
    "Operation": "TO_STRING_COLUMN",
    "Parameters": {
      "columnDataType": "string",
      "sourceColumn": "age"
    }
  }
}
```

Data cleaning recipe steps

Use these data cleaning recipe steps to perform simple transformations on existing data.

Topics

- [CAPITAL_CASE](#)
- [FORMAT_DATE](#)
- [LOWER_CASE](#)
- [UPPER_CASE](#)
- [SENTENCE_CASE](#)
- [ADD_DOUBLE_QUOTES](#)
- [ADD_PREFIX](#)
- [ADD_SINGLE_QUOTES](#)
- [ADD_SUFFIX](#)
- [EXTRACT_BETWEEN_DELIMITERS](#)
- [EXTRACT_BETWEEN_POSITIONS](#)

- [EXTRACT_PATTERN](#)
- [EXTRACT_VALUE](#)
- [REMOVE_COMBINED](#)
- [REPLACE_BETWEEN_DELIMITERS](#)
- [REPLACE_BETWEEN_POSITIONS](#)
- [REPLACE_TEXT](#)

CAPITAL_CASE

Changes each string in a column to capitalize each word. In *capital case*, the first letter of each word is capitalized and the rest of the word is transformed to lowercase. An example is: The Quick Brown Fox Jumped Over The Fence.

Parameters

- `sourceColumn` – The name of an existing column.

Example

```
{
  "RecipeAction": {
    "Operation": "CAPITAL_CASE",
    "Parameters": {
      "sourceColumn": "last_name"
    }
  }
}
```

FORMAT_DATE

Returns a column in which a date string is converted into a formatted value.

Parameters

- `sourceColumn` – The name of an existing column.
- `targetDateFormat` – One of the following date formats:

- mm/dd/yyyy
- mm-dd-yyyy
- dd month yyyy
- month yyyy
- dd month

Example

```
{
  "RecipeAction": {
    "Operation": "FORMAT_DATE",
    "Parameters": {
      "sourceColumn": "birth_date",
      "targetDateFormat": "mm-dd-yyyy"
    }
  }
}
```

LOWER_CASE

Changes each string in a column to lowercase, for example: the quick brown fox jumped over the fence

Parameters

- `sourceColumn` – The name of an existing column.

Example

```
{
  "RecipeAction": {
    "Operation": "LOWER_CASE",
    "Parameters": {
      "sourceColumn": "nationality"
    }
  }
}
```

UPPER_CASE

Changes each string in a column to uppercase, for example: THE QUICK BROWN FOX JUMPED OVER THE FENCE

Parameters

- `sourceColumn` – The name of an existing column.

Example

```
{
  "RecipeAction": {
    "Operation": "UPPER_CASE",
    "Parameters": {
      "sourceColumn": "nationality"
    }
  }
}
```

SENTENCE_CASE

Changes each string in a column to sentence case. In *sentence case*, the first letter of each sentence is capitalized, and the rest of the sentence is transformed to lowercase. An example is: The quick brown fox. Jumped over. The fence

Parameters

- `sourceColumn` – The name of an existing column.

Example

```
{
  "RecipeAction": {
    "Operation": "SENTENCE_CASE",
    "Parameters": {
      "sourceColumn": "description"
    }
  }
}
```

```
}  
}
```

ADD_DOUBLE_QUOTES

Encloses the characters in a column with double quotation marks.

Parameters

- `sourceColumn` – The name of an existing column.

Example

```
{  
  "RecipeAction": {  
    "Operation": "ADD_DOUBLE_QUOTES",  
    "Parameters": {  
      "sourceColumn": "info_url"  
    }  
  }  
}
```

ADD_PREFIX

Adds one or more characters, concatenating them as a prefix to the beginning of a column.

Parameters

- `sourceColumn` – The name of an existing column.
- `pattern` – The character or characters to place at the beginning of the column values.

Example

```
{  
  "RecipeAction": {  
    "Operation": "ADD_PREFIX",  
    "Parameters": {
```

```
        "pattern": "aaa",
        "sourceColumn": "info_url"
    }
}
```

ADD_SINGLE_QUOTES

Encloses the characters in a column with single quotation marks.

Parameters

- `sourceColumn` – The name of an existing column.

Example

```
{
  "RecipeAction": {
    "Operation": "ADD_SINGLE_QUOTES",
    "Parameters": {
      "sourceColumn": "info_url"
    }
  }
}
```

ADD_SUFFIX

Adds one more characters concatenating them as a suffix to the end of a column.

Parameters

- `sourceColumn` – The name of an existing column.
- `pattern` – The character or characters to place at the end of the column.

Example

```
{
  "RecipeAction": {
```


Parameters

- `sourceColumn` – The name of an existing column.
- `targetColumn` – The name of the new column to be created.
- `startPosition` – The character position at which to perform the extract.
- `endPosition` – The character position at which to end the extract.

Example

```
{
  "RecipeAction": {
    "Operation": "EXTRACT_BETWEEN_POSITIONS",
    "Parameters": {
      "endPosition": "9",
      "sourceColumn": "last_name",
      "startPosition": "3",
      "targetColumn": "characters_3_to_9"
    }
  }
}
```

EXTRACT_PATTERN

Creates a new column, based on a regular expression, from the values in an existing column.

Parameters

- `sourceColumn` – The name of an existing column.
- `targetColumn` – The name of the new column to be created.
- `pattern` – A regular expression that indicates which character or characters to extract and create the new column from.

Example

```
{
  "RecipeAction": {
```

```
    "Operation": "EXTRACT_PATTERN",
    "Parameters": {
      "pattern": "^....*...$",
      "sourceColumn": "last_name",
      "targetColumn": "first_and_last_few_characters"
    }
  }
}
```

EXTRACT_VALUE

Creates a new column with an extracted value from a user-specified path. If the source column is of the Map, Array, or Struct type, each field in the path should be escaped using back ticks (for example, `name`).

Parameters

- **targetColumn** – The name of the target column.
- **sourceColumn** – Name of the source column from which the value is to be extracted.
- **path** – The path to the specific key that the user wants to extract. If the source column is of the Map, Array, or Struct type, each field in the path should be escaped using back ticks (for example, `name`).

Consider the following example of user information:

```
user {
  name: "Ammy"
  address: {
    state: "CA",
    zipcode: 12345
  },
  phoneNumber: {"home": "123123123", "work": "456456456"}
  citizenship: ["Canada", "USA", "Mexico", "India"]
}
```

The following are examples of the paths you would provide, depending on the type of the source column:

- If the source column is of the type **map**, the path for extracting the home phone number is:

```
`user`.`phoneNumber`.`home`
```

- If the source column is of the type **array**, the path for extracting the second "citizenship" value is:

```
`user`.`citizenship`[1]
```

- If the source column is of the type **struct**, the path for extracting the zip code is:

```
`user`.`address`.`zipcode`
```

Example

```
{
  "RecipeAction": {
    "Operation": "EXTRACT_VALUE",
    "Parameters": {
      "sourceColumn": "age",
      "targetColumn": "columnName",
      "path": "`age`.`name`",
    }
  }
}
```

REMOVE_COMBINED

Removes one or more characters from a column, according to what a user specifies.

Parameters

- **sourceColumn** – The name of an existing column.
- **collapseConsecutiveWhitespace** – If **true**, replaces two or more white-space characters with exactly one white-space character.
- **removeAllPunctuation** – If **true**, removes all of the following characters: . ! , ?
- **removeAllQuotes** – If **true**, removes all single quotation marks and double quotation marks.
- **removeAllWhitespace** – If **true**, removes all white-space characters.
- **customCharacters** – One or more characters that can be acted upon.

- `customValue` – A value that can be acted upon.
- `removeCustomCharacters` – If true, removes all characters specified by `customCharacters` parameter.
- `removeCustomValue` – If true, removes all characters specified by `customValue` parameter.
- `punctuationally` – If true, removes the following characters if they occur at the start or end of the value: . ! , ?
- `antidisestablishmentarianism` – If true, removes single quotation marks and double quotation marks from the beginning and end of the value.
- `removeLeadingAndTrailingWhitespace` – If true, removes all white spaces from the beginning and end of the value.
- `removeLetters` – If true, removes all uppercase and lowercase alphabetic characters (A through Z; a through z).
- `removeNumbers` – If true, removes all numeric characters (0 through 9).
- `removeSpecialCharacters` – If true, removes all of the following characters: ! " # \$ % & ' () * + , - . / : ; < = > ? @ [\] ^ _ ` { | } ~

Examples

```
{
  "RecipeAction": {
    "Operation": "REMOVE_COMBINED",
    "Parameters": {
      "collapseConsecutiveWhitespace": "false",
      "removeAllPunctuation": "false",
      "removeAllQuotes": "false",
      "removeAllWhitespace": "false",
      "removeCustomCharacters": "false",
      "removeCustomValue": "false",
      "removeLeadingAndTrailingPunctuation": "false",
      "removeLeadingAndTrailingQuotes": "false",
      "removeLeadingAndTrailingWhitespace": "false",
      "removeLetters": "false",
      "removeNumbers": "false",
      "removeSpecialCharacters": "true",
      "sourceColumn": "info_url"
    }
  }
}
```

```
}
```

```
{
  "RecipeAction": {
    "Operation": "REMOVE_COMBINED",
    "Parameters": {
      "collapseConsecutiveWhitespace": "false",
      "customCharacters": "¶",
      "removeAllPunctuation": "false",
      "removeAllQuotes": "false",
      "removeAllWhitespace": "false",
      "removeCustomCharacters": "true",
      "removeCustomValue": "false",
      "removeLeadingAndTrailingPunctuation": "false",
      "removeLeadingAndTrailingQuotes": "false",
      "removeLeadingAndTrailingWhitespace": "false",
      "removeLetters": "false",
      "removeNumbers": "false",
      "removeSpecialCharacters": "false",
      "sourceColumn": "info_url"
    }
  }
}
```

```
{
  "RecipeAction": {
    "Operation": "REMOVE_COMBINED",
    "Parameters": {
      "collapseConsecutiveWhitespace": "true",
      "customValue": "M",
      "removeAllPunctuation": "true",
      "removeAllQuotes": "false",
      "removeAllWhitespace": "false",
      "removeCustomCharacters": "false",
      "removeCustomValue": "true",
      "removeLeadingAndTrailingPunctuation": "false",
      "removeLeadingAndTrailingQuotes": "true",
      "removeLeadingAndTrailingWhitespace": "true",
      "removeLetters": "true",
      "removeNumbers": "true",
      "removeSpecialCharacters": "false",
      "sourceColumn": "info_url"
    }
  }
}
```

```
}  
}
```

```
{  
  "RecipeAction": {  
    "Operation": "REMOVE_COMBINED",  
    "Parameters": {  
      "collapseConsecutiveWhitespace": "false",  
      "removeAllPunctuation": "false",  
      "removeAllQuotes": "false",  
      "removeAllWhitespace": "false",  
      "removeCustomCharacters": "false",  
      "removeCustomValue": "false",  
      "removeLeadingAndTrailingPunctuation": "false",  
      "removeLeadingAndTrailingQuotes": "false",  
      "removeLeadingAndTrailingWhitespace": "false",  
      "removeLetters": "false",  
      "removeNumbers": "true",  
      "removeSpecialCharacters": "false",  
      "sourceColumn": "first_name"  
    }  
  }  
}
```

```
{  
  "RecipeAction": {  
    "Operation": "REMOVE_COMBINED",  
    "Parameters": {  
      "collapseConsecutiveWhitespace": "false",  
      "removeAllPunctuation": "false",  
      "removeAllQuotes": "false",  
      "removeAllWhitespace": "false",  
      "removeCustomCharacters": "false",  
      "removeCustomValue": "false",  
      "removeLeadingAndTrailingPunctuation": "false",  
      "removeLeadingAndTrailingQuotes": "false",  
      "removeLeadingAndTrailingWhitespace": "false",  
      "removeLetters": "false",  
      "removeNumbers": "true",  
      "removeSpecialCharacters": "false",  
      "sourceColumn": "first_name"  
    }  
  }  
}
```

```
}
```

REPLACE_BETWEEN_DELIMITERS

Replaces the characters between two delimiters with user-specified text.

Parameters

- `sourceColumn` – The name of an existing column.
- `startPattern` – Character or characters or a regular expression, indicating where the substitution is to begin.
- `endPattern` – Character or characters or a regular expression, indicating where the substitution is to end.
- `value` – The replacement character or characters to be substituted.

Example

```
{
  "RecipeAction": {
    "Operation": "REPLACE_BETWEEN_DELIMITERS",
    "Parameters": {
      "endPattern": ">",
      "sourceColumn": "last_name",
      "startPattern": "&lt;",
      "value": "?"
    }
  }
}
```

REPLACE_BETWEEN_POSITIONS

Replaces the characters between two positions with user-specified text.

Parameters

- `sourceColumn` – The name of an existing column.
- `startPosition` – A number indicating at what character position in the string the substitution is to begin.

- **endPosition** – A number indicating at what character position in the string the substitution is to end.
- **value** – The replacement character or characters to be substituted.

Example

```
{
  "RecipeAction": {
    "Operation": "REPLACE_BETWEEN_POSITIONS",
    "Parameters": {
      "endPosition": "20",
      "sourceColumn": "nationality",
      "startPosition": "10",
      "value": "E"
    }
  }
}
```

REPLACE_TEXT

Replaces a specified sequence of characters with another.

Parameters

- **sourceColumn** – The name of an existing column.
- **pattern** – Character or characters or a regular expression, indicating which characters should be replaced in the source column.
- **value** – The replacement character or characters to be substituted.

Examples

```
{
  "RecipeAction": {
    "Operation": "REPLACE_TEXT",
    "Parameters": {
      "pattern": "x",
      "sourceColumn": "first_name",

```

```
        "value": "a"
      }
    }
  }
```

```
{
  "RecipeAction": {
    "Operation": "REPLACE_TEXT",
    "Parameters": {
      "pattern": "[0-9]",
      "sourceColumn": "nationality",
      "value": "!"
    }
  }
}
```

Data quality recipe steps

Use these data quality recipe steps to populate missing values, remove invalid data, or remove duplicates.

Topics

- [ADVANCED_DATATYPE_FILTER](#)
- [ADVANCED_DATATYPE_FLAG](#)
- [DELETE_DUPLICATE_ROWS](#)
- [EXTRACT_ADVANCED_DATATYPE_DETAILS](#)
- [FILL_WITH_AVERAGE](#)
- [FILL_WITH_CUSTOM](#)
- [FILL_WITH_EMPTY](#)
- [FILL_WITH_LAST_VALID](#)
- [FILL_WITH_MEDIAN](#)
- [FILL_WITH_MODE](#)
- [FILL_WITH_MOST_FREQUENT](#)
- [FILL_WITH_NULL](#)
- [FILL_WITH_SUM](#)

- [FLAG_DUPLICATE_ROWS](#)
- [FLAG_DUPLICATES_IN_COLUMN](#)
- [GET_ADVANCED_DATATYPE](#)
- [REMOVE_DUPLICATES](#)
- [REMOVE_INVALID](#)
- [REMOVE_MISSING](#)
- [REPLACE_WITH_AVERAGE](#)
- [REPLACE_WITH_CUSTOM](#)
- [REPLACE_WITH_EMPTY](#)
- [REPLACE_WITH_LAST_VALID](#)
- [REPLACE_WITH_MEDIAN](#)
- [REPLACE_WITH_MODE](#)
- [REPLACE_WITH_MOST_FREQUENT](#)
- [REPLACE_WITH_NULL](#)
- [REPLACE_WITH_ROLLING_AVERAGE](#)
- [REPLACE_WITH_ROLLING_SUM](#)
- [REPLACE_WITH_SUM](#)

ADVANCED_DATATYPE_FILTER

Filters the current source column based on advanced data type detection. For example, given a column that DataBrew has identified as containing zip codes, this transform can filter the column based on timezone. The details that you can extract depend on the pattern that is detected, as described in **Notes** below.

Parameters

- `sourceColumn` – The name of a string source column.
- `pattern` – The pattern to extract.
- `advancedDataType` – Can be one of Phone, Zip Code, Date Time, State, Credit Card, URL, Email, SSN, or Gender.
- `filter values` – List of string values that the user wants to filter the column based on.

- `strategy` – KEEP_ROWS or DISCARD_ROWS or CLEAR_FILTERS or CLEAR_OTHERS.
- `clearWithEmpty` – Boolean true or false, to clear rows with empty instead of null.

Notes

- If `advancedDataType` is **Phone**, then the pattern can be AREA_CODE, TIME_ZONE, or COUNTRY_CODE.
- If `advancedDataType` is **Zip Code**, then the pattern can be TIME_ZONE, COUNTRY, STATE, CITY, TYPE, or REGION.
- If `advancedDataType` is **Date Time**, then the pattern can be DAY, MONTH, MONTH_NAME, WEEK, QUARTER, or YEAR.
- If `advancedDataType` is **State**, then the pattern can be TIME_ZONE.
- If `advancedDataType` is **Credit Card**, then the pattern can be LENGTH or NETWORK.
- If `advancedDataType` is **URL**, then the pattern can be PROTOCOL, TLD, or DOMAIN.

Example

```
{
  "RecipeAction": {
    "Operation": "ADVANCED_DATATYPE_FILTER",
    "Parameters": {
      "pattern": "AREA_CODE",
      "sourceColumn": "phoneColumn",
      "advancedDataType": "Phone",
      "filterValues": ['Ohio'],
      "strategy": "KEEP_ROWS"
    }
  }
}
```

ADVANCED_DATATYPE_FLAG

Creates a new flag column based on the values for the current source column. For example, given a source column containing zip codes, this transform can be used to flag values as true or false based on a particular timezone. The details that you can extract depend on the pattern that is detected, as described in **Notes** below.

Parameters

- `sourceColumn` – The name of a string source column.
- `pattern` – The pattern to extract.
- `targetColumn` – The name of the target column.
- `advancedDataType` – Can be one of Phone, Zip Code, Date Time, State, Credit Card, URL, Email, SSN, or Gender.
- `filter values` – List of string values that the user wants to filter the column based on.
- `trueString` – The true value for the target column.
- `falseString` – The false value for the target column.

Notes

- If `advancedDataType` is **Phone**, then the pattern can be `AREA_CODE`, `TIME_ZONE`, or `COUNTRY_CODE`.
- If `advancedDataType` is **Zip Code**, then the pattern can be `TIME_ZONE`, `COUNTRY`, `STATE`, `CITY`, `TYPE`, or `REGION`.
- If `advancedDataType` is **Date Time**, then the pattern can be `DAY`, `MONTH`, `MONTH_NAME`, `WEEK`, `QUARTER`, or `YEAR`.
- If `advancedDataType` is **State**, then the pattern can be `TIME_ZONE`.
- If `advancedDataType` is **Credit Card**, then the pattern can be `LENGTH` or `NETWORK`.
- If `advancedDataType` is **URL**, then the pattern can be `PROTOCOL`, `TLD`, or `DOMAIN`.

Example

```
{
  "RecipeAction": {
    "Operation": "ADVANCED_DATATYPE_FLAG",
    "Parameters": {
      "pattern": "AREA_CODE",
      "sourceColumn": "phoneColumn",
      "advancedDataType": "Phone",
      "filterValues": ['Ohio'],
      "targetColumn": "targetColumnName",
      "trueString": "trueValue",
```

```
        "falseString": "falseValue"
    }
}
```

DELETE_DUPLICATE_ROWS

Deletes any row that is an exact match to an earlier row in the dataset. The initial occurrence is not deleted, because it doesn't match an earlier row.

Example

```
{
  "RecipeAction": {
    "Operation": "DELETE_DUPLICATE_ROWS"
  }
}
```

EXTRACT_ADVANCED_DATATYPE_DETAILS

Extracts details for the advanced data type. The details that you can extract depend on the pattern that is detected, as described in **Notes** below.

Parameters

- **sourceColumn** – The name of a string source column.
- **pattern** – The pattern to extract.
- **targetColumn** – The name of the target column.
- **advancedDataType** – Can be one of Phone, Zip Code, Date Time, State, Credit Card, URL, Email, SSN, or Gender.

Notes

- If **advancedDataType** is **Phone**, then the pattern can be AREA_CODE, TIME_ZONE, or COUNTRY_CODE.
- If **advancedDataType** is **Zip Code**, then the pattern can be TIME_ZONE, COUNTRY, STATE, CITY, TYPE, or REGION.

- If advancedDataType is **Date Time**, then the pattern can be DAY, MONTH, MONTH_NAME, WEEK, QUARTER, or YEAR.
- If advancedDataType is **State**, then the pattern can be TIME_ZONE.
- If advancedDataType is **Credit Card**, then the pattern can be LENGTH or NETWORK.
- If advancedDataType is **URL**, then the pattern can be PROTOCOL, TLD, or DOMAIN.

Example

```
{
  "RecipeAction": {
    "Operation": "EXTRACT_ADVANCED_DATATYPE_DETAILS",
    "Parameters": {
      "pattern": "TIMEZONE"
      "sourceColumn": "zipCode",
      "targetColumn": "timeZoneFromZipCode",
      "advancedDataType": "ZipCode"
    }
  }
}
```

FILL_WITH_AVERAGE

Returns a column with missing data replaced by the average of all values.

Parameters

- sourceColumn – The name of an existing column.

Example

```
{
  "RecipeAction": {
    "Operation": "FILL_WITH_AVERAGE",
    "Parameters": {
      "sourceColumn": "age"
    }
  }
}
```

```
}
```

FILL_WITH_CUSTOM

Returns a column with missing data replaced by a specific value.

Parameters

- `sourceColumn` – The name of an existing column.
- `columnDataType` – The data type for the column. This type must be date, number, boolean, unsupported, string, or timestamp.
- `value` – The custom value to fill in. The data type must match the value that you choose for `columnDataType`.

Example

```
{
  "RecipeAction": {
    "Operation": "FILL_WITH_CUSTOM",
    "Parameters": {
      "columnDataType": "string",
      "sourceColumn": "last_name",
      "value": "No last name provided"
    }
  }
}
```

FILL_WITH_EMPTY

Returns a column with missing data replaced by an empty string.

Parameters

- `sourceColumn` – The name of an existing column.

Example

```
{
  "RecipeAction": {
    "Operation": "FILL_WITH_EMPTY",
    "Parameters": {
      "sourceColumn": "wind_direction"
    }
  }
}
```

FILL_WITH_LAST_VALID

Returns a column with missing data replaced by the most recent valid value for that column.

Parameters

- `sourceColumn` – The name of an existing column.
- `columnDataType` – The data type for the column. This type must be date, number, boolean, unsupported, string, or timestamp.

Example

```
{
  "RecipeAction": {
    "Operation": "FILL_WITH_LAST_VALID",
    "Parameters": {
      "columnDataType": "string",
      "sourceColumn": "birth_date"
    }
  }
}
```

FILL_WITH_MEDIAN

Returns a column with missing data replaced by the median of all values.

Parameters

- `sourceColumn` – The name of an existing column.

Example

```
{
  "RecipeAction": {
    "Operation": "FILL_WITH_MEDIAN",
    "Parameters": {
      "sourceColumn": "age"
    }
  }
}
```

FILL_WITH_MODE

Returns a column with missing data replaced by the mode of all values.

You can also specify tie-breaker logic, where some of the values are identical. For example, consider the following values:

1 2 2 3 3 4

A modeType of MINIMUM causes FILL_WITH_MODE to return 2 as the mode value. If modeType is MAXIMUM, the mode is 3. For AVERAGE, the mode is 2.5.

Parameters

- **sourceColumn** – The name of an existing column.
- **modeType** – How to resolve tie values in the data. This value must be MINIMUM, NONE, AVERAGE, or MAXIMUM.

Example

```
{
  "RecipeAction": {
    "Operation": "FILL_WITH_MODE",
    "Parameters": {
      "modeType": "MAXIMUM",
      "sourceColumn": "age"
    }
  }
}
```

```
}  
}
```

FILL_WITH_MOST_FREQUENT

Returns a column with missing data replaced by the most frequent value.

Parameters

- `sourceColumn` – The name of an existing column.

Example

```
{  
  "RecipeAction": {  
    "Operation": "FILL_WITH_MOST_FREQUENT",  
    "Parameters": {  
      "sourceColumn": "position"  
    }  
  }  
}
```

FILL_WITH_NULL

Returns a column with data values replaced by null.

Parameters

- `sourceColumn` – The name of an existing column.

Example

```
{  
  "RecipeAction": {  
    "Operation": "FILL_WITH_NULL",  
    "Parameters": {  
      "sourceColumn": "rating"  
    }  
  }  
}
```

```
}  
}
```

FILL_WITH_SUM

Returns a column with missing data replaced by the sum of all values.

Parameters

- `sourceColumn` – The name of an existing column.

Example

```
{  
  "RecipeAction": {  
    "Operation": "FILL_WITH_SUM",  
    "Parameters": {  
      "sourceColumn": "age"  
    }  
  }  
}
```

FLAG_DUPLICATE_ROWS

Returns a new column with a specified value in each row that indicates whether that row is an exact match of an earlier row in the dataset. When matches are found, they are flagged as duplicates. The initial occurrence is not flagged, because it doesn't match an earlier row.

Parameters

- `trueString` – Value to be inserted if the row matches an earlier row.
- `falseString` – Value to be inserted if the row is unique.
- `targetColumn` – Name of the new column that is inserted in the dataset.

Example

```
{
```

```
    "RecipeAction": {
      "Operation": "FLAG_DUPLICATE_ROWS",
      "Parameters": {
        "trueString": "TRUE",
        "falseString": "FALSE",
        "targetColumn": "Flag"
      }
    }
  }
}
```

FLAG_DUPLICATES_IN_COLUMN

Returns a new column with a specified value in each row that indicates whether the value in the row's source column matches a value in an earlier row of the source column. When matches are found, they are flagged as duplicates. The initial occurrence is not flagged, because it doesn't match an earlier row.

Parameters

- **sourceColumn** – Name of the source column.
- **targetColumn** – Name of the target column.
- **trueString** – String to be inserted in the target column when a source column value duplicates an earlier value in that column.
- **falseString** – String to be inserted in the target column when a source column value is distinct from earlier values in that column.

Example

```
{
  "RecipeAction": {
    "Operation": "FLAG_DUPLICATES_IN_COLUMN",
    "Parameters": {
      "sourceColumn": "Name",
      "targetColumn": "Duplicate",
      "trueString": "TRUE",
      "falseString": "FALSE"
    }
  }
}
```

```
}
```

GET_ADVANCED_DATATYPE

Given a string column, identifies the advanced data type of the column, if any.

Parameters

- `columnName` – The name of the string column.

Example

```
{
  "RecipeAction": {
    "Operation": "GET_ADVANCED_DATATYPE",
    "Parameters": {
      "sourceColumn": "columnName"
    }
  }
}
```

REMOVE_DUPLICATES

Deletes an entire row, if a duplicate value is encountered in a selected source column.

Parameters

- `sourceColumn` – The name of an existing column.

Example

```
{
  "RecipeAction": {
    "Operation": "REMOVE_DUPLICATES",
    "Parameters": {
      "sourceColumn": "nationality"
    }
  }
}
```

REMOVE_INVALID

Deletes an entire row if an invalid value is encountered in a column of that row.

Parameters

- `sourceColumn` – The name of an existing column.
- `columnDataType` – The data type of the column.
- `advancedDataType` – Special data types that are detected by DataBrew in a column that has the data type `string`. The types that DataBrew can detect within a `string` column include SSN, Email, Phone Number, Gender, Credit Card, URL, IP Address, DateTime, Currency, ZipCode, Country, Region, State, and City.

Example

```
{
  "RecipeAction": {
    "Operation": "REMOVE_INVALID",
    "Parameters": {
      "columnDataType": "string",
      "sourceColumn": "help_url"
    }
  }
}
```

REMOVE_MISSING

Returns only the rows in which a specified column isn't missing data.

Parameters

- `sourceColumn` – The name of an existing column.

Example

```
{
  "RecipeAction": {
    "Operation": "REMOVE_MISSING",
```

```
    "Parameters": {
      "sourceColumn": "last_name"
    }
  }
}
```

REPLACE_WITH_AVERAGE

Replaces each invalid value in a column with the average of all other values.

Parameters

- `sourceColumn` – The name of an existing column.
- `columnDataType` – The data type of the column. This type must be number.

Example

```
{
  "RecipeAction": {
    "Operation": "REPLACE_WITH_AVERAGE",
    "Parameters": {
      "columnDataType": "number",
      "sourceColumn": "age"
    }
  }
}
```

REPLACE_WITH_CUSTOM

Replace detected entities with a custom value.

Parameters

- `sourceColumn` – The name of an existing column.
- `sourceColumns` – A list of existing column names.
- `columnDataType` – The data type of the column.
- `value` – The custom value to be used to replace invalid values.
- `advancedDataType` – Special data types that are detected by DataBrew in a column that has the data type `string`. The types that DataBrew can detect within a `string` column include

SSN, Email, Phone Number, Gender, Credit Card, URL, IP Address, DateTime, Currency, ZipCode, Country, Region, State, and City.

Note

Use either `sourceColumn` or `sourceColumns`, but not both.

Example

```
{
  "RecipeAction": {
    "Operation": "REPLACE_WITH_CUSTOM",
    "Parameters": {
      "columnDataType": "number",
      "sourceColumn": "",
      "sourceColumns": ["column1", "column2"],
      "value": 0
    }
  }
}
```

REPLACE_WITH_EMPTY

Replaces each invalid value in a column with an empty value.

Parameters

- `sourceColumn` – The name of an existing column.
- `columnDataType` – The data type of the column.
- `advancedDataType` – Special data types that are detected by DataBrew in a column that has the data type `string`. The types that DataBrew can detect within a `string` column include SSN, Email, Phone Number, Gender, Credit Card, URL, IP Address, DateTime, Currency, ZipCode, Country, Region, State, and City.

Example

```
{
  "RecipeAction": {
    "Operation": "REPLACE_WITH_EMPTY",
    "Parameters": {
      "columnDataType": "string",
      "sourceColumn": "nationality"
    }
  }
}
```

REPLACE_WITH_LAST_VALID

Replaces each invalid value in a column with the last valid value.

Parameters

- `sourceColumn` – The name of an existing column.
- `columnDataType` – The data type of the column.
- `advancedDataType` – Special data types that are detected by DataBrew in a column that has the data type `string`. The types that DataBrew can detect within a `string` column include SSN, Email, Phone Number, Gender, Credit Card, URL, IP Address, DateTime, Currency, ZipCode, Country, Region, State, and City.

Example

```
{
  "RecipeAction": {
    "Operation": "REPLACE_WITH_LAST_VALID",
    "Parameters": {
      "columnDataType": "number",
      "sourceColumn": "rating"
    }
  }
}
```

REPLACE_WITH_MEDIAN

Replaces each invalid value in a column with the median of all other values.

Parameters

- `sourceColumn` – The name of an existing column.
- `columnDataType` – The data type of the column. This type must be number.

Example

```
{
  "RecipeAction": {
    "Operation": "REPLACE_WITH_MEDIAN",
    "Parameters": {
      "columnDataType": "number",
      "sourceColumn": "games_won"
    }
  }
}
```

REPLACE_WITH_MODE

Replaces each invalid value in a column with the mode of all other values.

Parameters

- `sourceColumn` – The name of an existing column.
- `columnDataType` – The data type of the column. This type must be number.
- `modeType` – How to resolve tie values in the data. This value must be MINIMUM, NONE, AVERAGE, or MAXIMUM.

Example

```
{
  "RecipeAction": {
    "Operation": "REPLACE_WITH_MODE",
    "Parameters": {
      "columnDataType": "number",
      "modeType": "MAXIMUM",
      "sourceColumn": "height_cm"
    }
  }
}
```

```
}  
}
```

REPLACE_WITH_MOST_FREQUENT

Replaces each invalid value in a column with the most frequent column value.

Parameters

- `sourceColumn` – The name of an existing column.
- `columnDataType` – The data type of the column.
- `advancedDataType` – Special data types that are detected by DataBrew in a column that has the data type `string`. The types that DataBrew can detect within a `string` column include SSN, Email, Phone Number, Gender, Credit Card, URL, IP Address, DateTime, Currency, ZipCode, Country, Region, State, and City.

Example

```
{  
  "RecipeAction": {  
    "Operation": "REPLACE_WITH_MOST_FREQUENT",  
    "Parameters": {  
      "columnDataType": "string",  
      "sourceColumn": "wind_direction"  
    }  
  }  
}
```

REPLACE_WITH_NULL

Replaces each invalid value in a column with a null value.

Parameters

- `sourceColumn` – The name of an existing column.
- `columnDataType` – The data type of the column.
- `advancedDataType` – Special data types that are detected by DataBrew in a column that has the data type `string`. The types that DataBrew can detect within a `string` column include

SSN, Email, Phone Number, Gender, Credit Card, URL, IP Address, DateTime, Currency, ZipCode, Country, Region, State, and City.

Example

```
{
  "RecipeAction": {
    "Operation": "REPLACE_WITH_NULL",
    "Parameters": {
      "columnDataType": "number",
      "sourceColumn": "weight_kg"
    }
  }
}
```

REPLACE_WITH_ROLLING_AVERAGE

Replaces each value in a column with the rolling average from a previous "window" of rows.

Parameters

- **sourceColumn** – The name of an existing column.
- **columnDataType** – The data type of the column. This type must be number.
- **period** – The size of the window. For example, if period is 10, the rolling average is computed using the previous 10 rows.

Example

```
{
  "RecipeStep": {
    "Action": {
      "Operation": "REPLACE_WITH_ROLLING_AVERAGE",
      "Parameters": {
        "sourceColumn": "created_at",
        "columnDataType": "number",
        "period": "2"
      }
    }
  }
}
```

```
}
```

REPLACE_WITH_ROLLING_SUM

Replaces each value in a column with the rolling sum from a previous "window" of rows.

Parameters

- `sourceColumn` – The name of an existing column.
- `columnDataType` – The data type of the column. This type must be number.
- `period` – The size of the window. For example, if period is 10, the rolling sum is computed using the previous 10 rows.

Example

```
{
  "RecipeStep": {
    "Action": {
      "Operation": "REPLACE_WITH_ROLLING_SUM",
      "Parameters": {
        "sourceColumn": "created_at",
        "columnDataType": "number",
        "period": "2"
      }
    }
  }
}
```

REPLACE_WITH_SUM

Replaces each invalid value in a column with the sum of all other values.

Parameters

- `sourceColumn` – The name of an existing column.
- `columnDataType` – The data type of the column. This type must be number.

Example

```
{
  "RecipeAction": {
    "Operation": "REPLACE_WITH_SUM",
    "Parameters": {
      "columnDataType": "number",
      "sourceColumn": "games_won"
    }
  }
}
```

Personally identifiable information (PII) recipe steps

Use these recipe steps to perform transformations on personally identifiable information (PII) in a dataset.

Note

In addition to the recipe steps in this section, there are DataBrew recipe steps not designed specifically for PII that you can use to handle PII. An example is [DELETE](#), a basic column recipe step that deletes a column.

Topics

- [CRYPTOGRAPHIC_HASH](#)
- [DECRYPT](#)
- [DETERMINISTIC_DECRYPT](#)
- [DETERMINISTIC_ENCRYPT](#)
- [ENCRYPT](#)
- [MASK_CUSTOM](#)
- [MASK_DATE](#)
- [MASK_DELIMITER](#)
- [MASK_RANGE](#)
- [REPLACE_WITH_RANDOM_BETWEEN](#)
- [REPLACE_WITH_RANDOM_DATE_BETWEEN](#)

- [SHUFFLE_ROWS](#)

CRYPTOGRAPHIC_HASH

Applies an algorithm to hash values in the column.

Parameters

- `sourceColumns` – An array of existing columns.
- `secretId` – The ARN of the Secrets Manager secret key. The key used in the hash-based message authentication code (HMAC) prefix algorithm to hash the source columns, or `databrew!default` is the base64 decoded output for the value of the Secrets Manager secret key.
- `secretVersion` – Optional. Defaults to the latest secret version.
- `entityTypeFilter` – Optional array of [entity types](#). Can be used to encrypt only detected PII in free-text column.
- `createSecretIfMissing` – Optional boolean. If true will attempt to create the secret on behalf of the caller.
- `algorithm` – The algorithm used to hash your data. Valid enum values: MD5, SHA1, SHA256, SHA512, HMAC_MD5, HMAC_SHA1, HMAC_SHA256, HMAC_SHA512

Each option refers to a different hashing algorithm. Those options with the "HMAC" prefix refer to a keyed hashing algorithm, and require the `secretId` parameter. For options without the "HMAC" prefix, the `secretId` parameter is not required.

If you do not provide a hash algorithm, the service defaults to "HMAC_SHA256".

```
{
  "sourceColumns": ["phonenumber"],
  "secretId": "arn:aws:secretsmanager:us-east-1:012345678901:secret:mysecret",
  "entityTypeFilter": ["USA_ALL"]
}
```

When working in the interactive experience, in addition to the project's role, the console user must have permission to `secretsmanager:GetSecretValue` on the provided Secrets Manager secret.

Sample policy:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Resource": [
        "arn:aws:secretsmanager:us-east-1:012345678901:secret:mysecret"
      ]
    }
  ]
}
```

You may also opt to use the DataBrew-created default secret by passing `databrew!default` as `secretId` and parameter `createSecretIfMissing` as `true`. This is not recommended for production. Anyone with the **AwsGlueDataBrewFullAccessPolicy** role can use the default secret.

DECRYPT

You can use the DECRYPT transform to decrypt inside of DataBrew. Your data can also be decrypted outside of DataBrew with the AWS Encryption SDK. If the provided KMS key ARN does not match what was used to encrypt the column, the decrypt operation fails. For more information on the AWS Encryption SDK, see [What is the AWS Encryption SDK](#) in the *AWS Encryption SDK Developer Guide*.

Parameters

- `sourceColumns` – An array of existing columns.
- `kmsKeyArn` – The key ARN of the AWS Key Management Service key to use to decrypt the source columns. For more information on the key ARN, see [Key ARN](#) in the *AWS Key Management Service Developer Guide*.

```
{
  "sourceColumns": ["phonenumbers"],
```

```
"kmsKeyArn": "arn:aws:kms:us-east-1:012345678901:key/<kms-key-id>"
}
```

When working in the interactive experience, in addition to the project's role, the console user must have permission to `kms:GenerateDataKey` and `kms:Decrypt` on the provided KMS key.

Sample policy:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ],
      "Resource": [
        "arn:aws:kms:us-east-1:012345678901:key/kms-key-id"
      ]
    }
  ]
}
```

DETERMINISTIC_DECRYPT

Decrypts data encrypted with DETERMINISTIC_ENCRYPT.

This transformation is a no-op if the provided secret id and version does not match what was used to encrypt the column.

Parameters

- `sourceColumns` – An array of existing columns.
- `secretId` – The ARN of the Secrets Manager secret key to use to decrypt the source columns.
- `secretVersion` – Optional. Defaults to the latest secret version.

Example

```
{
  "sourceColumns": ["phonenumber"],
  "secretId": "arn:aws:secretsmanager:us-east-1:012345678901:secret:mysecret",
  "secretVersion": "adfe-1232-7563-3123"
}
```

When working in the interactive experience, in addition to the project's role, the console user must have permission to `secretsmanager:GetSecretValue` on the provided Secrets Manager secret.

Sample policy:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Resource": [
        "arn:aws:secretsmanager:us-east-1:012345678901:secret:mysecret"
      ]
    }
  ]
}
```

DETERMINISTIC_ENCRYPT

Encrypts the column using AES-GCM-SIV with a 256 bit key. Data encrypted with DETERMINISTIC_ENCRYPT can only be decrypted inside of DataBrew with the DETERMINISTIC_DECRYPT transform. This transform does not use AWS KMS or the AWS Encryption SDK, and instead uses the [AWS LC github library](#).

Can encrypt up to 400KB per cell. Does not preserve data type on decrypt.

Note

Note: Using a secret for more than a year is discouraged.

Parameters

- `sourceColumns` – An array of existing columns.
- `secretId` – The ARN of the Secrets Manager secret key to use to encrypt the source columns, or `databrew!default`.
- `secretVersion` – Optional. Defaults to the latest secret version.
- `entityTypeFilter` – Optional array of [entity types](#). Can be used to encrypt only detected PII in free-text column.
- `createSecretIfMissing` – Optional boolean. If true will attempt to create the secret on behalf of the caller.

Example

```
{
  "sourceColumns": ["phonenumber"],
  "secretId": "arn:aws:secretsmanager:us-east-1:012345678901:secret:mysecret",
  "secretVersion": "adfe-1232-7563-3123",
  "entityTypeFilter": ["USA_ALL"]
}
```

When working in the interactive experience, in addition to the project's role, the console user must have permission to `secretsmanager:GetSecretValue` on the provided Secrets Manager secret.

Sample policy

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
        "Action": [
            "secretsmanager:GetSecretValue"
        ],
        "Resource": [
            "arn:aws:secretsmanager:us-east-1:012345678901:secret:mysecret"
        ]
    }
}
```

ENCRYPT

Encrypts values in the source columns with the [AWS Encryption SDK](#). The DECRYPT transform can be used to decrypt inside of DataBrew. You can also decrypt the data outside of DataBrew using the AWS Encryption SDK.

The ENCRYPT transform can encrypt up to 128 MiB per cell. It will attempt to preserve the format on decryption. To preserve the data type, the data type metadata must serialize to less than 1KB. Otherwise, you must set the `preserveDataType` parameter to false. The data type metadata will be stored in plaintext in the encryption context. For more information on the encryption context, see [Encryption context](#) in the *AWS Key Management Service Developer Guide*.

Parameters

- `sourceColumns` – An array of existing columns.
- `kmsKeyArn` – The key ARN of the AWS Key Management Service key to use to encrypt the source columns. For more information on the key ARN, see [Key ARN](#) in the *AWS Key Management Service Developer Guide*.
- `entityTypeFilter` – Optional array of [entity types](#). Can be used to encrypt only detected PII in free-text column.
- `preserveDataType` – Optional boolean. Defaults to true. If false, the data type will not be stored.

In the following example, `entityTypeFilter` and `preserveDataType` are optional.

Example

```
{
```

```
"sourceColumns": ["phonenumber"],
"kmsKeyArn": "arn:aws:kms:us-east-1:012345678901:key/kms-key-id",
"entityTypeFilter": ["USA_ALL"],
"preserveDataType": "true"
}
```

When working in the interactive experience, in addition to the project's role, the console user must have permission to `kms:GenerateDataKey` on the provided AWS KMS key.

Sample policy:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:GenerateDataKey"
      ],
      "Resource": [
        "arn:aws:kms:us-east-1:012345678901:key/kms-key-id"
      ]
    }
  ]
}
```

MASK_CUSTOM

Masks characters that match a provided custom value.

Parameters

- `sourceColumns` – A list of existing column names.
- `maskSymbol` – A symbol that will be used to replace specified characters.
- `regex` – If true, treats `customValue` as a regex pattern to match.
- `customValue` – All occurrences (or regex matches) of `customValue` will be masked in the string.

- `entityTypeFilter` – Optional array of [entity types](#). Can be used to encrypt only detected PII in free-text column.

Example

```
// Mask all occurrences of 'amazon' in the column
{
  "RecipeAction": {
    "Operation": "MASK_CUSTOM",
    "Parameters": {
      "sourceColumns": ["company"],
      "maskSymbol": "#",
      "customValue": "amazon"
    }
  }
}
```

MASK_DATE

Masks components of a date with a user-specified mask symbol.

Parameters

- `sourceColumns` – A list of existing column names.
- `maskSymbol` – A symbol that will be used to replace specified characters.
- `redact` – An array of date component enums to mask. Valid enum values: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, MILLISECOND.
- `locale` – Optional IETF BCP 47 language tag. Defaults to en. The locale to use for date formatting.

Example

```
// Mask year
{
  "RecipeAction": {
    "Operation": "MASK_DATE",
    "Parameters": {
```

```
        "sourceColumns": ["birthday"],
        "maskSymbol": "#",
        "redact": ["YEAR"]
    }
}
```

MASK_DELIMITER

Masks characters between two delimiters with a user-specified masking symbol.

Parameters

- **sourceColumns** – A list of existing column names.
- **maskSymbol** – A symbol that will be used to replace specified characters.
- **startDelimiter** – A character indicating where masking is to begin. Omitting this parameter will apply the mask starting from the start of the string.
- **endDelimiter** – A character indicating where masking is to end. Omitting this parameter will apply the masking from the startDelimiter to the end of the string.
- **preserveDelimiters** – If true, applies mask to delimiters.
- **alphabet** – An array of character sets to preserve during masking. Valid enum values: SYMBOLS, WHITESPACE.
- **entityTypeFilter** – Optional array of [entity types](#). Can be used to encrypt only detected PII in free-text column.

Example

```
// Mask string between '<' and '>', ignoring white spaces, symbols, and lowercase letters
{
  "RecipeAction": {
    "Operation": "MASK_DELIMITER",
    "Parameters": {
      "sourceColumns": ["name"],
      "maskSymbol": "#",
      "startDelimiter": "<",
      "endDelimiter": ">",
    }
  }
}
```

```
        "preserveDelimiters": false,  
        "alphabet": ["WHITESPACE", "SYMBOLS"]  
    }  
}
```

MASK_RANGE

Masks characters between two positions with a user-specified masking symbol.

Parameters

- **sourceColumns** – A list of existing column names.
- **maskSymbol** – A symbol that will be used to replace specified characters.
- **start** – A number indicating at which character position the masking is to begin (0-indexed, inclusive). Negative indexing is allowed. Omitting this parameter will apply the mask from the beginning of the string until 'stop'.
- **stop** – A number indicating at which character position the masking is to end (0-indexed, exclusive). Negative indexing is allowed. Omitting this parameter will apply the mask from 'start' until the end of the string.
- **alphabet** – An array of character sets enums to preserve during masking. Valid enum values: SYMBOLS, WHITESPACE.
- **entityTypeFilter** – Optional array of [entity types](#). Can be used to encrypt only detected PII in free-text column.

Example

```
// Mask entire string  
{  
  "RecipeAction": {  
    "Operation": "MASK_RANGE",  
    "Parameters": {  
      "sourceColumns": ["firstName", "lastName"],  
      "maskSymbol": "#"  
    }  
  }  
}
```

REPLACE_WITH_RANDOM_BETWEEN

Replaces values with a random number.

Parameters

- `lowerBound` – The lower bound of the random number range.
- `sourceColumns` – A list of existing column names.
- `upperBound` – The upper bound of the random number range.

Example

```
{
  "RecipeAction": {
    "Operation": "REPLACE_WITH_RANDOM_BETWEEN",
    "Parameters": {
      "lowerBound": "1",
      "sourceColumns": ["column1", "column2"],
      "upperBound": "100"
    }
  }
}
```

REPLACE_WITH_RANDOM_DATE_BETWEEN

Replaces values with a random date.

Parameters

- `startDate` – The start of the range of dates from which a random date will be taken.
- `sourceColumns` – A list of existing column names.
- `endDate` – The end of the range of dates from which a random date will be taken.

Example

```
{
  "RecipeAction": {
```

```
    "Operation": "REPLACE_WITH_RANDOM_DATE_BETWEEN",
    "Parameters": {
      "startDate": "2020-12-12 12:12:12",
      "sourceColumns": ["column1", "column2"],
      "endDate": "2021-12-12 12:12:12"
    }
  }
}
```

SHUFFLE_ROWS

Shuffles values in a given column. The shuffling can occur with values grouped by a secondary column.

Parameters

- `sourceColumns` – An array of existing columns.
- `groupByColumns` – An array of columns to group the source columns by while shuffling.

Example

```
{
  "sourceColumns": ["age"],
  "*groupByColumns*": ["country"]
}
```

Outlier detection and handling recipe steps

Use these recipe steps to work with outliers in your data and perform advanced transformations on them..

Topics

- [FLAG_OUTLIERS](#)
- [REMOVE_OUTLIERS](#)
- [REPLACE_OUTLIERS](#)
- [RESCALE_OUTLIERS_WITH_Z_SCORE](#)
- [RESCALE_OUTLIERS_WITH_SKEW](#)

FLAG_OUTLIERS

Returns a new column containing a customizable value in each row that indicates if the source column value is an outlier.

Parameters

- `sourceColumn` – Specifies the name of an existing numeric column that might contain outliers.
- `targetColumn` – Specifies the name of a new column where the results of the outlier evaluation strategy is to be inserted.
- `outlierStrategy` – Specifies the approach to use in detecting outliers. Valid values include the following:
 - `Z_SCORE` – Identifies a value as an outlier when it deviates from the mean by more than the standard deviation threshold.
 - `MODIFIED_Z_SCORE` – Identifies a value as an outlier when it deviates from the median by more than the median absolute deviation threshold.
 - `IQR` – Identifies a values as an outlier when it falls beyond the first and last quartile of column data. The interquartile range (IQR) measures where the middle 50% of the data points are.
- `threshold` – Specifies the threshold value to use when detecting outliers. The `sourceColumn` value is identified as an outlier if the score that's calculated with the `outlierStrategy` exceeds this number. The default is 3.
- `trueString` – Specifies the string value to use if an outlier is detected. The default is "True".
- `falseString` – Specifies the string value to use if no outlier is detected. The default is "False".

The following examples display syntax for a single [RecipeAction](#) operation. A *recipe* contains at least one [RecipeStep](#) operation, and a recipe step contains at least one recipe action. A *recipe action* runs the data transform that you specify. A group of recipe actions run in sequential order to create the final dataset.

JSON

The following shows an example `RecipeAction` to use as member of an example `RecipeStep` for a DataBrew [Recipe](#), using JSON syntax. For syntax examples showing a list of recipe actions, see [Defining a recipe structure](#).

Example in JSON

```
{
  "Action": {
    "Operation": "FLAG_OUTLIERS",
    "Parameters": {
      "sourceColumn": "name-of-existing-column",
      "targetColumn": "name-of-new-column",
      "outlierStrategy": "IQR",
      "threshold": "1.5",
      "trueString": "Yes",
      "falseString": "No"
    }
  }
}
```

For more information on using this recipe action in an API operation, see [CreateRecipe](#) or [UpdateRecipe](#). You can use these and other API operations in your own code.

YAML

The following shows an example `RecipeAction` to use as member of an example `RecipeStep` for a DataBrew [Recipe](#), using YAML syntax. For syntax examples showing a list of recipe actions, see [Defining a recipe structure](#).

Example in YAML

```
- Action:
  Operation: FLAG_OUTLIERS
  Parameters:
    sourceColumn: name-of-existing-column
    targetColumn: name-of-new-column
    outlierStrategy: IQR
    trueString: Outlier
    falseString: No
    threshold: '1.5'
```

For more information on using this recipe action in an API operation, see [CreateRecipe](#) or [UpdateRecipe](#). You can use these and other API operations in your own code.

REMOVE_OUTLIERS

Removes data points that classify as outliers, based on the settings in the parameters.

Parameters

- `sourceColumn` – Specifies the name of an existing numeric column that might contain outliers.
- `outlierStrategy` – Specifies the approach to use in detecting outliers. Valid values include the following:
 - `Z_SCORE` – Identifies a value as an outlier when it deviates from the mean by more than the standard deviation threshold.
 - `MODIFIED_Z_SCORE` – Identifies a value as an outlier when it deviates from the median by more than the median absolute deviation threshold.
 - `IQR` – Identifies a values as an outlier when it falls beyond the first and last quartile of column data. The interquartile range (IQR) measures where the middle 50% of the data points are.
- `threshold` – Specifies the threshold value to use when detecting outliers. The `sourceColumn` value is identified as an outlier if the score that's calculated with the `outlierStrategy` exceeds this number. The default is 3.
- `removeType` – Specifies the way to remove the data. Valid values include `DELETE_ROWS` and `CLEAR`.
- `trimValue` – Specifies whether to remove all or some of the outliers. This Boolean value defaults to `FALSE`.
 - `FALSE` – Removes all outliers
 - `TRUE` – Removes outliers that rank outside of the percentile threshold specified in `minValue` and `maxValue`.
- `minValue` – Indicates the minimum percentile value for the outlier range. Valid range is 0–100.
- `maxValue` – Indicates the maximum percentile value for the outlier range. Valid range is 0–100.

The following examples display syntax for a single [RecipeAction](#) operation. A *recipe* contains at least one [RecipeStep](#) operation, and a recipe step contains at least one recipe action. A *recipe action* runs the data transform that you specify. A group of recipe actions run in sequential order to create the final dataset.

JSON

The following shows an example `RecipeAction` to use as member of an example `RecipeStep` for a DataBrew [Recipe](#), using JSON syntax. For syntax examples showing a list of recipe actions, see [Defining a recipe structure](#).

Example in JSON

```
{
  "Action": {
    "Operation": "REMOVE_OUTLIERS",
    "Parameters": {
      "sourceColumn": "name-of-existing-column",
      "outlierStrategy": "Z_SCORE",
      "threshold": "3",
      "removeType": "DELETE_ROWS",
      "trimValue": "TRUE",
      "minValue": "5",
      "maxValue": "95"
    }
  }
}
```

For more information on using this recipe action in an API operation, see [CreateRecipe](#) or [UpdateRecipe](#). You can use these and other API operations in your own code.

YAML

The following shows an example `RecipeAction` to use as member of an example `RecipeStep` for a DataBrew [Recipe](#), using YAML syntax. For syntax examples showing a list of recipe actions, see [Defining a recipe structure](#).

Example in YAML

```
- Action:
  Operation: REMOVE_OUTLIERS
  Parameters:
    sourceColumn: name-of-existing-column
    outlierStrategy: Z_SCORE
    threshold: '3'
    removeType: DELETE_ROWS
    trimValue: 'TRUE'
    minValue: '5'
```

```
maxValue: '95'
```

For more information on using this recipe action in an API operation, see [CreateRecipe](#) or [UpdateRecipe](#). You can use these and other API operations in your own code.

REPLACE_OUTLIERS

Updates the data point values that classify as outliers, based on the settings in the parameters.

Parameters

- **sourceColumn** – Specifies the name of an existing numeric column that might contain outliers.
- **outlierStrategy** – Specifies the approach to use in detecting outliers. Valid values include the following:
 - **Z_SCORE** – Identifies a value as an outlier when it deviates from the mean by more than the standard deviation threshold.
 - **MODIFIED_Z_SCORE** – Identifies a value as an outlier when it deviates from the median by more than the median absolute deviation threshold.
 - **IQR** – Identifies a values as an outlier when it falls beyond the first and last quartile of column data. The interquartile range (IQR) measures where the middle 50% of the data points are.
- **threshold** – Specifies the threshold value to use when detecting outliers. The **sourceColumn** value is identified as an outlier if the score that's calculated with the **outlierStrategy** exceeds this number. The default is 3.
- **replaceType** – Specifies the method to use when replacing outliers. Valid values include the following:
 - **WINSORIZE_VALUES** – Specifies using the minimum and maximum percentile to cap the values.
 - **REPLACE_WITH_CUSTOM**
 - **REPLACE_WITH_EMPTY**
 - **REPLACE_WITH_NULL**
 - **REPLACE_WITH_MODE**
 - **REPLACE_WITH_AVERAGE**
 - **REPLACE_WITH_MEDIAN**
 - **REPLACE_WITH_SUM**

- `REPLACE_WITH_MAX`
- `modeType` – Indicates the type of modal function to use when `replaceType` is `REPLACE_WITH_MODE`. Valid values include the following: `MIN`, `MAX`, and `AVERAGE`.
- `minValue` – Indicates the minimum percentile value for the outlier range that is to be applied when `trimValue` is used. Valid range is 0–100.
- `maxValue` – Indicates the maximum percentile value for the outlier range that is to be applied when `trimValue` is used. . Valid range is 0–100.
- `value` – Specifies the value to insert when using `REPLACE_WITH_CUSTOM`.
- `trimValue` – Specifies whether to remove all or some of the outliers. This Boolean value is set to `TRUE` when `replaceType` is `REPLACE_WITH_NULL`, `REPLACE_WITH_MODE`, or `WINSORIZE_VALUES`. It defaults to `FALSE` for all others.
 - `FALSE` – Removes all outliers
 - `TRUE` –Removes outliers that rank outside of the percentile cap threshold specified in `minValue` and `maxValue`.

The following examples display syntax for a single [RecipeAction](#) operation. A *recipe* contains at least one [RecipeStep](#) operation, and a recipe step contains at least one recipe action. A *recipe action* runs the data transform that you specify. A group of recipe actions run in sequential order to create the final dataset.

JSON

The following shows an example `RecipeAction` to use as member of an example `RecipeStep` for a DataBrew [Recipe](#), using JSON syntax. For syntax examples showing a list of recipe actions, see [Defining a recipe structure](#).

Example in JSON

```
{
  "Action": {
    "Operation": "REPLACE_OUTLIERS",
    "Parameters": {
      "maxValue": "95",
      "minValue": "5",
      "modeType": "AVERAGE",
      "outlierStrategy": "Z_SCORE",
      "replaceType": "REPLACE_WITH_MODE",
```

```
        "sourceColumn": "name-of-existing-column",
        "threshold": "3",
        "trimValue": "TRUE"
      }
    }
  }
```

For more information on using this recipe action in an API operation, see [CreateRecipe](#) or [UpdateRecipe](#). You can use these and other API operations in your own code.

YAML

The following shows an example `RecipeAction` to use as member of an example `RecipeStep` for a DataBrew [Recipe](#), using YAML syntax. For syntax examples showing a list of recipe actions, see [Defining a recipe structure](#).

Example in YAML

```
- Action:
  Operation: REMOVE_OUTLIERS
  Parameters:
    sourceColumn: name-of-existing-column
    outlierStrategy: Z_SCORE
    threshold: '3'
    replaceType: REPLACE_WITH_MODE
    modeType: AVERAGE
    minValue: '5'
    maxValue: '95'
    trimValue: 'TRUE'
```

For more information on using this recipe action in an API operation, see [CreateRecipe](#) or [UpdateRecipe](#). You can use these and other API operations in your own code.

RESCALE_OUTLIERS_WITH_Z_SCORE

Returns a new column with a rescaled outlier value in each row, based on the settings in the parameters. This action also applies Z-score normalization to linearly scale data values to have a mean (μ) of 0 and standard deviation (σ) of 1. We recommend this action for handling outliers.

Parameters

- `sourceColumn` – Specifies the name of an existing numeric column that might contain outliers.

- **targetColumn** – Specifies the name of an existing numeric column that might contain outliers.
- **outlierStrategy** – Specifies the approach to use in detecting outliers. Valid values include the following:
 - **Z_SCORE** – Identifies a value as an outlier when it deviates from the mean by more than the standard deviation threshold.
 - **MODIFIED_Z_SCORE** – Identifies a value as an outlier when it deviates from the median by more than the median absolute deviation threshold.
 - **IQR** – Identifies a values as an outlier when it falls beyond the first and last quartile of column data. The interquartile range (IQR) measures where the middle 50% of the data points are.
- **threshold** – The threshold value to use when detecting outliers. The **sourceColumn** value is identified as an outlier if the score that's calculated with the **outlierStrategy** exceeds this number. The default is 3.

The following examples display syntax for a single [RecipeAction](#) operation. A *recipe* contains at least one [RecipeStep](#) operation, and a recipe step contains at least one recipe action. A *recipe action* runs the data transform that you specify. A group of recipe actions run in sequential order to create the final dataset.

JSON

The following shows an example [RecipeAction](#) to use as member of an example [RecipeStep](#) for a DataBrew [Recipe](#) operation, using JSON syntax. For syntax examples showing a list of recipe actions, see [Defining a recipe structure](#).

Example in JSON

```
{
  "Action": {
    "Operation": "RESCALE_OUTLIERS_WITH_Z_SCORE",
    "Parameters": {
      "sourceColumn": "name-of-existing-column",
      "targetColumn": "name-of-new-column",
      "outlierStrategy": "Z_SCORE",
      "threshold": "3"
    }
  }
}
```

For more information on using this recipe action in an API operation, see [CreateRecipe](#) or [UpdateRecipe](#). You can use these and other API operations in your own code.

YAML

The following shows an example `RecipeAction` to use as member of an example `RecipeStep` for a DataBrew [Recipe](#) operation, using YAML syntax. For syntax examples showing a list of recipe actions, see [Defining a recipe structure](#).

Example in YAML

```
- Action:
  Operation: REMOVE_OUTLIERS
  Parameters:
    sourceColumn: name-of-existing-column
    targetColumn: name-of-new-column
    outlierStrategy: Z_SCORE
    threshold: '3'
```

For more information on using this recipe action in an API operation, see [CreateRecipe](#) or [UpdateRecipe](#). You can use these and other API operations in your own code.

RESCALE_OUTLIERS_WITH_SKEW

Returns a new column with a rescaled outlier value in each row, based on the settings in the parameters. This action works to reduce distribution skewness by applying the specified log or root transform. We recommend this action for handling skewed data.

Parameters

- `sourceColumn` – Specifies the name of an existing numeric column that might contain outliers.
- `targetColumn` – Specifies the name of an existing numeric column that might contain outliers.
- `outlierStrategy` – Specifies the approach to use in detecting outliers. Valid values include the following:
 - `Z_SCORE` – Identifies a value as an outlier when it deviates from the mean by more than the standard deviation threshold.
 - `MODIFIED_Z_SCORE` – Identifies a value as an outlier when it deviates from the median by more than the median absolute deviation threshold.

- **IQR** – Identifies a values as an outlier when it falls beyond the first and last quartile of column data. The interquartile range (IQR) measures where the middle 50% of the data points are.
- **threshold** – Specifies the threshold value to use when detecting outliers. The `sourceColumn` value is identified as an outlier if the score that's calculated with the `outlierStrategy` exceeds this number. The default is 3.
- **skewFunction** – Specifies the method to use when replacing outliers. Valid values include the following:
 - **LOG** – Applies a strong transformation to reduce positive and negative skew. This is a natural logarithm (2.718281828).
 - **ROOT** (with `value = 3`) – Applies a fairly strong transformation to reduce positive and negative skew. (Cube root)
 - **ROOT** (with `value = 2`) – Applies a moderate transformation to reduce positive skew only. (Square root)
 - **SQUARE** – Applies a moderate transformation to reduce negative skew. (Square)
 - **Custom transform** – Applies the specified LOG or ROOT transform using the custom number provided in the `value` parameter.
- **value** – Specifies the value to use for the custom transform. If `skewFunction` is LOG, this value represents the base of the log. If `skewFunction` is ROOT, this value represents the power of the root.

The following examples display syntax for a single [RecipeAction](#) operation. A *recipe* contains at least one [RecipeStep](#) operation, and a recipe step contains at least one recipe action. A *recipe action* runs the data transform that you specify. A group of recipe actions run in sequential order to create the final dataset.

JSON

The following shows an example `RecipeAction` to use as member of an example `RecipeStep` for a DataBrew [Recipe](#), using JSON syntax. For syntax examples showing a list of recipe actions, see [Defining a recipe structure](#).

Example in JSON

```
{
  "Action": {
    "Operation": "RESCALE_OUTLIERS_WITH_SKEW",
```

```
    "Parameters": {
      "outlierStrategy": "Z_SCORE",
      "threshold": "3",
      "skewFunction": "ROOT",
      "sourceColumn": "name-of-existing-column",
      "targetColumn": "name-of-new-column",
      "value": "4"
    }
  }
}
```

For more information on using this recipe action in an API operation, see [CreateRecipe](#) or [UpdateRecipe](#). You can use these and other API operations in your own code.

YAML

The following shows an example `RecipeAction` to use as member of an example `RecipeStep` for a DataBrew [Recipe](#), using YAML syntax. For syntax examples showing a list of recipe actions, see [Defining a recipe structure](#).

Example in YAML

```
- Action:
  Operation: RESCALE_OUTLIERS_WITH_SKEW
  Parameters:
    outlierStrategy: Z_SCORE
    threshold: '3'
    skewFunction: ROOT
    sourceColumn: name-of-existing-column
    targetColumn: name-of-new-column
    value: '4'
```

For more information on using this recipe action in an API operation, see [CreateRecipe](#) or [UpdateRecipe](#). You can use these and other API operations in your own code.

Column structure recipe steps

Use these column structure recipe steps to modify the column structure of your data.

Topics

- [BOOLEAN_OPERATION](#)

- [CASE_OPERATION](#)
- [FLAG_COLUMN_FROM_NULL](#)
- [FLAG_COLUMN_FROM_PATTERN](#)
- [MERGE](#)
- [SPLIT_COLUMN_BETWEEN_DELIMITER](#)
- [SPLIT_COLUMN_BETWEEN_POSITIONS](#)
- [SPLIT_COLUMN_FROM_END](#)
- [SPLIT_COLUMN_FROM_START](#)
- [SPLIT_COLUMN_MULTIPLE_DELIMITER](#)
- [SPLIT_COLUMN_SINGLE_DELIMITER](#)
- [SPLIT_COLUMN_WITH_INTERVALS](#)

BOOLEAN_OPERATION

Create a new column, based on the result of logical condition IF. Return true value if the boolean expression is true, false value if the boolean expression is false, or return a custom value.

Parameters

- `trueValueExpression` – Result when the condition is met.
- `falseValueExpression` – Result when the condition is not met.
- `valueExpression` – Boolean condition.
- `withExpressions` – Configuration for aggregate results.
- `targetColumn` – A name for the newly created column.

You can use constant values, column references, and aggregate results in `trueValueExpression`, `falseValueExpression` and `valueExpression`.

Example: Constant values

Values that remain unchanged, like a number or a sentence.

```
{
  "RecipeStep": {
    "Action": {
```

```

    "Operation": "BOOLEAN_OPERATION",
    "Parameters": {
      "trueValueExpression": "It is true.",
      "falseValueExpression": "It is false.",
      "valueExpression": "`column.1` < 2000",
      "targetColumn": "result.column"
    }
  }
}

```

Example: Column references

Values that are columns in the dataset.

```

{
  "RecipeStep": {
    "Action": {
      "Operation": "BOOLEAN_OPERATION",
      "Parameters": {
        "trueValueExpression": "`column.2`",
        "falseValueExpression": "`column.3`",
        "valueExpression": "`column.1` < `column.4`",
        "targetColumn": "result.column"
      }
    }
  }
}

```

Example: Aggregate results

Values that are calculated by aggregate functions. An aggregate function performs a calculation on a column, and returns a single value.

```

{
  "RecipeStep": {
    "Action": {
      "Operation": "BOOLEAN_OPERATION",
      "Parameters": {
        "trueValueExpression": "`:mincolumn.2`",

```

```

        "falseValueExpression": "`:maxcolumn.3`",
        "valueExpression": "`column.1` < `:avgcolumn.4`",
        "withExpressions": "[{\"name\": \"mincolumn.2\", \"value\": \"min(`column.2`)\",
        \"type\": \"aggregate\"}, {\"name\": \"maxcolumn.3\", \"value\": \"max(`column.3`)\", \"type\": \"aggregate\"}, {\"name\": \"avgcolumn.4\", \"value\": \"avg(`column.4`)\", \"type\": \"aggregate\"}]",
        "targetColumn": "result.column"
    }
}
}
}

```

Users need to convert the JSON to a string by escaping.

Note that the parameter names in `trueValueExpression`, `falseValueExpression`, and `valueExpression` must match the names in `withExpressions`. To use the aggregate results from some columns, you need to create parameters for them and provide the aggregate functions.

Example:

```

{
  "RecipeStep": {
    "Action": {
      "Operation": "BOOLEAN_OPERATION",
      "Parameters": {
        "trueValueExpression": "It is true.",
        "falseValueExpression": "It is false.",
        "valueExpression": "`column.1` < 2000",
        "targetColumn": "result.column"
      }
    }
  }
}

```

Example: and/or

You can use `and` and `or` to combine multiple conditions.

```

{
  "RecipeStep": {
    "Action": {
      "Operation": "BOOLEAN_OPERATION",

```

```
    "Parameters": {
      "trueValueExpression": "It is true.",
      "falseValueExpression": "It is false.",
      "valueExpression": "`column.1` < 2000 and `column.2` >= `column.3",
      "targetColumn": "result.column"
    }
  }
}
{
  "RecipeStep": {
    "Action": {
      "Operation": "BOOLEAN_OPERATION",
      "Parameters": {
        "trueValueExpression": "`column.4`",
        "falseValueExpression": "`column.5`",
        "valueExpression": "startsWith(`column1`, 'value1') or endsWith(`column2`, 'value2')",
        "targetColumn": "result.column"
      }
    }
  }
}
```

Valid aggregate functions

The table below shows all of the valid aggregate functions that can be used in a boolean operation.

Column type	Condition	valueExpression	withExpressions	Return value
Numeric	Sum	`:sum.column.1`	[{ "name": "sum.column.1", "value": "sum(`column.1`)", "type":	Returns the sum of column.1

Column type	Condition	valueExpression	withExpressions	Return value
			<pre>"aggregate" { }</pre>	
	Mean	<code>`:mean.column.1`</code>	<pre>[{ "name": "mean.column.1", "value": "avg(`column.1`)", "type": "aggregate" }</pre>	Returns the mean of column.1

Column type	Condition	valueExpression	withExpressions	Return value
	Mean absolute deviation	`:meanabsolute_deviation.column.1`	<pre>[{ "name": "meanabsolute_deviation.column.1", "value": "mean_absolute_deviation(`column.1`)", ", "type": "aggregate" }]</pre>	Returns the mean absolute deviation of column.1
	Median	`:median.column.1`	<pre>[{ "name": "median.column.1", "value": "median(`column.1`)", ", "type": "aggregate" }]</pre>	Returns the median of column.1

Column type	Condition	valueExpression	withExpressions	Return value
	Product	`:product .column.1`	<pre>[{ "name": "product. column.1", "value": "product(`column.1 `)", "type": "aggregat e" }]</pre>	Returns the product of column.1
	Standard deviation	`:standar ddeviatio n.column.1`	<pre>[{ "name": "standard deviation .column.1 ", "value": "stddev(` column.1`)", "type": "aggregat e" }]</pre>	Returns the standard deviation of column.1

Column type	Condition	valueExpression	withExpressions	Return value
	Variance	<code>`:variance.column.1`</code>	<pre>[{ "name": "variance .column.1", "value": "variance (`column. 1`)", "type": "aggregate" }]</pre>	Returns the variance of column.1
	Standard error of mean	<code>`:standarderrorofmean.column.1`</code>	<pre>[{ "name": "standard errorofmean.column .1", "value": "standard _error_of _mean(`co lumn.1`)", "type": "aggregate" }]</pre>	Returns the standard error of mean of column.1

Column type	Condition	valueExpression	withExpressions	Return value
	Skewness	`:skewness.column.1`	<pre>[{ "name": "skewness .column.1", "value": "skewness (`column. 1`)", "type": "aggregate" }]</pre>	Returns the skewness of column.1
	Kurtosis	`:kurtosis.column.1`	<pre>[{ "name": "kurtosis .column.1", "value": "kurtosis (`column. 1`)", "type": "aggregate" }]</pre>	Returns the kurtosis of column.1

Column type	Condition	valueExpression	withExpressions	Return value
Datetime/ Numeric/Text	Count	`:count.column.1`	<pre>[{ "name": "count.column.1", "value": "count(`column.1`)", " "type": "aggregate" }]</pre>	Returns the total number of rows in column.1
	Count distinct	`:countdistinct.column.1`	<pre>[{ "name": "count.column.1", "value": "count(distinct `column.1`)", " "type": "aggregate" }]</pre>	Returns the total number of distinct rows in column.1

Column type	Condition	valueExpression	withExpressions	Return value
	Min	`:min.column.1`	<pre>[{ "name": "min.colu mn.1", "value": "min(`col umn.1`)", "type": "aggregat e" }]</pre>	Returns the minimum value of column.1
	Max	`:max.col umn.1`	<pre>[{ "name": "max.colu mn.1", "value": "max(`col umn.1`)", "type": "aggregat e" }]</pre>	Returns the maximum value of column.1

Valid conditions in a valueExpression

The table below shows supported conditions and the value expressions you can use.

Column type	Condition	valueExpression	Description
String	Contains	contains(`column`, 'text')	Condition to test if the value in column contains text
	Does not contain	!contains(`column`, 'text')	Condition to test if the value in column is does not contain text
	Matches	matches(`column`, 'pattern')	Condition to test if the value in column matches pattern
	Does not match	!matches(`column`, 'pattern')	Condition to test if the value in column does not match pattern
	Starts with	startsWith(`column`, 'text')	Condition to test if the value in column starts with text
	Does not start with	!startsWith(`column`, 'text')	Condition to test if the value in column does not start with text
	Ends with	endsWith(`column`, 'text')	Condition to test if the value in column ends with text
	Does not end with	!endsWith(`column`, 'text')	Condition to test if the value in column does not end with text

Column type	Condition	valueExpression	Description
Numeric	Less than	<code>`column` < number</code>	Condition to test if the value in column is less than number
	Less than or equal to	<code>`column` <= number</code>	Condition to test if the value in column is less than or equal to number
	Greater than	<code>`column` > number</code>	Condition to test if the value in column is greater than number
	Greater than or equal to	<code>`column` >= number</code>	Condition to test if the value in column is greater than or equal to number
	Is between	<code>isBetween(`column`, minNumber, maxNumber)</code>	Condition to test if the value in column is in between minNumber and maxNumber
	Is not between	<code>!isBetween(`column`, minNumber, maxNumber)</code>	Condition to test if the value in column is not in between minNumber and maxNumber
Boolean	Is true	<code>`column` = TRUE</code>	Condition to test if the value in column is boolean TRUE

Column type	Condition	valueExpression	Description
	Is false	<code>`column` = FALSE</code>	Condition to test if the value in column is boolean FALSE
Date/Timestamp	Earlier than	<code>`column` < 'date'</code>	Condition to test if the value in column is earlier than date
	Earlier than or equal to	<code>`column` <= 'date'</code>	Condition to test if the value in column is earlier than or equal to date
	Later than	<code>`column` > 'date'</code>	Condition to test if the value in column is later than date
	Later than or equal to	<code>`column` >= 'date'</code>	Condition to test if the value in column is later than or equal to date
String/Numeric/Date/Timestamp	Is exactly	<code>`column` = 'value'</code>	Condition to test if the value in column is exactly value
	Is not	<code>`column` != 'value'</code>	Condition to test if the value in column is not value
	Is missing	<code>isMissing(`column`)</code>	Condition to test if the value in column is missing
	Is not missing	<code>!isMissing(`column`)</code>	Condition to test if the value in column is not missing

Column type	Condition	valueExpression	Description
	Is valid	isValid(`column`, datatype)	Condition to test if the value in column is valid (the value is of datatype or it can be converted to datatype)
	Is not valid	!isValid(`column`, datatype)	Condition to test if the value in column is not valid (the value is of datatype or it can be converted to datatype)
Nested	Is missing	isMissing(`column`)	Condition to test if the value in column is missing
	Is not missing	!isMissing(`column`)	Condition to test if the value in column is not missing
	Is valid	isValid(`column`, datatype)	Condition to test if the value in column is valid (the value is of datatype or it can be converted to datatype)
	Is not valid	!isValid(`column`, datatype)	Condition to test if the value in column is not valid (the value is of datatype or it can be converted to datatype)

CASE_OPERATION

Create a new column, based on the result of logical condition CASE. The case operation goes through case conditions and returns a value when the first condition is met. Once a condition is true, the operation stops reading and returns the result. If no conditions are true, it returns the default value.

Parameters

- `valueExpression` – Conditions.
- `withExpressions` – Configuration for aggregate results.
- `targetColumn` – Name for the newly created column.

Example

```
{
  "RecipeStep": {
    "Action": {
      "Operation": "CASE_OPERATION",
      "Parameters": {
        "valueExpression": "case when `column11` < `column.2` then 'result1' when
`column2` < 'value2' then 'result2' else 'high' end",
        "targetColumn": "result.column"
      }
    }
  }
}
```

Valid aggregate functions

The table below shows all of the valid aggregate functions that can be used in a case operation.

Column type	Condition	valueExpression	withExpressions	Return value
Numeric	Sum	`:sum.col umn.1`	[{ "name":	Returns the sum of column.1

Column type	Condition	valueExpression	withExpressions	Return value
			<pre>"sum.column.1", "value": "sum(`column.1`)", "type": "aggregate" }]</pre>	
	Mean	<code>`:mean.column.1`</code>	<pre>[{ "name": "mean.column.1", "value": "avg(`column.1`)", "type": "aggregate" }]</pre>	Returns the mean of column.1

Column type	Condition	valueExpression	withExpressions	Return value
	Mean absolute deviation	<code>`:meanabsolute_deviation.column.1`</code>	<pre>[{ "name": "meanabsolute_deviation.column.1", "value": "mean_absolute_deviation(`column.1`)" , "type": "aggregate" }]</pre>	Returns the mean absolute deviation of column.1
	Median	<code>`:median.column.1`</code>	<pre>[{ "name": "median.column.1", "value": "median(`column.1`)" , "type": "aggregate" }]</pre>	Returns the median of column.1

Column type	Condition	valueExpression	withExpressions	Return value
	Product	`:product .column.1`	<pre>[{ "name": "product. column.1", "value": "product(`column.1 `)", "type": "aggregat e" }]</pre>	Returns the product of column.1
	Standard deviation	`:standar ddeviatio n.column.1`	<pre>[{ "name": "standard deviation .column.1 ", "value": "stddev(` column.1`)", "type": "aggregat e" }]</pre>	Returns the standard deviation of column.1

Column type	Condition	valueExpression	withExpressions	Return value
	Variance	<code>`:variance.column.1`</code>	<pre>[{ "name": "variance .column.1", "value": "variance (`column. 1`)", "type": "aggregate" }]</pre>	Returns the variance of column.1
	Standard error of mean	<code>`:standarderrorofmean.column.1`</code>	<pre>[{ "name": "standard errorofme an.column .1", "value": "standard _error_of _mean(`co lumn.1`)", "type": "aggregate" }]</pre>	Returns the standard error of mean of column.1

Column type	Condition	valueExpression	withExpressions	Return value
	Skewness	`:skewness.column.1`	<pre>[{ "name": "skewness .column.1 ", "value": "skewness (`column. 1`)", "type": "aggregate" }]</pre>	Returns the skewness of column.1
	Kurtosis	`:kurtosis.column.1`	<pre>[{ "name": "kurtosis .column.1 ", "value": "kurtosis (`column. 1`)", "type": "aggregate" }]</pre>	Returns the kurtosis of column.1

Column type	Condition	valueExpression	withExpressions	Return value
Datetime/ Numeric/Text	Count	`:count.column.1`	<pre>[{ "name": "count.column.1", "value": "count(`column.1`)", "type": "aggregate" }]</pre>	Returns the total number of rows in column.1
	Count distinct	`:countdistinct.column.1`	<pre>[{ "name": "count.column.1", "value": "count(distinct `column.1`)", "type": "aggregate" }]</pre>	Returns the total number of distinct rows in column.1

Column type	Condition	valueExpression	withExpressions	Return value
	Min	`:min.column.1`	[{ "name": "min.colu mn.1", "value": "min(`col umn.1`)", "type": "aggregat e" }]	Returns the minimum value of column.1
	Max	`:max.col umn.1`	[{ "name": "max.colu mn.1", "value": "max(`col umn.1`)", "type": "aggregat e" }]	Returns the maximum value of column.1

Valid conditions in a valueExpression

The table below shows supported conditions and the value expressions you can use.

Column type	Condition	valueExpression	Description
String	Contains	contains(`column`, 'text')	Condition to test if the value in column contains text
	Does not contain	!contains(`column`, 'text')	Condition to test if the value in column is does not contain text
	Matches	matches(`column`, 'pattern')	Condition to test if the value in column matches pattern
	Does not match	!matches(`column`, 'pattern')	Condition to test if the value in column does not match pattern
	Starts with	startsWith(`column`, 'text')	Condition to test if the value in column starts with text
	Does not start with	!startsWith(`column`, 'text')	Condition to test if the value in column does not start with text
	Ends with	endsWith(`column`, 'text')	Condition to test if the value in column ends with text
	Does not end with	!endsWith(`column`, 'text')	Condition to test if the value in column does not end with text

Column type	Condition	valueExpression	Description
Numeric	Less than	<code>`column` < number</code>	Condition to test if the value in column is less than number
	Less than or equal to	<code>`column` <= number</code>	Condition to test if the value in column is less than or equal to number
	Greater than	<code>`column` > number</code>	Condition to test if the value in column is greater than number
	Greater than or equal to	<code>`column` >= number</code>	Condition to test if the value in column is greater than or equal to number
	Is between	<code>isBetween(`column`, minNumber, maxNumber)</code>	Condition to test if the value in column is in between minNumber and maxNumber
	Is not between	<code>!isBetween(`column`, minNumber, maxNumber)</code>	Condition to test if the value in column is not in between minNumber and maxNumber
Boolean	Is true	<code>`column` = TRUE</code>	Condition to test if the value in column is boolean TRUE

Column type	Condition	valueExpression	Description
	Is false	<code>`column` = FALSE</code>	Condition to test if the value in column is boolean FALSE
Date/Timestamp	Earlier than	<code>`column` < 'date'</code>	Condition to test if the value in column is earlier than date
	Earlier than or equal to	<code>`column` <= 'date'</code>	Condition to test if the value in column is earlier than or equal to date
	Later than	<code>`column` > 'date'</code>	Condition to test if the value in column is later than date
	Later than or equal to	<code>`column` >= 'date'</code>	Condition to test if the value in column is later than or equal to date
String/Numeric/Date/Timestamp	Is exactly	<code>`column` = 'value'</code>	Condition to test if the value in column is exactly value
	Is not	<code>`column` != 'value'</code>	Condition to test if the value in column is not value
	Is missing	<code>isMissing(`column`)</code>	Condition to test if the value in column is missing
	Is not missing	<code>!isMissing(`column`)</code>	Condition to test if the value in column is not missing

Column type	Condition	valueExpression	Description
	Is valid	isValid(`column`, datatype)	Condition to test if the value in column is valid (the value is of datatype or it can be converted to datatype)
	Is not valid	!isValid(`column`, datatype)	Condition to test if the value in column is not valid (the value is of datatype or it can be converted to datatype)
Nested	Is missing	isMissing(`column`)	Condition to test if the value in column is missing
	Is not missing	!isMissing(`column`)	Condition to test if the value in column is not missing
	Is valid	isValid(`column`, datatype)	Condition to test if the value in column is valid (the value is of datatype or it can be converted to datatype)
	Is not valid	!isValid(`column`, datatype)	Condition to test if the value in column is not valid (the value is of datatype or it can be converted to datatype)

FLAG_COLUMN_FROM_NULL

Creates a new column, based on the presence of null values in an existing column.

Parameters

- `sourceColumn` – The name of an existing column.
- `targetColumn` – The name of a new column to be created.
- `flagType` – A value that must be set to `Null values`.
- `trueString` – A value for the new column, if a null value is found in the source. If no value is specified, the default is `True`.
- `falseString` – A value for the new column, if a non-null value is found in the source. If no value is specified, the default is `False`.

Example

```
{
  "RecipeAction": {
    "Operation": "FLAG_COLUMN_FROM_NULL",
    "Parameters": {
      "flagType": "Null values",
      "sourceColumn": "weight_kg",
      "targetColumn": "is_weight_kg_missing"
    }
  }
}
```

FLAG_COLUMN_FROM_PATTERN

Creates a new column, based on the presence of a user-specified pattern in an existing column.

Parameters

- `sourceColumn` – The name of an existing column.
- `targetColumn` – The name of a new column to be created.
- `flagType` – A value that must be set to `Pattern`.
- `pattern` – A regular expression, indicating the pattern to be evaluated.

- **trueString** – A value for the new column, if a null value is found in the source. If no value is specified, the default is `True`.
- **falseString** – A value for the new column, if a non-null value is found in the source. If no value is specified, the default is `False`.

Example

```
{
  "RecipeAction": {
    "Operation": "FLAG_COLUMN_FROM_PATTERN",
    "Parameters": {
      "falseString": "No",
      "flagType": "Pattern",
      "pattern": "N.*",
      "sourceColumn": "wind_direction",
      "targetColumn": "northerly",
      "trueString": "yes"
    }
  }
}
```

MERGE

Merges two or more columns into a new column.

Parameters

- **sourceColumns** – A JSON-encoded string representing a list of one or more columns to be merged.
- **delimiter** – An optional separator between the values, to appear in the target column.
- **targetColumn** – The name of the merged column to be created.

Example

```
{
  "RecipeAction": {
    "Operation": "MERGE",
    "Parameters": {
```

```

        "delimiter": " ",
        "sourceColumns": "[\"first_name\", \"last_name\"]",
        "targetColumn": "Merged Column 1"
    }
}

```

SPLIT_COLUMN_BETWEEN_DELIMITER

Splits a column into three new columns, according to a beginning and ending delimiter.

Parameters

- **sourceColumn** – The name of an existing column.
- **patternOption1** – A JSON-encoded string representing one or more characters that indicate the first delimiter.
- **patternOption2** – A JSON-encoded string representing one or more characters that indicate the second delimiter.
- **pattern** – One or more characters to use as a separator, when splitting the data.
- **includeInSplit** – If true, includes the pattern in the new column; otherwise, the pattern is discarded.

Example

```

{
  "RecipeAction": {
    "Operation": "SPLIT_COLUMN_BETWEEN_DELIMITER",
    "Parameters": {
      "patternOption1": "{\"pattern\": \"H\", \"includeInSplit\": true}",
      "patternOption2": "{\"pattern\": \"M\", \"includeInSplit\": true}",
      "sourceColumn": "last_name"
    }
  }
}

```

SPLIT_COLUMN_BETWEEN_POSITIONS

Splits a column into three new columns, according to offsets that you specify.

Parameters

- `sourceColumn` – The name of an existing column.
- `startPosition` – The character position where the split is to begin.
- `endPosition` – The character position where the split is to end.

Example

```
{
  "RecipeAction": {
    "Operation": "SPLIT_COLUMN_BETWEEN_POSITIONS",
    "Parameters": {
      "endPosition": "12",
      "sourceColumn": "last_name",
      "startPosition": "2"
    }
  }
}
```

SPLIT_COLUMN_FROM_END

Splits a column into two new columns, at an offset from the end of the string.

Parameters

- `sourceColumn` – The name of an existing column.
- `position` – The character position, from the right end of the string, where the split is to occur.

Example

```
{
  "RecipeAction": {
    "Operation": "SPLIT_COLUMN_FROM_END",
    "Parameters": {
      "position": "1",
      "sourceColumn": "nationality"
    }
  }
}
```

```
}  
}
```

SPLIT_COLUMN_FROM_START

Splits a column into two new columns, at an offset from the beginning of the string.

Parameters

- `sourceColumn` – The name of an existing column.
- `position` – The character position, from the left end of the string, where the split is to occur.

Example

```
{  
  "RecipeAction": {  
    "Operation": "SPLIT_COLUMN_FROM_START",  
    "Parameters": {  
      "position": "1",  
      "sourceColumn": "first_name"  
    }  
  }  
}
```

SPLIT_COLUMN_MULTIPLE_DELIMITER

Splits a column according to multiple delimiters.

Parameters

- `sourceColumn` – The name of an existing column.
- `patternOptions` – A JSON-encoded string representing one or more patterns that determine the split criteria.
- `pattern` – One or more characters to use as a separator, when splitting the data.
- `limit` – How many splits to perform. The minimum is 1; the maximum is 20.
- `includeInSplit` – If true, includes the pattern in the new column; otherwise, the pattern is discarded.

Example

```
{
  "RecipeAction": {
    "Operation": "SPLIT_COLUMN_MULTIPLE_DELIMITER",
    "Parameters": {
      "limit": "1",
      "patternOptions": "[{\"pattern\":\"\\\",\\\",\\\"includeInSplit\\\":true},{\"pattern\":\"\\\" \\\",\\\"includeInSplit\\\":true}]",
      "sourceColumn": "description"
    }
  }
}
```

SPLIT_COLUMN_SINGLE_DELIMITER

Splits a column into one or more new columns, according to a specific delimiter.

Parameters

- **sourceColumn** – The name of an existing column.
- **pattern** – One or more characters to use as a separator, when splitting the data.
- **limit** – How many splits to perform. The minimum is 1; the maximum is 20.
- **includeInSplit** – If true, includes the pattern in the new column; otherwise, the pattern is discarded.

Example

```
{
  "RecipeAction": {
    "Operation": "SPLIT_COLUMN_SINGLE_DELIMITER",
    "Parameters": {
      "includeInSplit": "true",
      "limit": "1",
      "pattern": "/",
      "sourceColumn": "info_url"
    }
  }
}
```

```
}
```

SPLIT_COLUMN_WITH_INTERVALS

Splits a column at intervals of n characters, where you specify n .

Parameters

- `sourceColumn` – The name of an existing column.
- `startPosition` – The character position where the split is to begin.
- `interval` – The number of characters to skip before the next split.

Example

```
{
  "RecipeAction": {
    "Operation": "SPLIT_COLUMN_WITH_INTERVALS",
    "Parameters": {
      "interval": "4",
      "sourceColumn": "nationality",
      "startPosition": "1"
    }
  }
}
```

Column formatting recipe steps

Use column formatting recipe steps to change the format of the data in your columns.

Topics

- [NUMBER_FORMAT](#)
- [FORMAT_PHONE_NUMBER](#)

NUMBER_FORMAT

Returns a column in which a numeric value is converted into a formatted string.

Parameters

- `sourceColumn` – String. The name of an existing column.
- `decimalPlaces` – Integer. The value of number of digits after the decimal separator.
- `numericDecimalSeparator` – String. One of the following values indicating the decimal separator:
 - `"."`
 - `","`
- `numericThousandSeparator` – String. One of the following values indicating the thousand separator:
 - `null`. Indicates that a thousand separator isn't enabled.
 - `","`
 - `" "`
 - `"."`
 - `"\""`
- `numericAbbreviatedUnit` – String. One of the following values indicating the abbreviation unit:
 - `null`. Indicates that an abbreviation unit isn't enabled.
 - `"THOUSAND"`
 - `"MILLION"`
 - `"BILLION"`
 - `"TRILLION"`
- `numericUnitAbbreviation` – String. One of the following values or any custom value, indicating unit abbreviation:
 - `null`. Indicates that unit abbreviation isn't enabled.

Abbreviation unit	Options
Thousands	K, k, M, thousand, custom
Million	M, m, MM, million, custom
Billion	B, bn, billion, custom

Abbreviation unit	Options
Trillion	T, tn, trillion, custom

Example

```
{
  "RecipeAction": {
    "Operation": "NUMBER_FORMAT",
    "Parameters": {
      "sourceColumn": "income",
      "decimalPlaces": "2",
      "numericDecimalSeparator": ".",
      "numericThousandSeparator": ",",
      "numericAbbreviatedUnit": "THOUSAND",
      "numericUnitAbbreviation": "K"
    }
  }
}
```

FORMAT_PHONE_NUMBER

Returns a column in which a phone number string is converted into a formatted value.

Parameters

- `sourceColumn` – The name of an existing column.
- `phoneNumberFormat` – The format to convert the phone number to. If no format is specified, the default is E.164, an internationally-recognized standard phone number format. Valid values include the following:
 - E164 (omit the period after E)
- `defaultRegion` – A valid region code consisting of two or three uppercase letters that specifies the region for the phone number when no country code is present in the number itself. At most, one of `defaultRegion` or `defaultRegionColumn` can be provided.
- `defaultRegionColumn` – The name of a column of the [advanced data type](#) Country. The region code from the specified column is used to determine the country code for the phone

number when no country code is present in the number itself. At most, one of `defaultRegion` or `defaultRegionColumn` can be provided.

Notes

- Inputs that can't be formatted to a valid phone number remain unmodified.
- If no default region is provided, and a phone number doesn't start with a plus symbol (+) and country calling code, the phone number isn't formatted.

Example

Example: Fixed default region

```
{
  "Action": {
    "Operation": "FORMAT_PHONE_NUMBER",
    "Parameters": {
      "sourceColumn": "Phone Number",
      "defaultRegion": "US"
    }
  }
}
```

Example: Default region column option

```
{
  "Action": {
    "Operation": "FORMAT_PHONE_NUMBER",
    "Parameters": {
      "sourceColumn": "Phone Number",
      "defaultRegionColumn": "Country Code"
    }
  }
}
```

Data structure recipe steps

Use these recipe steps to tabulate and summarize data from different perspectives, or to perform advanced functions.

Topics

- [NEST_TO_ARRAY](#)
- [NEST_TO_MAP](#)
- [NEST_TO_STRUCT](#)
- [UNNEST_ARRAY](#)
- [UNNEST_MAP](#)
- [UNNEST_STRUCT](#)
- [UNNEST_STRUCT_N](#)
- [GROUP_BY](#)
- [JOIN](#)
- [PIVOT](#)
- [SCALE](#)
- [TRANSPOSE](#)
- [UNION](#)
- [UNPIVOT](#)

NEST_TO_ARRAY

Converts user-selected columns into array values. The order of the selected columns is maintained while creating the resultant array. The different column data types are typecast to a common type that supports the data types of all columns.

Parameters

- `sourceColumns` — List of the source columns.
- `targetColumn` — The name of the target column.
- `removeSourceColumns` — Contains the value `true` or `false` to indicate whether or not the user wants to remove the selected source columns.

Example

```
{
  "RecipeAction": {
    "Operation": "NEST_TO_ARRAY",
    "Parameters": {
      "sourceColumns": "[\"age\", \"weight_kg\", \"height_cm\"]",
      "targetColumn": "columnName",
      "removeSourceColumns": "true"
    }
  }
}
```

NEST_TO_MAP

Converts user-selected columns into key-value pairs, each with a key representing the column name and a value representing the row value. The order of the selected column is not maintained while creating the resultant map. The different column data types are typecast to a common type that supports the data types of all columns.

Parameters

- **sourceColumns** — List of the source columns.
- **targetColumn** — The name of the target column.
- **removeSourceColumns** — Contains the value `true` or `false` to indicate whether or not the user wants to remove the selected source columns.

Example

```
{
  "RecipeAction": {
    "Operation": "NEST_TO_MAP",
    "Parameters": {
      "sourceColumns": "[\"age\", \"weight_kg\", \"height_cm\"]",
      "targetColumn": "columnName",
      "removeSourceColumns": "true"
    }
  }
}
```

NEST_TO_STRUCT

Converts user-selected columns into key-value pairs, each with a key representing the column name and a value representing the row value. The order of the selected columns and the data type of each column are maintained in the resultant struct.

Parameters

- **sourceColumns** — List of the source columns.
- **targetColumn** — The name of the target column.
- **removeSourceColumns** — Contains the value `true` or `false` to indicate whether or not the user wants to remove the selected source columns.

Example

```
{
  "RecipeAction": {
    "Operation": "NEST_TO_STRUCT",
    "Parameters": {
      "sourceColumns": "[\"age\", \"weight_kg\", \"height_cm\"]",
      "targetColumn": "columnName",
      "removeSourceColumns": "true"
    }
  }
}
```

UNNEST_ARRAY

Unnests a column of type `array` into a new column. If the array contains more than one value, then a row corresponding to each element is generated. This function only unnests one level of an array column.

Parameters

- **sourceColumn** — The name of an existing column. This column must be of `struct` type.
- **targetColumn** — Name of the target column that is generated.

Example

```
{
  "RecipeAction": {
    "Operation": "UNNEST_ARRAY",
    "Parameters": {
      "sourceColumn": "address",
      "targetColumn": "address"
    }
  }
}
```

UNNEST_MAP

Unnests a column of type map and generates a column for the key and value. If there is more than one key-value pair, a row corresponding to each key value would be generated. This function only unnests one level of a map column.

Parameters

- **sourceColumn** — The name of an existing column. This column must be of struct type.
- **removeSourceColumn** — If true, the source column is deleted after the function is complete.
- **targetColumn** — If provided, each of the generated column will start with this as the prefix.

Example

```
{
  "RecipeAction": {
    "Operation": "UNNEST_MAP",
    "Parameters": {
      "sourceColumn": "address",
      "removeSourceColumn": "false",
      "targetColumn": "address"
    }
  }
}
```

UNNEST_STRUCT

Unnest a column of type struct and generates a column for each of the keys present in the struct. This function only unnests struct level one.

Parameters

- **sourceColumn** — The name of an existing column. This column must be of struct type.
- **removeSourceColumn** — If true, the source column is deleted after the function is complete.
- **targetColumn** — If provided, each of the generated column will start with this as the prefix.

Example

```
{
  "RecipeAction": {
    "Operation": "UNNEST_STRUCT",
    "Parameters": {
      "sourceColumn": "address",
      "removeSourceColumn": "false"
      "targetColumn": "add"
    }
  }
}
```

UNNEST_STRUCT_N

Creates a new column for each field of a selected column of type struct.

For example, given the following struct:

```
user {
  name: "Ammy"
  address: {
    state: "CA",
    zipcode: 12345
  }
}
```

This function creates 3 columns:

user.name	user.address.state	user.address.zipcode
Ammy	CA	12345

Parameters

- `sourceColumns` — List of the source columns.
- `regexColumnSelector` — A regular expression to select the columns to unnest.
- `removeSourceColumn` — A Boolean value. If true, then remove the source column; otherwise keep it.
- `unnestLevel` — The number of levels to unnest.
- `delimiter` — The delimiter is used in the newly created column name to separate the different levels of the struct. For example: if the delimiter is `/`, the column name will be in this form: `"user/address/state"`.
- `conditionExpressions` — Condition expressions.

Example

```
{
  "RecipeAction": {
    "Operation": "UNNEST_STRUCT_N",
    "Parameters": {
      "sourceColumns": "[\"address\"]",
      "removeSourceColumn": "true",
      "unnestLevel": "2",
      "delimiter": "/"
    }
  }
}
```

GROUP_BY

Summarizes the data by grouping rows by one or more columns, and then applying an aggregation function to each group.

Parameters

- **sourceColumns** — A JSON-encoded string representing a list of columns that form the basis of each group.
- **groupByAggFunctions** — A JSON-encoded string representing a list of aggregation function to apply. (If you don't want aggregation, specify UNAGGREGATED.)
- **useNewDataFrame** — If true, the results from GROUP_BY are made available in the project session, replacing its current contents.

Example

```
[
  {
    "Action": {
      "Operation": "GROUP_BY",
      "Parameters": {
        "groupByAggFunctionOptions": "[{\"sourceColumnName\":\"all_votes\",
        \"targetColumnName\":\"all_votes_count\", \"targetColumnDataType\":\"number\",
        \"functionName\":\"COUNT\"}]",
        "sourceColumns": "[\"year\", \"state_name\"]",
        "useNewDataFrame": "true"
      }
    }
  }
]
```

JOIN

Performs a join operation on two datasets.

Parameters

- **joinKeys** — A JSON-encoded string representing a list of columns from each dataset to act as join keys.
- **joinType** — The type of join to perform. Must be one of: INNER_JOIN | LEFT_JOIN | RIGHT_JOIN | OUTER_JOIN | LEFT_EXCLUDING_JOIN | RIGHT_EXCLUDING_JOIN | OUTER_EXCLUDING_JOIN

- **leftColumns** — A JSON-encoded string representing a list of columns from the current active dataset.
- **rightColumns** — A JSON-encoded string representing a list of columns from another (secondary) dataset to join to the current one.
- **secondInputLocation** — An Amazon S3 URL that resolves to the data file for the secondary dataset.
- **secondaryDatasetName** — The name of the secondary dataset.

Example

```
{
  "Action": {
    "Operation": "JOIN",
    "Parameters": {
      "joinKeys": "[{\"key\":\"assembly_session\",\"value\":\"assembly_session\"},{\"key\":\"state_code\",\"value\":\"state_code\"}]",
      "joinType": "INNER_JOIN",
      "leftColumns": "[\"year\",\"assembly_session\",\"state_code\",\"state_name\",\"all_votes\",\"yes_votes\",\"no_votes\",\"abstain\",\"idealpoint_estimate\",\"affinityscore_usa\",\"affinityscore_russia\",\"affinityscore_china\",\"affinityscore_india\",\"affinityscore_brazil\",\"affinityscore_israel\"]",
      "rightColumns": "[\"assembly_session\",\"vote_id\",\"resolution\",\"state_code\",\"state_name\",\"member\",\"vote\"]",
      "secondInputLocation": "s3://databrew-public-datasets-us-east-1/votes.csv",
      "secondaryDatasetName": "votes"
    }
  }
}
```

PIVOT

Converts all the row values in a selected column into individual columns with values.



Parameters

- **sourceColumn** — The name of an existing column. The column can have a maximum of 10 distinct values.
- **valueColumn** — The name of an existing column. The column can have a maximum of 10 distinct values.
- **aggregateFunction** — The name of an aggregation function. If you don't want aggregation, use the keyword `COLLECT_LIST`.

Example

```
{
  "Action": {
    "Operation": "PIVOT",
    "Parameters": {
      "aggregateFunction": "SUM",
      "sourceColumn": "state_name",
      "valueColumn": "all_votes"
    }
  }
}
```

SCALE

Scales or normalizes the range of data in a numeric column.

Parameters

- **sourceColumn** — The name of an existing column.
- **strategy** — The operation to be applied to the column values:
 - **MIN_MAX** — Rescales the values into a range of [0,1].
 - **SCALE_BETWEEN** — Rescales the values into a range of two specified values.
 - **MEAN_NORMALIZATION** — Rescales the data to have a mean (μ) of 0 and standard deviation (σ) of 1 within a range of [-1, 1].
 - **Z_SCORE** — Linearly scales data values to have a mean (μ) of 0 and standard deviation (σ) of 1. Best for handling outliers.

- **targetColumn** — The name of a column to contain the results.

Example

```
{
  "Action": {
    "Operation": "NORMALIZATION",
    "Parameters": {
      "sourceColumn": "all_votes",
      "strategy": "MIN_MAX",
      "targetColumn": "all_votes_normalized"
    }
  }
}
```

TRANSPOSE

Converts all selected rows to columns and columns to rows.

Column 1	Column A	Column B	Column C
Row A	Value A	Value B	Value C
Row B	Value A1	Value B1	Value C1



New column	Row A	Row B
Column A	Value A	Value A1
Column B	Value B	Value B1
Column C	Value C	Value C1

Parameters

- **pivotColumns** — A JSON-encoded string representing a list of columns whose rows will be converted to column names.
- **valueColumns** — A JSON-encoded string representing a list of one or more columns to be converted to rows.
- **aggregateFunction** — The name of an aggregation function. If you don't want aggregation, use the keyword `COLLECT_LIST`.

- **newColumn** — The column to hold transposed columns as values.

Example

```
{
  "Action": {
    "Operation": "TRANSPOSE",
    "Parameters": {
      "pivotColumns": "[\"Teacher\"]",
      "valueColumns": "[\"Tom\", \"John\", \"Harry\"]",
      "aggregateFunction": "COLLECT_LIST",
      "newColumn": "Student"
    }
  }
}
```

UNION

Combines the rows from two or more datasets into a single result.

Parameters

- **datasetsColumns** — A JSON-encoded string representing a list of all the columns in the datasets.
- **secondaryDatasetNames** — A JSON-encoded string representing a list of one or more secondary datasets.
- **secondaryInputs** — A JSON-encoded string representing a list of Amazon S3 buckets and object key names that tell DataBrew where to find the secondary dataset(s).
- **targetColumnNames** — A JSON-encoded string representing a list of column names for the results.

Example

```
{
  "Action": {
```

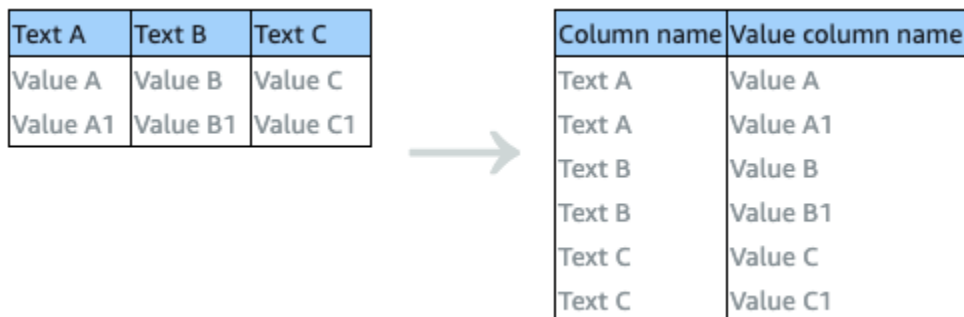
```

    "Operation": "UNION",
    "Parameters": {
      "datasetsColumns": "[\"assembly_session\", \"state_code\", \"state_name\", \"year\", \"all_votes\", \"yes_votes\", \"no_votes\", \"abstain\", \"idealpoint_estimate\", \"affinityscore_usa\", \"affinityscore_russia\", \"affinityscore_china\", \"affinityscore_india\", \"affinityscore_brazil\", \"affinityscore_israel\"]\", [\"assembly_session\", \"state_code\", \"state_name\", null, null, null, null, null, null, null, null, null, null, null, null]]\",
      \"secondaryDatasetNames\": \"[\"votes\"]\",
      \"secondaryInputs\": \"[{\"S3InputDefinition\": {\"Bucket\": \"databrew-public-datasets-us-east-1\", \"Key\": \"votes.csv\"}}]\",
      \"targetColumnNames\": \"[\"assembly_session\", \"state_code\", \"state_name\", \"year\", \"all_votes\", \"yes_votes\", \"no_votes\", \"abstain\", \"idealpoint_estimate\", \"affinityscore_usa\", \"affinityscore_russia\", \"affinityscore_china\", \"affinityscore_india\", \"affinityscore_brazil\", \"affinityscore_israel\"]\"
    }
  }
}

```

UNPIVOT

Converts all the column values in a selected row into individual rows with values.



Parameters

- **sourceColumns** — A JSON-encoded string representing a list of one or more columns to be unpivoted.
- **unpivotColumn** — The value column for the unpivot operation.
- **valueColumn** — The column to hold unpivoted values.

Example

```
{
  "Action": {
    "Operation": "UNPIVOT",
    "Parameters": {
      "sourceColumns": "[\"idealpoint_estimate\"]",
      "unpivotColumn": "unpivoted_idealpoint_estimate",
      "valueColumn": "unpivoted_column_values"
    }
  }
}
```

Data science recipe steps

Use these recipe steps to tabulate and summarize data from different perspectives, or to perform advanced transformations.

Topics

- [BINARIZATION](#)
- [BUCKETIZATION](#)
- [CATEGORICAL_MAPPING](#)
- [ONE_HOT_ENCODING](#)
- [SCALE](#)
- [SKEWNESS](#)
- [TOKENIZATION](#)

BINARIZATION

Takes all the values in a selected numeric source column, compares them to a threshold value, and outputs a new column with a 1 or 0 for each row.

Parameters

- **sourceColumn** – The name of an existing column.
- **targetColumn** – The name of the new column to be created.
- **threshold** – Number indicating the threshold for assigning the value of 0 or 1.

flip – Option to flip binary assignment so that lower values are assigned 1 and higher values are assigned 0. When the flip parameter is true, values lower than or equal to the threshold value result in 1, and values greater than the threshold value result in 0.

Example

```
{
  "Action": {
    "Operation": "BINARIZATION",
    "Parameters": {
      "sourceColumn": "level",
      "targetColumn": "bin",
      "threshold": "100.0",
      "flip": "false"
    }
  }
}
```

BUCKETIZATION

Bucketization (called Binning in the console) takes the items in a column of numeric values, groups them into bins defined by numeric ranges, and outputs a new column that displays the bin for each row. Bucketization can be done using splits or percentage. The first example below uses splits and the second example uses a percentage.

Parameters

- **sourceColumn** – The name of an existing column.

targetColumn – The name of the new column to be created.

bucketNames – List of bucket names.

splits – List of bucket levels. Buckets are consecutive, and an upper bound for a bucket will be a lower bound for the next bucket.

percentage – Each bucket will be described as a percentage.

Example using splits

```
{
  "Action": {
    "Operation": "BUCKETIZATION",
    "Parameters": {
      "sourceColumn": "level",
      "targetColumn": "bin",
      "bucketNames": "[\"Bin1\\\", \"Bin2\\\", \"Bin3\\\"]",
      "splits": "[\"-Infinity\\\", \"2\\\", \"20\\\", \"Infinity\\\"]"
    }
  }
}
```

Example using a percentage

```
{
  "Action": {
    "Operation": "BUCKETIZATION",
    "Parameters": {
      "sourceColumn": "level",
      "targetColumn": "bin",
      "bucketNames": "[\"Bin1\\\", \"Bin2\\\"]",
      "percentage": "50"
    }
  }
}
```

CATEGORICAL_MAPPING

Maps one or more categorical values to numeric or other values

Parameters

- **sourceColumn** – The name of an existing column.
- **categoryMap** – A JSON-encoded string representing a map of values to categories.
- **deleteOtherRows** – If `true`, all non-mapped rows will be removed from the dataset.
- **other** – When provided, all non-mapped values will be replaced by this value.

`keepOthers` – If true, all non-mapped values will remain the same.

`mapType` – The data type of the mapped column.

`targetColumn` – The name of a column to contain the results.

Example

```
{
  "Action": {
    "Operation": "CATEGORICAL_MAPPING",
    "Parameters": {
      "categoryMap": "{\"United States of America\":\"1\",\"Canada\":\"2\",\"Cuba\":\"3\",\"Haiti\":\"4\",\"Dominican Republic\":\"5\"}",
      "deleteOtherRows": "false",
      "keepOthers": "true",
      "mapType": "NUMERIC",
      "sourceColumn": "state_name",
      "targetColumn": "state_name_mapped"
    }
  }
}
```

ONE_HOT_ENCODING

Creates n numerical columns, where n is the number of unique values in a selected categorical variable.

For example, consider a column named `shirt_size`. Shirts are available in small, medium, large, or extra large. The column data might look like the following.

```
shirt_size
-----
L
XL
M
S
M
M
S
```

```
XL
M
L
XL
M
```

In this scenario, there are four distinct values for `shirt_size`. Therefore, `ONE_HOT_ENCODING` generates four new columns. Each new column is named `shirt_size_x`, where `x` represents a distinct `shirt_size` value.

The results of `shirt_size` and the four generated columns look like this.

<code>shirt_size</code>	<code>shirt_size_S</code>	<code>shirt_size_M</code>	<code>shirt_size_L</code>	<code>shirt_size_XL</code>
L	0	0	1	0
XL	0	0	0	1
M	0	1	0	0
S	1	0	0	0
M	0	1	0	0
M	0	1	0	0
S	1	0	0	0
XL	0	0	0	1
M	0	1	0	0
L	0	0	1	0
XL	0	0	0	1
M	0	1	0	0

The column that you specify for `ONE_HOT_ENCODING` can have a maximum of ten (10) distinct values.

Parameters

- `sourceColumn` – The name of an existing column. The column can have a maximum of 10 distinct values.

Example

```
{
  "RecipeAction": {
    "Operation": "ONE_HOT_ENCODING",
```

```
    "Parameters": {
      "sourceColumn": "shirt_size"
    }
  }
}
```

SCALE

Scales or normalizes the range of data in a numeric column.

Parameters

- **sourceColumn** – The name of an existing column.
- **strategy** – The operation to be applied to the column values:
 - **MIN_MAX** – Rescales the values into a range of [0,1]
 - **SCALE_BETWEEN** – Rescales the values into a range of 2 specified values.
 - **MEAN_NORMALIZATION** – Rescales the data to have a mean (μ) of 0 and standard deviation (σ) of 1 within a range of [-1, 1]
 - **Z_SCORE** – Linearly scale data values to have a mean (μ) of 0 and standard deviation (σ) of 1. Best for handling outliers.
- **targetColumn** – The name of a column to contain the results.

Example

```
{
  "Action": {
    "Operation": "NORMALIZATION",
    "Parameters": {
      "sourceColumn": "all_votes",
      "strategy": "MIN_MAX",
      "targetColumn": "all_votes_normalized"
    }
  }
}
```

SKEWNESS

Applies transformations on your data values to change the distribution shape and its skew.

Parameters

- **sourceColumn** – The name of an existing column.

targetColumn – The name of the new column to be created.

skewFunction
 - **ROOT** – extract value-root. The root can be provided in the **value** parameter.
 - **LOG** – log base value. The log base can be provided in the **value** parameter.
 - **SQUARE** – square function
- **value** – Argument of the **skewFunction**.

Example

```
{
  "RecipeAction": {
    "Operation": "SKEWNESS",
    "Parameters": {
      "sourceColumn": "level",
      "targetColumn": "bin",
      "skewFunction": "LOG",
      "value": "2.718281828"
    }
  }
}
```

TOKENIZATION

Splits text into smaller units, or tokens, such as individual words or terms.

Parameters

- **sourceColumn** – The name of an existing column.
- **delimiter** — A custom delimiter that appears between tokenized words. (The default behavior is to separate each token by a space.)
- **expandContractions** — If **ENABLED**, expands contracted words. For example: "don't" becomes "do not".

- **stemmingMode** — Splits text into smaller units or tokens, such as individual lowercase words or terms. Two stemming modes are available: PORTER | LANCASTER.
- **stopWordRemovalMode** — Removes common words like a, an, the, and more.
- **customStopWords** — For StopWordRemovalMode, allows you to specify a custom list of stop words.
- **targetColumn** — The name of a column to contain the results.

Example

```
{
  "Action": {
    "Operation": "TOKENIZATION",
    "Parameters": {
      "customStopWords": "[]",
      "delimiter": "- ",
      "expandContractions": "ENABLED",
      "sourceColumn": "dimensions",
      "stemmingMode": "PORTER",
      "stopWordRemovalMode": "DEFAULT",
      "targetColumn": "dimensions_tokenized"
    }
  }
}
```

Mathematical functions

Following, find reference topics for mathematical functions that work with recipe actions.

Topics

- [ABSOLUTE](#)
- [ADD](#)
- [CEILING](#)
- [DEGREES](#)
- [DIVIDE](#)
- [EXPONENT](#)

- [FLOOR](#)
- [IS_EVEN](#)
- [IS_ODD](#)
- [LN](#)
- [LOG](#)
- [MOD](#)
- [MULTIPLY](#)
- [NEGATE](#)
- [PI](#)
- [POWER](#)
- [RADIANS](#)
- [RANDOM](#)
- [RANDOM_BETWEEN](#)
- [ROUND](#)
- [SIGN](#)
- [SQUARE_ROOT](#)
- [SUBTRACT](#)

ABSOLUTE

Returns the absolute value of the input number in a new column. *Absolute value* is how far the number is from zero, regardless of whether it is positive or negative

Parameters

- `sourceColumn` – The name of an existing column.
- `targetColumn` – The name of the new column to be created.

Example

```
{
  "RecipeAction": {
    "Operation": "ABSOLUTE",
    "Parameters": {
```

```
        "sourceColumn": "freezingTemps",
        "targetColumn": "absValueOfFreezingTemps"
    }
}
```

ADD

Sums the input column values in a new column, using (sourceColumn1 + sourceColumn2) or (sourceColumn1 + value1).

Parameters

- sourceColumn1 – The name of an existing column.
- value1 – A numeric value.
- sourceColumn2 – The name of an existing column.
- targetColumn – The name of the new column to be created.

Example

```
{
  "RecipeAction": {
    "Operation": "ADD",
    "Parameters": {
      "sourceColumn1": "weight_kg",
      "sourceColumn2": "height_cm",
      "targetColumn": "weight_plus_height"
    }
  }
}
```

CEILING

Returns the smallest integer number greater than or equal to the input decimal numbers in a new column.

Parameters

- sourceColumn – The name of an existing column.

- `value1` – A numeric value.
- `targetColumn` – The name of the new column to be created.

Example

```
{
  "RecipeAction": {
    "Operation": "CEILING",
    "Parameters": {
      "sourceColumn": "weight_kg",
      "targetColumn": "weight_kg_CEILING"
    }
  }
}
```

DEGREES

Converts radians for an angle to degrees and returns the result in a new column.

Parameters

- `sourceColumn` – The name of an existing column.
- `targetColumn` – The name of the new column to be created.

Example

```
{
  "RecipeAction": {
    "Operation": "DEGREES",
    "Parameters": {
      "sourceColumn": "height_cm",
      "targetColumn": "height_cm_DEGREES"
    }
  }
}
```

DIVIDE

Divides one input number by another and returns the result in a new column.

Parameters

- `sourceColumn1` – The name of an existing column.
- `value1` – A numeric value.
- `sourceColumn2` – The name of an existing column.
- `value2` – A numeric value.
- `targetColumn` – The name of the new column to be created.

Example

```
{
  "RecipeAction": {
    "Operation": "DIVIDE",
    "Parameters": {
      "sourceColumn1": "height_cm",
      "targetColumn": "divide_by_2",
      "value2": "2"
    }
  }
}
```

EXPONENT

Returns Euler's number raised to the *n*th degree in a new column.

Parameters

- `sourceColumn` – The name of an existing column.
- `targetColumn` – The name of the new column to be created.

Example

```
{
  "RecipeAction": {
    "Operation": "EXPONENT",
    "Parameters": {
      "sourceColumn": "age",

```

```
        "targetColumn": "age_EXPONENT"
    }
}
```

FLOOR

Returns the largest integral number greater than or equal to the input number in a new column.

Parameters

- `sourceColumn1` – The name of an existing column.
- `value` – A numeric value.
- `targetColumn` – The name of the new column to be created.

Example

```
{
  "RecipeAction": {
    "Operation": "FLOOR",
    "Parameters": {
      "targetColumn": "FLOOR Column 1",
      "value": "42"
    }
  }
}
```

IS_EVEN

Returns a Boolean value in a new column that indicates whether the source column or value is even. If the source column or value is a decimal, the result is false.

Parameters

- `sourceColumn` – The name of an existing column.
- `targetColumn` – The name of the new column to be created.
- `trueString` – A string that indicates whether the value is even.
- `falseString` – A string that indicates whether the value is *not* even.

Example

```
{
  "RecipeAction": {
    "Operation": "IS_EVEN",
    "Parameters": {
      "falseString": "Value is odd",
      "sourceColumn": "height_cm",
      "targetColumn": "height_cm_IS_EVEN",
      "trueString": "Value is even"
    }
  }
}
```

IS_ODD

Returns a Boolean value in a new column that indicates whether the source column or value is odd. If the source column or value is a decimal, the result is false.

Parameters

- `sourceColumn` – The name of an existing column.
- `targetColumn` – The name of the new column to be created.
- `trueString` – A string that indicates whether the value is odd.
- `falseString` – A string that indicates whether the value is *not* odd.

Example

```
{
  "RecipeAction": {
    "Operation": "IS_ODD",
    "Parameters": {
      "falseString": "Value is even",
      "sourceColumn": "weight_kg",
      "targetColumn": "weight_kg_IS_ODD",
      "trueString": "Value is odd"
    }
  }
}
```

LN

Returns the natural logarithm (Euler's number) of a value in a new column.

Parameters

- `sourceColumn` – The name of an existing column.
- `targetColumn` – The name of the new column to be created.

Example

```
{
  "RecipeAction": {
    "Operation": "LN",
    "Parameters": {
      "sourceColumn": "weight_kg",
      "targetColumn": "weight_kg_LN"
    }
  }
}
```

LOG

Returns the logarithm of a value in a new column.

Parameters

- `sourceColumn` – The name of an existing column.
- `targetColumn` – The name of the new column to be created.
- `base` – The base of the logarithm. The default is 10.

Example

```
{
  "RecipeAction": {
    "Operation": "LOG",
    "Parameters": {
      "base": "10",

```

```
        "sourceColumn": "age",
        "targetColumn": "age_LOG"
    }
}
```

MOD

Returns the percent that one number is of another number in a new column.

Parameters

- `sourceColumn1` – The name of an existing column.
- `sourceColumn2` – The name of an existing column.
- `targetColumn` – The name of the new column to be created.

Example

```
{
  "RecipeAction": {
    "Operation": "MOD",
    "Parameters": {
      "sourceColumn1": "start_date",
      "sourceColumn2": "end_date",
      "targetColumn": "MOD Column 1"
    }
  }
}
```

MULTIPLY

Multiplies two numbers and returns the result in a new column.

Parameters

- `sourceColumn1` – The name of an existing column.
- `value1` – A numeric value.
- `sourceColumn2` – The name of an existing column.
- `value2` – A numeric value.

- **targetColumn** – The name of the new column to be created.

Example

```
{
  "RecipeAction": {
    "Operation": "MULTIPLY",
    "Parameters": {
      "sourceColumn1": "hourly_rate",
      "sourceColumn2": "hours",
      "targetColumn": "total_pay"
    }
  }
}
```

NEGATE

Negates a value and returns the result in a new column.

Parameters

- **sourceColumn** – The name of an existing column.
- **targetColumn** – The name of the new column to be created.

Example

```
{
  "RecipeAction": {
    "Operation": "NEGATE",
    "Parameters": {
      "sourceColumn": "age",
      "targetColumn": "age_NEGATE"
    }
  }
}
```

PI

Returns the value of pi (3.141592653589793) in a new column.

Parameters

- **targetColumn** – The name of the new column to be created.

Example

```
{
  "RecipeAction": {
    "Operation": "PI",
    "Parameters": {
      "targetColumn": "PI Column 1"
    }
  }
}
```

POWER

Returns the value of a number to the power of the exponent in a new column.

Parameters

- **sourceColumn** – The name of an existing column.
- **value** – A number whose value is to be raised.
- **targetColumn** – The name of the new column to be created.
- **exponent** – The power to which the value will be raised.

Note

You can specify either **sourceColumn** or **value**, but not both.

Example

```
{
  "RecipeAction": {
    "Operation": "POWER",
    "Parameters": {
```

```
        "exponent": "3",
        "sourceColumn": "age",
        "targetColumn": "age_cubed"
    }
}
```

RADIANS

Converts degrees to radians (divides by 180/pi) and returns the value in a new column.

Parameters

- `sourceColumn` – The name of an existing column.
- `targetColumn` – The name of the new column to be created.

Example

```
{
  "RecipeAction": {
    "Operation": "RADIANS",
    "Parameters": {
      "sourceColumn": "weight_kg",
      "targetColumn": "weight_kg_RADIANS"
    }
  }
}
```

RANDOM

Returns a random number between 0 and 1 in a new column.

Parameters

- `targetColumn` – The name of the new column to be created.

Example

```
{
```

```
    "RecipeAction": {  
      "Operation": "RANDOM",  
      "Parameters": {  
        "targetColumn": "RANDOM Column 1"  
      }  
    }  
  }  
}
```

RANDOM_BETWEEN

In a new column, returns a random number between a specified lower bound (inclusive) and a specified upper bound (inclusive).

Parameters

- `lowerBound` – The lower bound of the random number range.
- `upperBound` – The upper bound of the random number range.
- `targetColumn` – The name of the new column to be created.

Example

```
{  
  "RecipeAction": {  
    "Operation": "RANDOM_BETWEEN",  
    "Parameters": {  
      "lowerBound": "1",  
      "targetColumn": "RANDOM_BETWEEN Column 1",  
      "upperBound": "100"  
    }  
  }  
}
```

ROUND

Rounds a numerical value to the nearest integer in a new column. It rounds up when the fraction is 0.5 or more.

Parameters

- `sourceColumn` – The name of an existing column.

- **targetColumn** – The name of the new column to be created.

Example

```
{
  "RecipeAction": {
    "Operation": "ROUND",
    "Parameters": {
      "sourceColumn": "rating",
      "targetColumn": "rating_ROUND"
    }
  }
}
```

SIGN

Returns a new column with -1 if the value is less than 0, 0 if the value is 0, and +1 if the value is greater than 0.

Parameters

- **sourceColumn** – The name of an existing column.
- **targetColumn** – The name of the new column to be created.

Example

```
{
  "RecipeAction": {
    "Operation": "SIGN",
    "Parameters": {
      "sourceColumn": "age",
      "targetColumn": "age_SIGN"
    }
  }
}
```

SQUARE_ROOT

Returns the square root of a value in a new column.

Parameters

- `sourceColumn` – The name of an existing column.
- `targetColumn` – The name of the new column to be created.

Example

```
{
  "RecipeAction": {
    "Operation": "SQUARE_ROOT",
    "Parameters": {
      "sourceColumn": "age",
      "targetColumn": "age_SQUARE_ROOT"
    }
  }
}
```

SUBTRACT

Subtracts one number from another and returns the result in a new column.

Parameters

- `sourceColumn1` – The name of an existing column.
- `value1` – A numeric value.
- `sourceColumn2` – The name of an existing column.
- `value2` – A numeric value.
- `targetColumn` – The name of the new column to be created.

Example

```
{
  "RecipeAction": {
    "Operation": "SUBTRACT",
    "Parameters": {
      "sourceColumn1": "weight_kg",
```

```
        "targetColumn": "weight_minus_10_kg",  
        "value2": "10"  
    }  
}
```

Aggregate functions

Following, find reference topics for aggregate functions that work with recipe actions.

Topics

- [ANY](#)
- [AVERAGE](#)
- [COUNT](#)
- [COUNT_DISTINCT](#)
- [KTH_LARGEST](#)
- [KTH_LARGEST_UNIQUE](#)
- [MAX](#)
- [MEDIAN](#)
- [MIN](#)
- [MODE](#)
- [STANDARD_DEVIATION](#)
- [SUM](#)
- [VARIANCE](#)

ANY

Returns any values from the selected source columns in a new column. Empty and null values are ignored.

Parameters

- `sourceColumns` – A JSON-encoded string representing a list of existing columns.
- `targetColumn` – A name for the newly created column.

Example

```
{
  "RecipeAction": {
    "Operation": "ANY",
    "Parameters": {
      "sourceColumns": "[\"age\", \"last_name\"]",
      "targetColumn": "ANY Column 1"
    }
  }
}
```

AVERAGE

Calculates the average of the values in the source columns and returns the result in a new column. Any non-number is ignored.

Parameters

- `sourceColumns` – A JSON-encoded string representing a list of existing columns.
- `targetColumn` – A name for the newly created column.

Example

```
{
  "RecipeAction": {
    "Operation": "AVERAGE",
    "Parameters": {
      "sourceColumns": "[\"age\", \"weight_kg\", \"height_cm\"]",
      "targetColumn": "AVERAGE Column 1"
    }
  }
}
```

COUNT

Returns the number of values from the selected source columns in a new column. Empty and null values are ignored.

Parameters

- `sourceColumns` – A JSON-encoded string representing a list of existing columns.
- `targetColumn` – A name for the newly created column.

Example

```
{
  "RecipeAction": {
    "Operation": "COUNT",
    "Parameters": {
      "sourceColumns": "[\"ANY Column 1\", \"birth_date\", \"last_name\"]",
      "targetColumn": "COUNT Column 1"
    }
  }
}
```

COUNT_DISTINCT

Returns the total number of distinct values from the selected source columns in a new column. Empty and null values are ignored.

Parameters

- `sourceColumns` – A JSON-encoded string representing a list of existing columns.
- `targetColumn` – A name for the newly created column.

Example

```
{
  "RecipeAction": {
    "Operation": "COUNT_DISTINCT",
    "Parameters": {
      "sourceColumns": "[\"long_name\", \"weight_kg\"]",
      "targetColumn": "COUNT_DISTINCT Column 1"
    }
  }
}
```

KTH_LARGEST

Returns the *k*th largest number from the selected source columns in a new column.

Parameters

- `sourceColumns` – A JSON-encoded string representing a list of existing columns.
- `targetColumn` – A name for the newly created column.
- `value` – A number representing *k*.

Example

```
{
  "RecipeAction": {
    "Operation": "KTH_LARGEST",
    "Parameters": {
      "sourceColumns": "[\"height_cm\",\"weight_kg\",\"age\"]",
      "targetColumn": "KTH_LARGEST Column 1",
      "value": "2"
    }
  }
}
```

KTH_LARGEST_UNIQUE

Returns the *k*th largest unique number from the selected source columns in a new column.

Parameters

- `sourceColumns` – A JSON-encoded string representing a list of existing columns.
 - `targetColumn` – A name for the newly created column.
- `value` – A number representing *k*.

Example

```
{
  "RecipeAction": {
```

```
    "Operation": "KTH_LARGEST_UNIQUE",
    "Parameters": {
      "sourceColumns": "[\"age\", \"height_cm\", \"weight_kg\"]",
      "targetColumn": "KTH_LARGEST_UNIQUE Column 1",
      "value": "3"
    }
  }
}
```

MAX

Returns the maximum numerical value from the selected source columns in a new column. Any non-number is ignored.

Parameters

- `sourceColumns` – A JSON-encoded string representing a list of existing columns.
- `targetColumn` – A name for the newly created column.

Example

```
{
  "RecipeAction": {
    "Operation": "MAX",
    "Parameters": {
      "sourceColumns": "[\"age\", \"height_cm\", \"weight_kg\"]",
      "targetColumn": "MAX Column 1"
    }
  }
}
```

MEDIAN

Returns the median, the middle number of a sorted group of numbers, from the selected source columns in a new column. Any non-number is ignored.

Parameters

- `sourceColumns` – A JSON-encoded string representing a list of existing columns.
- `targetColumn` – A name for the newly created column.

Example

```
{
  "RecipeAction": {
    "Operation": "MEDIAN",
    "Parameters": {
      "sourceColumns": "[\"age\", \"years_in_service\"]",
      "targetColumn": "MEDIAN Column 1"
    }
  }
}
```

MIN

Returns the minimum value from the selected source columns in a new column. Any non-number is ignored.

Parameters

- `sourceColumns` – A JSON-encoded string representing a list of existing columns.
- `targetColumn` – A name for the newly created column.

Example

```
{
  "RecipeAction": {
    "Operation": "MIN",
    "Parameters": {
      "sourceColumns": "[\"age\", \"height_cm\", \"weight_kg\"]",
      "targetColumn": "MIN Column 1"
    }
  }
}
```

MODE

Returns the mode, the number that appears most often, from the selected source columns in a new column. Any non-number is ignored. For multiple modes, the mode is calculated with the modal function.

Parameters

- `sourceColumns` – A JSON-encoded string representing a list of existing columns.
- `targetColumn` – A name for the newly created column.

Example

```
{
  "RecipeAction": {
    "Operation": "MODE",
    "Parameters": {
      "modeType": "MINIMUM",
      "sourceColumns": "[\"years_in_service\", \"age\"]",
      "targetColumn": "MODE Column 1"
    }
  }
}
```

STANDARD_DEVIATION

Returns the standard deviation from the selected source columns in a new column.

Parameters

- `sourceColumns` – A JSON-encoded string representing a list of existing columns.
- `targetColumn` – A name for the newly created column.

Example

```
{
  "RecipeAction": {
    "Operation": "STANDARD_DEVIATION",
    "Parameters": {
      "sourceColumns": "[\"years_in_service\", \"age\"]",
      "targetColumn": "STANDARD_DEVIATION Column 1"
    }
  }
}
```

SUM

Returns the sum of the values from the selected source columns in a new column. Any non-number is treated as 0.

Parameters

- `sourceColumns` – A JSON-encoded string representing a list of existing columns.
- `targetColumn` – A name for the newly created column.

Example

```
{
  "RecipeAction": {
    "Operation": "SUM",
    "Parameters": {
      "sourceColumns": "[\"age\", \"years_in_service\"]",
      "targetColumn": "SUM Column 1"
    }
  }
}
```

VARIANCE

Returns the variance from the selected source columns in a new column. Variance is defined as $\text{Var}(X) = [\text{Sum}((X - \text{mean}(X))^2)] / \text{Count}(X)$.

Parameters

- `sourceColumns` – A JSON-encoded string representing a list of existing columns.
- `targetColumn` – A name for the newly created column.

Example

```
{
  "RecipeAction": {
    "Operation": "VARIANCE",
    "Parameters": {
```

```
        "sourceColumns": "[\"age\", \"years_in_service\"]",
        "targetColumn": "VARIANCE Column 1"
    }
}
```

Text functions

Following, find reference topics for text functions that work with recipe actions.

Topics

- [CHAR](#)
- [ENDS_WITH](#)
- [EXACT](#)
- [FIND](#)
- [LEFT](#)
- [LEN](#)
- [LOWER](#)
- [MERGE_COLUMNS_AND_VALUES](#)
- [PROPER](#)
- [REMOVE_SYMBOLS](#)
- [REMOVE_WHITESPACE](#)
- [REPEAT_STRING](#)
- [RIGHT](#)
- [RIGHT_FIND](#)
- [STARTS_WITH](#)
- [STRING_GREATER_THAN](#)
- [STRING_GREATER_THAN_EQUAL](#)
- [STRING_LESS_THAN](#)
- [STRING_LESS_THAN_EQUAL](#)
- [SUBSTRING](#)
- [TRIM](#)
- [UNICODE](#)

- [UPPER](#)

CHAR

Returns in a new column the Unicode character for each integer in the source column, or for a custom integer value.

Parameters

- `sourceColumn` – The name of an existing column.
- `value` – An integer that represents a Unicode value.
- `targetColumn` – The name of the new column to be created.

Note

You can specify either `sourceColumn` or `value`, but not both.

Examples

```
{
  "RecipeAction": {
    "Operation": "CHAR",
    "Parameters": {
      "sourceColumn": "age",
      "targetColumn": "age_char"
    }
  }
}
```

```
{
  "RecipeAction": {
    "Operation": "CHAR",
    "Parameters": {
      "value": 42,
      "targetColumn": "asterisk"
    }
  }
}
```

```
}
```

ENDS_WITH

Returns true in a new column if a specified number of rightmost characters, or custom string, matches a pattern.

Parameters

- `sourceColumn` – The name of an existing column.
- `value` – A character string to evaluate.
- `pattern` – A regular expression that must match the end of the string.
- `targetColumn` – The name of the new column to be created.

Note

You can specify either `sourceColumn` or `value`, but not both.

Example

```
{
  "RecipeAction": {
    "Operation": "ENDS_WITH",
    "Parameters": {
      "sourceColumn": "nationality",
      "pattern": "[Ss]",
      "targetColumn": "nationality_ends_with"
    }
  }
}
```

EXACT

Creates a new column populated with one of the following:

- True if one string in a column (or value) exactly matches another string in a different column (or value).

- False if there is no match.

Parameters

- sourceColumn1 – The name of an existing column.
- sourceColumn2 – The name of an existing column.
- value1 – A character string to evaluate.
- value2 – A character string to evaluate.
- targetColumn – The name of the new column to be created.

Note

You can specify only one of the following combinations:

- Both of sourceColumn N .
- One of sourceColumn N and one of value N .
- Both of value N .

Example

```
{
  "RecipeAction": {
    "Operation": "EXACT",
    "Parameters": {
      "sourceColumn1": "nationality",
      "value2": "Argentina",
      "targetColumn": "nationality_exact"
    }
  }
}
```

FIND

Searching left to right, finds strings that match a specified string from the source column or from a custom value, and returns the result in a new column.

Parameters

- `sourceColumn` – The name of an existing column.
- `pattern` – A regular expression to search for.
- `position` – The character position to begin with, from the left end of the string.
- `ignoreCase` – If `true`, ignore differences of case (between uppercase and lowercase) among letters. To enforce strict matching, use `false` instead.
- `targetColumn` – The name of the new column to be created.

Example

```
{
  "RecipeAction": {
    "Operation": "FIND",
    "Parameters": {
      "sourceColumn": "city",
      "pattern": "[AEIOU]",
      "position": "1",
      "ignoreCase": "false",
      "targetColumn": "begins_with_a_vowel"
    }
  }
}
```

LEFT

Given a number of characters, takes the leftmost number of characters in the string from the source column or custom string, and returns the specified number of leftmost characters in a new column.

Parameters

- `sourceColumn` – The name of an existing column.
- `value` – A character string to evaluate.
- `position` – The character position to begin with, from the left end of the string.
- `targetColumn` – The name of the new column to be created.

Note

You can specify either `sourceColumn` or `value`, but not both.

Examples

```
{
  "RecipeAction": {
    "Operation": "LEFT",
    "Parameters": {
      "position": "3",
      "sourceColumn": "city",
      "targetColumn": "city_left"
    }
  }
}
```

```
{
  "RecipeAction": {
    "Operation": "LEFT",
    "Parameters": {
      "position": "5",
      "value": "How now brown cow",
      "targetColumn": "how_now_5_left_chars"
    }
  }
}
```

LEN

Returns in a new column the length of strings from the source column or of custom strings.

Parameters

- `sourceColumn` – The name of an existing column.
- `value` – A character string to evaluate.
- `targetColumn` – The name of the new column to be created.

Note

You can specify either `sourceColumn` or `value`, but not both.

Examples

```
{
  "RecipeAction": {
    "Operation": "LEN",
    "Parameters": {
      "sourceColumn": "last_name",
      "targetColumn": "last_name_len"
    }
  }
}
```

```
{
  "RecipeAction": {
    "Operation": "LEN",
    "Parameters": {
      "value": "Hello",
      "targetColumn": "hello_len"
    }
  }
}
```

LOWER

Converts all alphabetical characters from the strings in the source column or custom strings to lowercase, and returns the result in a new column.

Parameters

- `sourceColumn` – The name of an existing column.
- `value` – A character string to evaluate.
- `targetColumn` – The name of the new column to be created.

Note

You can specify either `sourceColumn` or `value`, but not both.

Examples

```
{
  "RecipeAction": {
    "Operation": "LOWER",
    "Parameters": {
      "sourceColumn": "last_name",
      "targetColumn": "last_name_lower"
    }
  }
}
```

```
{
  "RecipeAction": {
    "Operation": "LOWER",
    "Parameters": {
      "value": "GOODBYE",
      "targetColumn": "goodbye_lower"
    }
  }
}
```

MERGE_COLUMNS_AND_VALUES

Concatenates the strings in the source columns and returns the result in a new column. You can insert a delimiter between the merged values.

Parameters

- `sourceColumns` – The names of two or more existing columns, in JSON-encoded format.
- `delimiter` – Optional. One or more characters to place between each two source column values.
- `targetColumn` – The name of the new column to be created.

Example

```
{
  "RecipeAction": {
    "Operation": "MERGE_COLUMNS_AND_VALUES",
    "Parameters": {
      "sourceColumns": "[\"last_name\", \"birth_date\"]",
      "delimiter": " was born on: ",
      "targetColumn": "merged_column"
    }
  }
}
```

PROPER

Converts all alphabetical characters from the strings in the source column or custom values to proper case, and returns the result in a new column.

In *proper case*, also called capital case, the first letter of each word is capitalized and the rest of the word is transformed to lowercase. An example is: The Quick Brown Fox Jumped Over The Fence

Parameters

- `sourceColumn` – The name of an existing column.
- `value` – A character string to evaluate.
- `targetColumn` – The name of the new column to be created.

Note

You can specify either `sourceColumn` or `value`, but not both.

Examples

```
{
  "RecipeAction": {
    "Operation": "PROPER",
    "Parameters": {
```

```
        "sourceColumn": "first_name",
        "targetColumn": "first_name_proper"
    }
}
```

```
{
  "RecipeAction": {
    "Operation": "PROPER",
    "Parameters": {
      "value": "MR. H. SMITH, ESQ.",
      "targetColumn": "formal_name_proper"
    }
  }
}
```

REMOVE_SYMBOLS

Removes characters that aren't letters, numbers, accented Latin characters, or white space from the strings in the source column or custom strings, and returns the result in a new column.

Parameters

- **sourceColumn** – The name of an existing column.
- **value** – A character string to evaluate.
- **targetColumn** – The name of the new column to be created.

Note

You can specify either **sourceColumn** or **value**, but not both.

Examples

```
{
  "RecipeAction": {
    "Operation": "REMOVE_SYMBOLS",
    "Parameters": {
```

```
        "sourceColumn": "info_url",
        "targetColumn": "info_url_remove_symbols"
    }
}
```

```
{
  "RecipeAction": {
    "Operation": "REMOVE_SYMBOLS",
    "Parameters": {
      "value": "$&#$&HEY!#@@",
      "targetColumn": "without_symbols"
    }
  }
}
```

REMOVE_WHITESPACE

Removes white space from the strings in the source column or custom strings, and returns the result in a new column.

Parameters

- **sourceColumn** – The name of an existing column.
- **value** – A character string to evaluate.
- **targetColumn** – The name of the new column to be created.

Note

You can specify either **sourceColumn** or **value**, but not both.

Examples

```
{
  "RecipeAction": {
    "Operation": "REMOVE_WHITESPACE",
    "Parameters": {
```

```
        "sourceColumn": "job_desc",
        "targetColumn": "job_desc_remove_whitespace"
    }
}
```

```
{
  "RecipeAction": {
    "Operation": "REMOVE_WHITESPACE",
    "Parameters": {
      "value": "This string has spaces in it",
      "targetColumn": "string_without_spaces"
    }
  }
}
```

REPEAT_STRING

Repeats the strings in the source column or custom input value a specified number of times, and returns the result in a new column.

Parameters

- `sourceColumn` – The name of an existing column.
- `value` – A character string to evaluate.
- `count` – The number of times to repeat the string.
- `targetColumn` – The name of the new column to be created.

Note

You can specify either `sourceColumn` or `value`, but not both.

Examples

```
{
  "RecipeAction": {
    "Operation": "REPEAT_STRING",
```

```
    "Parameters": {
      "count": 3,
      "sourceColumn": "last_name",
      "targetColumn": "last_name_repeat_string"
    }
  }
}
```

```
{
  "RecipeAction": {
    "Operation": "REPEAT_STRING",
    "Parameters": {
      "count": 80,
      "value": "*",
      "targetColumn": "80_stars"
    }
  }
}
```

RIGHT

Given a number of characters, takes the rightmost number of characters in the strings from the source column or custom strings, and returns the specified number of rightmost characters in a new column.

Parameters

- **sourceColumn** – The name of an existing column.
- **value** – A character string to evaluate.
- **position** – The character position to begin with, from the right side of the string.
- **targetColumn** – The name of the new column to be created.

Note

You can specify either **sourceColumn** or **value**, but not both.

Examples

```
{
  "RecipeAction": {
    "Operation": "RIGHT",
    "Parameters": {
      "sourceColumn": "nationality",
      "position": "3",
      "targetColumn": "nationality_right"
    }
  }
}
```

```
{
  "RecipeAction": {
    "Operation": "RIGHT",
    "Parameters": {
      "value": "United States of America",
      "position": "7",
      "targetColumn": "usa_right"
    }
  }
}
```

RIGHT_FIND

Searching right to left, finds strings that match a specified string from the source column or from a custom value, and returns the result in a new column.

Parameters

- **sourceColumn** – The name of an existing column.
- **pattern** – A regular expression to search for.
- **position** – The character position to begin with, from the right end of the string.
- **ignoreCase** – If `true`, ignore differences of case (between uppercase and lowercase) among letters. To enforce strict matching, use `false` instead.
- **targetColumn** – The name of the new column to be created.

Example

```
{
  "RecipeAction": {
    "Operation": "RIGHT_FIND",
    "Parameters": {
      "sourceColumn": "nationality",
      "pattern": "s",
      "position": "1",
      "ignoreCase": "true",
      "targetColumn": "ends_with_an_s"
    }
  }
}
```

STARTS_WITH

Returns true in a new column if a specified number of leftmost characters, or custom string, matches a pattern.

Parameters

- **sourceColumn** – The name of an existing column.
- **value** – A character string to evaluate.
- **pattern** – A regular expression that must match the start of the string.
- **targetColumn** – The name of the new column to be created.

Note

You can specify either **sourceColumn** or **value**, but not both.

Example

```
{
  "RecipeAction": {
    "Operation": "STARTS_WITH",
    "Parameters": {
      "sourceColumn": "nationality",
      "pattern": "[AEIOU]",
      "targetColumn": "nationality_starts_with"
    }
  }
}
```

```
    }  
  }  
}
```

STRING_GREATER_THAN

Creates a new column populated with one of the following:

- True if one string in a column (or value) is greater than another string in a different column (or value).
- False if there is no match.

Parameters

- `sourceColumn1` – The name of an existing column.
- `sourceColumn2` – The name of an existing column.
- `value1` – A character string to evaluate.
- `value2` – A character string to evaluate.
- `targetColumn` – The name of the new column to be created.

Note

You can specify only one of the following combinations:

- Both of `sourceColumnN`.
- One of `sourceColumnN` and one of `valueN`.
- Both of `valueN`.

Example

```
{  
  "RecipeAction": {  
    "Operation": "STRING_GREATER_THAN",  
    "Parameters": {  
      "sourceColumn1": "first_name",
```

```
        "sourceColumn2": "last_name",
        "targetColumn": "string_greater_than"
    }
}
```

STRING_GREATER_THAN_EQUAL

Creates a new column populated with one of the following:

- True if one string in a column (or value) is greater than or equal to another string in a different column (or value).
- False if there is no match.

Parameters

- `sourceColumn1` – The name of an existing column.
- `sourceColumn2` – The name of an existing column.
- `value1` – A character string to evaluate.
- `value2` – A character string to evaluate.
- `targetColumn` – The name of the new column to be created.

Note

You can specify only one of the following combinations:

- Both of `sourceColumnN`.
- One of `sourceColumnN` and one of `valueN`.
- Both of `valueN`.

Example

```
{
  "RecipeAction": {
    "Operation": "STRING_GREATER_THAN_EQUAL",
```

```
    "Parameters": {
      "sourceColumn1": "nationality",
      "targetColumn": "string_greater_than_equal",
      "value2": "s"
    }
  }
}
```

STRING_LESS_THAN

Creates a new column populated with one of the following:

- True if one string in a column (or value) is less than another string in a different column (or value).
- False if there is no match.

Parameters

- sourceColumn1 – The name of an existing column.
- sourceColumn2 – The name of an existing column.
- value1 – A character string to evaluate.
- value2 – A character string to evaluate.
- targetColumn – The name of the new column to be created.

Note

You can specify only one of the following combinations:

- Both of sourceColumn N .
- One of sourceColumn N and one of value N .
- Both of value N .

Example

```
{
```

```
"RecipeAction": {  
  "Operation": "STRING_LESS_THAN",  
  "Parameters": {  
    "sourceColumn1": "first_name",  
    "sourceColumn2": "last_name",  
    "targetColumn": "string_less_than"  
  }  
}
```

STRING_LESS_THAN_EQUAL

Creates a new column populated with one of the following:

- True if one string in a column (or value) is less than or equal to another string in a different column (or value).
- False if there is no match.

Parameters

- `sourceColumn1` – The name of an existing column.
- `sourceColumn2` – The name of an existing column.
- `value1` – A character string to evaluate.
- `value2` – A character string to evaluate.
- `targetColumn` – The name of the new column to be created.

Note

You can specify only one of the following combinations:

- Both of `sourceColumnN`.
- One of `sourceColumnN` and one of `valueN`.
- Both of `valueN`.

Example

```
{
  "RecipeAction": {
    "Operation": "STRING_LESS_THAN_EQUAL",
    "Parameters": {
      "sourceColumn1": "first_name",
      "targetColumn": "string_less_than_equal",
      "value2": "s"
    }
  }
}
```

SUBSTRING

Returns in a new column some or all of the specified strings in the source column, based on the user-defined starting and ending index values.

Parameters

- **sourceColumn** – The name of an existing column.
- **startPosition** – The character position to begin with, from the left end of the string.
- **endPosition** – The character position to end with, from the left end of the string.
- **targetColumn** – The name of the new column to be created.

Note

You can specify either **sourceColumn** or **value**, but not both.

Example

```
{
  "RecipeAction": {
    "Operation": "SUBSTRING",
    "Parameters": {
      "sourceColumn": "last_name",
      "startPosition": "5",
      "endPosition": "8",
      "targetColumn": "chars_5_through_8"
    }
  }
}
```

```
    }  
  }  
}
```

TRIM

Removes leading and trailing white space from the strings in the source column or custom strings, and returns the result in a new column. Spaces between words aren't removed.

Parameters

- **sourceColumn** – The name of an existing column.
- **value** – A character string to evaluate.
- **targetColumn** – The name of the new column to be created.

Note

You can specify either **sourceColumn** or **value**, but not both.

Examples

```
{  
  "RecipeAction": {  
    "Operation": "TRIM",  
    "Parameters": {  
      "sourceColumn": "nationality",  
      "targetColumn": "nationality_trim"  
    }  
  }  
}
```

```
{  
  "RecipeAction": {  
    "Operation": "TRIM",  
    "Parameters": {  
      "value": "  This string should be trimmed  ",  
      "targetColumn": "string_trimmed"  
    }  
  }  
}
```

```
    }  
  }  
}
```

UNICODE

Returns in a new column the Unicode index value for the first character of the strings in the source column or for custom strings.

Parameters

- **sourceColumn** – The name of an existing column.
- **value** – A character string to evaluate.
- **targetColumn** – The name of the new column to be created.

Note

You can specify either **sourceColumn** or **value**, but not both.

Examples

```
{  
  "RecipeAction": {  
    "Operation": "UNICODE",  
    "Parameters": {  
      "sourceColumn": "first_name",  
      "targetColumn": "first_name_unicode"  
    }  
  }  
}
```

```
{  
  "RecipeAction": {  
    "Operation": "UNICODE",  
    "Parameters": {  
      "value": "?",  
      "targetColumn": "sixty_three"  
    }  
  }  
}
```

```
    }  
  }  
}
```

UPPER

Converts all alphabetical characters from the strings in the source column or custom strings to uppercase, and returns the result in a new column.

Parameters

- **sourceColumn** – The name of an existing column.
- **value** – A character string to evaluate.
- **targetColumn** – The name of the new column to be created.

Note

You can specify either **sourceColumn** or **value**, but not both.

Examples

```
{  
  "RecipeAction": {  
    "Operation": "UPPER",  
    "Parameters": {  
      "sourceColumn": "last_name",  
      "targetColumn": "last_name_upper"  
    }  
  }  
}
```

```
{  
  "RecipeAction": {  
    "Operation": "UPPER",  
    "Parameters": {  
      "value": "a string of lowercase letters",  
      "targetColumn": "string_upper"  
    }  
  }  
}
```

```
}  
  }  
}
```

Date and time functions

Following, find reference topics for date and time functions that work with recipe actions.

Topics

- [CONVERT_TIMEZONE](#)
- [DATE](#)
- [DATE_ADD](#)
- [DATE_DIFF](#)
- [DATE_FORMAT](#)
- [DATE_TIME](#)
- [DAY](#)
- [HOUR](#)
- [MILLISECOND](#)
- [MINUTE](#)
- [MONTH](#)
- [MONTH_NAME](#)
- [NOW](#)
- [QUARTER](#)
- [SECOND](#)
- [TIME](#)
- [TODAY](#)
- [UNIX_TIME](#)
- [UNIX_TIME_FORMAT](#)
- [WEEK_DAY](#)
- [WEEK_NUMBER](#)
- [YEAR](#)

CONVERT_TIMEZONE

Converts a time value from the source column into a new column based on a specified timezone.

Parameters

- **sourceColumn** – The name of an existing column. The source column can be of type string, date, or timestamp.
- **fromTimeZone** – Source value timezone. If nothing is specified, the default timezone is UTC.
- **toTimeZone** – Timezone to be converted to. If nothing is specified, the default timezone is UTC.
- **targetColumn** – A name for the newly-created column.
- **dateTimeFormat** – Optional. A format string for the date. If the format isn't specified, the default format is used: yyyy-mm-dd HH:MM:SS.

Example

```
{
  "RecipeAction": {
    "Operation": "CONVERT_TIMEZONE",
    "Parameters": {
      "sourceColumn": "DATETIME Column 1",
      "fromTimeZone": "UTC+08:00",
      "toTimeZone": "UTC+08:00",
      "targetColumn": "DATETIME Column CONVERT_TIMEZONE",
      "dateTimeFormat": "yyyy-mm-dd HH:MM:SS"
    }
  }
}
```

DATE

Creates a new column containing the date value, from the source columns or from values provided.

Parameters

- **dateTimeFormat** – Optional. A format string for the date, as it is to appear in the new column. If this string isn't specified, the default format is yyyy-mm-dd HH:MM:SS.

- **dateTimeParameters** – A JSON-encoded string representing the components of the date and time:
 - year
 - value
 - month
 - day
 - hour
 - second

Each component must specify one of the following:

- **sourceColumn** – The name of an existing column.
- **value** – A character string to evaluate.
- **targetColumn** – A name for the newly created column.

Example

```
{
  "RecipeAction": {
    "Operation": "DATE",
    "Parameters": {
      "dateTimeFormat": "mm/dd/yy",
      "dateTimeParameters": "{\"year\":{\"value\":\"2019\"},\"month\":{\"value\":\"12\"},\"day\":{\"value\":\"31\"},\"hour\":{\"value\":\"\"},\"minute\":{\"value\":\"\"},\"second\":{\"value\":\"\"}}",
      "targetColumn": "DATE Column 1"
    }
  }
}
```

DATE_ADD

Adds a year, month, or day to the date from a source column or value, and creates a new column containing the results.

Parameters

- **sourceColumn** – The name of an existing column.
- **value** – A character string to evaluate.

- **units** – A unit of measure for adjusting the date. Valid values are MONTHS, YEARS, MILLISECONDS, QUARTERS, HOURS, MICROSECONDS, WEEKS, SECONDS, DAYS, and MINUTES.
- **dateAddValue** – The number of units to be added to the date.
- **dateTimeFormat** – Optional. A format string for the date, as it is to appear in the new column. If not specified, the default format is yyyy-mm-dd HH:MM:SS.
- **targetColumn** – A name for the newly created column.

Note

You can specify either `sourceColumn` or `value`, but not both.

Example

```
{
  "RecipeAction": {
    "Operation": "DATE_ADD",
    "Parameters": {
      "sourceColumn": "DATE Column 1",
      "units": "DAYS",
      "dateAddValue": "14",
      "dateTimeFormat": "mm/dd/yyyy",
      "targetColumn": "DATE Column 1_DATEADD"
    }
  }
}
```

DATE_DIFF

Creates a new column containing the difference between two dates.

Parameters

- **sourceColumn1** – The name of an existing column.
- **sourceColumn2** – The name of an existing column.
- **value1** – A character string to evaluate.
- **value2** – A character string to evaluate.

- **units** – A unit of measure for describe the difference between the dates. Valid values are MONTHS, YEARS, MILLISECONDS, QUARTERS, HOURS, MICROSECONDS, WEEKS, SECONDS, DAYS, and MINUTES.
- **targetColumn** – A name for the newly created column.

Note

You can only specify one of the following combinations:

- Both of `sourceColumn1` and `sourceColumn2`.
- One of `sourceColumn1` or `sourceColumn2` and one of `value1` or `value2`.
- Both of `value1` and `value2`.

Example

```
{
  "RecipeAction": {
    "Operation": "DATE_DIFF",
    "Parameters": {
      "value1": "2020-01-01",
      "value2": "2020-10-06",
      "units": "DAYS",
      "targetColumn": "DATEDIFF Column 1"
    }
  }
}
```

DATE_FORMAT

Creates a new column containing a date, in a specific format, from a string that represents a date.

Parameters

- **sourceColumn** – The name of an existing column.
- **value** – A string to evaluate.
- **dateTimeFormat** – Optional. A format string for the date, as it is to appear in the new column. If not specified, the default format is `yyyy-mm-dd HH:MM:SS`.

- **targetColumn** – A name for the newly created column.

Note

You can specify either **sourceColumn** or **value**, but not both.

Examples

```
{
  "RecipeAction": {
    "Operation": "DATE_FORMAT",
    "Parameters": {
      "sourceColumn": "DATE Column 1",
      "dateTimeFormat": "month*dd*yyyy",
      "targetColumn": "DATE Column 1_DATEFORMAT"
    }
  }
}
```

```
{
  "RecipeAction": {
    "Operation": "DATE_FORMAT",
    "Parameters": {
      "value": "22:10:47",
      "dateTimeFormat": "HH:MM:SS",
      "targetColumn": "formatted_date_value"
    }
  }
}
```

DATE_TIME

Creates a new column containing the date and time value, from the source columns or from values provided.

Parameters

- **dateTimeFormat** – Optional. A format string for the date, as it is to appear in the new column. If this string isn't specified, the default format is `yyyy-mm-dd HH:MM:SS`.

- **dateTimeParameters** – A JSON-encoded string representing the components of the date and time:
 - **year**
 - **value**
 - **month**
 - **day**
 - **hour**
 - **second**

Each component must specify one of the following:

- **sourceColumn** – The name of an existing column.
- **value** – A character string to evaluate.

Example

```
{
  "RecipeAction": {
    "Operation": "DATE_TIME",
    "Parameters": {
      "dateTimeFormat": "yyyy-mm-dd HH:MM:SS",
      "dateTimeParameters": "{\"year\":{\"value\":\"2010\"},\"month\":{\"value\": \"5\"},\"day\":{\"value\":\"21\"},\"hour\":{\"value\":\"13\"},\"minute\":{\"value\": \"34\"},\"second\":{\"value\":\"25\"}}",
      "targetColumn": "DATETIME Column 1"
    }
  }
}
```

DAY

Creates a new column containing the day of the month, from a string that represents a date.

Parameters

- **sourceColumn** – The name of an existing column.
- **value** – A character string to evaluate.
- **targetColumn** – A name for the newly created column.

Note

You can specify either `sourceColumn` or `value`, but not both.

Example

```
{
  "RecipeAction": {
    "Operation": "DAY",
    "Parameters": {
      "sourceColumn": "DATETIME Column 1",
      "targetColumn": "DATETIME Column 1_DAY"
    }
  }
}
```

HOUR

Creates a new column containing the hour value, from a string that represents a date.

Parameters

- `sourceColumn` – The name of an existing column.
- `value` – A character string to evaluate.
- `targetColumn` – A name for the newly created column.

Note

You can specify either `sourceColumn` or `value`, but not both.

Example

```
{
  "RecipeAction": {
    "Operation": "HOUR",
```

```
    "Parameters": {
      "sourceColumn": "DATETIME Column 1",
      "targetColumn": "DATETIME Column 1_HOUR"
    }
  }
}
```

MILLISECOND

Creates a new column containing the millisecond value from a source column or input value.

Parameters

- **sourceColumn** – The name of an existing column. The source column can be of type string, date, or timestamp.
- **value** – A character string to evaluate.
- **targetColumn** – A name for the newly-created column.

Note

You can specify either **sourceColumn** or **value**, but not both.

Example

```
{
  "RecipeAction": {
    "Operation": "MILLISECOND",
    "Parameters": {
      "sourceColumn": "DATETIME Column 1",
      "targetColumn": "DATETIME Column 1_MILLISECOND"
    }
  }
}
```

MINUTE

Creates a new column containing the minute value, from a string that represents a date.

Parameters

- `sourceColumn` – The name of an existing column.
- `value` – A character string to evaluate.
- `targetColumn` – A name for the newly created column.

Note

You can specify either `sourceColumn` or `value`, but not both.

Example

```
{
  "RecipeAction": {
    "Operation": "MINUTE",
    "Parameters": {
      "sourceColumn": "DATETIME Column 1",
      "targetColumn": "DATETIME Column 1_MINUTE"
    }
  }
}
```

MONTH

Creates a new column containing the number of the month, from a string that represents a date.

Parameters

- `sourceColumn` – The name of an existing column.
- `value` – A character string to evaluate.
- `targetColumn` – A name for the newly created column.

Note

You can specify either `sourceColumn` or `value`, but not both.

Example

```
{
  "RecipeAction": {
    "Operation": "MONTH",
    "Parameters": {
      "value": "2018-05-27",
      "targetColumn": "MONTH Column 1"
    }
  }
}
```

MONTH_NAME

Creates a new column containing the name of the month, from a string that represents a date.

Parameters

- **sourceColumn** – The name of an existing column.
- **value** – A character string to evaluate.
- **targetColumn** – A name for the newly created column.

Note

You can specify either **sourceColumn** or **value**, but not both.

Example

```
{
  "RecipeAction": {
    "Operation": "MONTH_NAME",
    "Parameters": {
      "value": "2018-05-27",
      "targetColumn": "MONTHNAME Column 1"
    }
  }
}
```

```
}
```

NOW

Creates a new column containing the current date and time in the format yyyy-mm-dd HH:MM:SS.

Parameters

- **timeZone** – The name of a time zone. If no time zone is specified, then the default is Universal Coordinated Time (UTC).
- **targetColumn** – A name for the newly created column.

Example

```
{
  "RecipeAction": {
    "Operation": "NOW",
    "Parameters": {
      "timeZone": "US/Pacific",
      "targetColumn": "NOW Column 1"
    }
  }
}
```

QUARTER

Creates a new column containing the date-based quarter from a string that represents a date.

Note

Quarters are designated in the new column as 1, 2, 3, or 4.

- 1 is January, February, and March.
- 2 is April, May, and June.
- 3 is July, August, and September.
- 4 is October, November, and December.

Parameters

- **sourceColumn** – The name of an existing column. The source column can be of type string, date, or timestamp.
- **value** – A character string to evaluate.
- **targetColumn** – A name for the newly-created column.

Note

You can specify either **sourceColumn** or **value**, but not both.

Example

```
{
  "RecipeAction": {
    "Operation": "QUARTER",
    "Parameters": {
      "sourceColumn": "DATETIME Column 1",
      "targetColumn": "DATETIME Column 1_QUARTER"
    }
  }
}
```

SECOND

Creates a new column containing the second value, from a string that represents a date.

Parameters

- **sourceColumn** – The name of an existing column.
- **value** – A character string to evaluate.
- **targetColumn** – A name for the newly created column.

Note

You can specify either **sourceColumn** or **value**, but not both.

Example

```
{
  "RecipeAction": {
    "Operation": "SECOND",
    "Parameters": {
      "sourceColumn": "DATETIME Column 1",
      "targetColumn": "DATETIME Column 1_SECOND"
    }
  }
}
```

TIME

Creates a new column containing the time value, from the source columns or values provided.

Parameters

- **dateTimeFormat** – Optional. A format string for the date, as it is to appear in the new column. If this string isn't specified, the default format is yyyy-mm-dd HH:MM:SS.
- **dateTimeParameters** – A JSON-encoded string representing the components of the date and time:
 - year
 - value
 - month
 - day
 - hour
 - second

Each component must specify one of the following:

- **sourceColumn** – The name of an existing column.
- **value** – A character string to evaluate.
- **targetColumn** – A name for the newly created column.

Example

```
{
  "RecipeAction": {
    "Operation": "TIME",
    "Parameters": {
      "dateTimeFormat": "HH:MM:SS",
      "dateTimeParameters": "{\\"year\\":{\\},\\"month\\":{\\},\\"day\\":{\\},\\"hour\\":{\\},\\"sourceColumn\\":\\"rand_hour\\"},\\"minute\\":{\\},\\"second\\":{\\},\\"sourceColumn\\":\\"rand_minute\\"},\\"second\\":{\\},\\"sourceColumn\\":\\"rand_second\\"}}",
      "targetColumn": "TIME Column 1"
    }
  }
}
```

TODAY

Creates a new column containing the current date in the format yyyy-mm-dd.

Parameters

- **timeZone** – The name of a time zone. If no time zone is specified, then the default is Universal Coordinated Time (UTC).
- **targetColumn** – A name for the newly created column.

Example

```
{
  "RecipeAction": {
    "Operation": "TODAY",
    "Parameters": {
      "timeZone": "US/Pacific",
      "targetColumn": "TODAY Column 1"
    }
  }
}
```

UNIX_TIME

Creates a new column containing a number representing epoch time (Unix time)—the number of seconds since January 1, 1970—based on a source column or input value. If time zone can be

inferred, the output is in that time zone. Otherwise, the output is in Universal Coordinated Time (UTC).

Parameters

- `sourceColumn` – The name of an existing column.
- `value` – A character string to evaluate.
- `targetColumn` – A name for the newly created column.

Note

You can specify either `sourceColumn` or `value`, but not both.

Example

```
{
  "RecipeAction": {
    "Operation": "UNIX_TIME",
    "Parameters": {
      "sourceColumn": "TIME Column 1",
      "targetColumn": "TIME Column 1_UNIXTIME"
    }
  }
}
```

UNIX_TIME_FORMAT

Converts Unix time for a source column or input value to a specified numerical date format, and returns the result in a new column.

Parameters

- `sourceColumn` – The name of an existing column.
- `value` – An integer that represents a Unix epoch timestamp.
- `dateTimeFormat` – Optional. A format string for the date, as it is to appear in the new column. If not specified, the default format is `yyyy-mm-dd HH:MM:SS`.
- `targetColumn` – A name for the newly created column.

Note

You can specify either `sourceColumn` or `value`, but not both.

Example

```
{
  "RecipeAction": {
    "Operation": "UNIX_TIME_FORMAT",
    "Parameters": {
      "value": "1601936554",
      "dateTimeFormat": "yyyy-mm-dd HH:MM:SS",
      "targetColumn": "UNIXTIMEFORMAT Column 1"
    }
  }
}
```

WEEK_DAY

Creates a new column containing the day of the week, from a string that represents a date.

Parameters

- `sourceColumn` – The name of an existing column.
- `value` – A character string to evaluate.
- `targetColumn` – A name for the newly created column.

Note

You can specify either `sourceColumn` or `value`, but not both.

Example

```
{
  "RecipeAction": {
```

```
    "Operation": "WEEK_DAY",
    "Parameters": {
      "sourceColumn": "DATETIME Column 1",
      "targetColumn": "DATETIME Column 1_WEEKDAY"
    }
  }
}
```

WEEK_NUMBER

Creates a new column containing the number of the week (from 1 to 52), from a string that represents a date.

Parameters

- `sourceColumn` – The name of an existing column.
- `value` – A character string to evaluate.
- `targetColumn` – A name for the newly created column.

Note

You can specify either `sourceColumn` or `value`, but not both.

Example

```
{
  "RecipeAction": {
    "Operation": "WEEK_NUMBER",
    "Parameters": {
      "sourceColumn": "DATETIME Column 1",
      "targetColumn": "DATETIME Column 1_WEEK_NUMBER"
    }
  }
}
```

YEAR

Creates a new column containing the year, from a string that represents a date.

Parameters

- `sourceColumn` – The name of an existing column.
- `value` – A character string to evaluate.
- `targetColumn` – A name for the newly created column.

Note

You can specify either `sourceColumn` or `value`, but not both.

Example

```
{
  "RecipeAction": {
    "Operation": "YEAR",
    "Parameters": {
      "value": "2019-06-12",
      "targetColumn": "YEAR Column 1"
    }
  }
}
```

Window functions

Following, find reference topics for window functions that work with recipe actions.

Topics

- [FILL](#)
- [NEXT](#)
- [PREV](#)
- [ROLLING_AVERAGE](#)
- [ROLLING_COUNT_A](#)
- [ROLLING_KTH_LARGEST](#)
- [ROLLING_KTH_LARGEST_UNIQUE](#)

- [ROLLING_MAX](#)
- [ROLLING_MIN](#)
- [ROLLING_MODE](#)
- [ROLLING_STANDARD_DEVIATION](#)
- [ROLLING_SUM](#)
- [ROLLING_VARIANCE](#)
- [ROW_NUMBER](#)
- [SESSION](#)

FILL

Returns a new column based on a specified source column. For any missing or null values in the source column, FILL chooses the most recent nonblank value from a window of rows before and after the source value in question. The chosen value is then placed in the new column.

Parameters

- `sourceColumn` – The name of an existing column.
- `numRowsBefore` – A number of rows before the current source row, representing the start of the window.
- `numRowsAfter` – A number of rows after the current source row, representing the end of the window.
- `targetColumn` – A name for the newly created column.

Example

```
{
  "Action": {
    "Operation": "FILL",
    "Parameters": {
      "numRowsAfter": "10",
      "numRowsBefore": "10",
      "sourceColumn": "last_name",
      "targetColumn": "last_name_FILL"
    }
  }
}
```

```
}
```

NEXT

Returns a new column, where each value represents a value that is n rows later in the source column.

Parameters

- `sourceColumn` – The name of an existing column.
- `numRows` – A value that represents n rows earlier in the source column. For example, if `numRows` is 3, then NEXT uses the third-next `sourceColumn` value as the new `targetColumn` value.
- `targetColumn` – A name for the newly created column.

Example

```
{
  "Action": {
    "Operation": "NEXT",
    "Parameters": {
      "numRows": "1",
      "sourceColumn": "age",
      "targetColumn": "age_NEXT"
    }
  }
}
```

PREV

Returns a new column, where each value represents a value that is n rows earlier in the source column.

Parameters

- `sourceColumn` – The name of an existing column.
- `numRows` – A value that represents n rows earlier in the source column. For example, if `numRows` is 3, then PREV uses the third-previous `sourceColumn` value as the new `targetColumn` value.
- `targetColumn` – A name for the newly created column.

Example

```
{
  "Action": {
    "Operation": "PREV",
    "Parameters": {
      "numRows": "1",
      "sourceColumn": "age",
      "targetColumn": "age_PREV"
    }
  }
}
```

ROLLING_AVERAGE

Returns in a new column the rolling average of values from a specified number of rows before to a specified number of rows after the current row in the specified column.

Parameters

- **sourceColumn** – The name of an existing column.
- **numRowsBefore** – A number of rows before the current source row, representing the start of the window.
- **numRowsAfter** – A number of rows after the current source row, representing the end of the window.
- **targetColumn** – A name for the newly created column.

Example

```
{
  "Action": {
    "Operation": "ROLLING_AVERAGE",
    "Parameters": {
      "numRowsAfter": "10",
      "numRowsBefore": "10",
      "sourceColumn": "weight_kg",
      "targetColumn": "weight_kg_ROLLING_AVERAGE"
    }
  }
}
```

```
}  
}
```

ROLLING_COUNT_A

Returns in a new column the rolling count of non-null values from a specified number of rows before to a specified number of rows after the current row in the specified column.

Parameters

- `sourceColumn` – The name of an existing column.
- `numRowsBefore` – A number of rows before the current source row, representing the start of the window.
- `numRowsAfter` – A number of rows after the current source row, representing the end of the window.
- `targetColumn` – A name for the newly created column.

Example

```
{  
  "Action": {  
    "Operation": "ROLLING_COUNT_A",  
    "Parameters": {  
      "numRowsAfter": "10",  
      "numRowsBefore": "10",  
      "sourceColumn": "weight_kg",  
      "targetColumn": "weight_kg_ROLLING_COUNT_A"  
    }  
  }  
}
```

ROLLING_KTH_LARGEST

Returns in a new column the rolling *k*th largest value from a specified number of rows before to a specified number of rows after the current row in the specified column.

Parameters

- `sourceColumn` – The name of an existing column.

- `numRowsBefore` – A number of rows before the current source row, representing the start of the window.
- `numRowsAfter` – A number of rows after the current source row, representing the end of the window.
- `value` – The value for k .
- `targetColumn` – A name for the newly created column.

Example

```
{
  "Action": {
    "Operation": "ROLLING_KTH_LARGEST",
    "Parameters": {
      "sourceColumn": "weight_kg",
      "numRowsBefore": "5",
      "numRowsAfter": "5",
      "value": "3"
      "targetColumn": "weight_kg_ROLLING_KTH_LARGEST"
    }
  }
}
```

ROLLING_KTH_LARGEST_UNIQUE

Returns in a new column the rolling unique k th largest value from a specified number of rows before to a specified number of rows after the current row in the specified column.

Parameters

- `sourceColumn` – The name of an existing column.
- `numRowsBefore` – A number of rows before the current source row, representing the start of the window.
- `numRowsAfter` – A number of rows after the current source row, representing the end of the window.
- `value` – The value for k .
- `targetColumn` – A name for the newly created column.

Example

```
{
  "Action": {
    "Operation": "ROLLING_KTH_LARGEST_UNIQUE",
    "Parameters": {
      "sourceColumn": "games_played",
      "numRowsBefore": "3",
      "numRowsAfter": "3",
      "value": "5",
      "targetColumn": "weight_kg_ROLLING_KTH_LARGEST_UNIQUE"
    }
  }
}
```

ROLLING_MAX

Returns in a new column the rolling maximum of values from a specified number of rows before to a specified number of rows after the current row in the specified column.

Parameters

- **sourceColumn** – The name of an existing column.
- **numRowsBefore** – A number of rows before the current source row, representing the start of the window.
- **numRowsAfter** – A number of rows after the current source row, representing the end of the window.
- **targetColumn** – A name for the newly created column.

Example

```
{
  "Action": {
    "Operation": "ROLLING_MAX",
    "Parameters": {
      "numRowsAfter": "10",
      "numRowsBefore": "10",
```

```
        "sourceColumn": "weight_kg",
        "targetColumn": "weight_kg_ROLLING_MAX"
    }
}
```

ROLLING_MIN

Returns in a new column the rolling minimum of values from a specified number of rows before to a specified number of rows after the current row in the specified column.

Parameters

- **sourceColumn** – The name of an existing column.
- **numRowsBefore** – A number of rows before the current source row, representing the start of the window.
- **numRowsAfter** – A number of rows after the current source row, representing the end of the window.
- **targetColumn** – A name for the newly created column.

Example

```
{
  "Action": {
    "Operation": "ROLLING_MIN",
    "Parameters": {
      "numRowsAfter": "10",
      "numRowsBefore": "10",
      "sourceColumn": "weight_kg",
      "targetColumn": "weight_kg_ROLLING_MIN"
    }
  }
}
```

ROLLING_MODE

Returns in a new column the rolling mode (most common value) from a specified number of rows before to a specified number of rows after the current row in the specified column.

Parameters

- `sourceColumn` – The name of an existing column.
- `numRowsBefore` – A number of rows before the current source row, representing the start of the window.
- `numRowsAfter` – A number of rows after the current source row, representing the end of the window.
- `modeType` – The modal function to apply to the window. Valid values are NONE, MINIMUM, MAXIMUM, and AVERAGE.
- `targetColumn` – A name for the newly created column.

Example

```
{
  "Action": {
    "Operation": "ROLLING_MODE",
    "Parameters": {
      "modeType": "MINIMUM",
      "numRowsAfter": "10",
      "numRowsBefore": "10",
      "sourceColumn": "weight_kg",
      "targetColumn": "weight_kg_ROLLING_MODE"
    }
  }
}
```

ROLLING_STANDARD_DEVIATION

Returns in a new column the rolling standard deviation of values from a specified number of rows before to a specified number of rows after the current row in the specified column.

Parameters

- `sourceColumn` – The name of an existing column.
- `numRowsBefore` – A number of rows before the current source row, representing the start of the window.
- `numRowsAfter` – A number of rows after the current source row, representing the end of the window.

- **targetColumn** – A name for the newly created column.

Example

```
{
  "Action": {
    "Operation": "ROLLING_STDEV",
    "Parameters": {
      "numRowsAfter": "10",
      "numRowsBefore": "10",
      "sourceColumn": "weight_kg",
      "targetColumn": "weight_kg_ROLLING_STDEV"
    }
  }
}
```

ROLLING_SUM

Returns in a new column the rolling sum of values from a specified number of rows before to a specified number of rows after the current row in the specified column.

Parameters

- **sourceColumn** – The name of an existing column.
- **numRowsBefore** – A number of rows before the current source row, representing the start of the window.
- **numRowsAfter** – A number of rows after the current source row, representing the end of the window.
- **targetColumn** – A name for the newly created column.

Example

```
{
  "Action": {
    "Operation": "ROLLING_SUM",
    "Parameters": {
      "numRowsAfter": "10",
```

```
        "numRowsBefore": "10",
        "sourceColumn": "weight_kg",
        "targetColumn": "weight_kg_ROLLING_SUM"
    }
}
```

ROLLING_VARIANCE

Returns in a new column the rolling variance of values from a specified number of rows before to a specified number of rows after the current row in the specified column.

Parameters

- **sourceColumn** – The name of an existing column.
- **numRowsBefore** – A number of rows before the current source row, representing the start of the window.
- **numRowsAfter** – A number of rows after the current source row, representing the end of the window.
- **targetColumn** – A name for the newly created column.

Example

```
{
  "Action": {
    "Operation": "ROLLING_VAR",
    "Parameters": {
      "numRowsAfter": "10",
      "numRowsBefore": "10",
      "sourceColumn": "weight_kg",
      "targetColumn": "weight_kg_ROLLING_VAR"
    }
  }
}
```

ROW_NUMBER

Returns in a new column a session identifier based on a window created by column names from "group by" and "order by" statements.

Parameters

- **groupByColumns** – A JSON-encoded string describing the "group by" columns.
- **orderByColumns** – A JSON-encoded string describing the "order by" columns.
- **targetColumn** – A name for the newly created column.

Example

```
{
  "Action": {
    "Operation": "ROW_NUMBER",
    "Parameters": {
      "groupByColumns": "[\"is public domain\"]",
      "orderByColumns": "[\"dimensions\"]",
      "targetColumn": "Row number"
    }
  }
}
```

SESSION

Returns in a new column a session identifier based on a window created by column names from "group by" and "order by" statements.

Parameters

- **sourceColumn** – The name of an existing column.
- **units** – A unit of measure for describe the session length. Valid values are MONTHS, YEARS, MILLISECONDS, QUARTERS, HOURS, MICROSECONDS, WEEKS, SECONDS, DAYS, and MINUTES.
- **value** – The number of units to define the time period.
- **groupByColumns** – A JSON-encoded string describing the "group by" columns.
- **orderByColumns** – A JSON-encoded string describing the "order by" columns.
- **targetColumn** – A name for the newly created column.

Example

```
{
```

```
"Action": {
  "Operation": "SESSION",
  "Parameters": {
    "sourceColumn": "object number",
    "units": "MINUTES",
    "value": "10",
    "groupByColumns": "[\"is public domain\"]",
    "orderByColumns": "[\"dimensions\"]",
    "targetColumn": "object number_SESSION",
  }
}
```

Web functions

Following, find reference topics for web functions that work with recipe actions.

Topics

- [IP_TO_INT](#)
- [INT_TO_IP](#)
- [URL_PARAMS](#)

IP_TO_INT

Converts the Internet Protocol version 4 (IPv4) value of the source column or other value to the corresponding integer value in the target column, and returns the result in a new column. This function works for IPv4 only.

For example, consider the following IP address.

```
192.168.1.1
```

If you use this value as an input to IP_TO_INT, the output value is as follows.

```
3232235777
```

Parameters

- `sourceColumn` – The name of an existing column.

- **value** – A character string to evaluate.
- **targetColumn** – The name of the new column to be created.

You can specify either **sourceColumn** or **value**, but not both.

Example

```
{
  "RecipeAction": {
    "Operation": "IP_TO_INT",
    "Parameters": {
      "sourceColumn": "my_ip_address",
      "targetColumn": "IP_TO_INT Column 1"
    }
  }
}
```

INT_TO_IP

Converts the integer value of source column or other value to the corresponding IPv4 value in then target column, and returns the result in a new column. This function works for IPv4 only.

For example, consider the following integer.

167772410

If you use this value as an input to **INT_TO_IP**, the output value is as follows.

10.0.0.250

Parameters

- **sourceColumn** – The name of an existing column.
- **value** – A character string to evaluate.
- **targetColumn** – The name of the new column to be created.

You can specify either **sourceColumn** or **value**, but not both.

Example

```
[ {
  "RecipeAction": {
    "Operation": "INT_TO_IP",
    "Parameters": {
      "sourceColumn": "my_integer",
      "targetColumn": "INT_TO_IP Column 1"
    }
  }
}
```

URL_PARAMS

Extracts query parameters from a URL string, formats them as a JSON object, and returns the result in a new column.

For example, consider the following URL.

```
https://example.com/?firstParam=answer&secondParam=42
```

If you use this value as an input to URL_PARAMS, the output value is as follows.

```
{"firstParam": ["answer"], "secondParam": ["42"]}
```

Parameters

- `sourceColumn` – The name of an existing column.
- `value` – A character string to evaluate.
- `targetColumn` – The name of the new column to be created.

You can specify either `sourceColumn` or `value`, but not both.

Example

```
{
```

```
"RecipeAction": {
  "Operation": "URL_PARAMS",
  "Parameters": {
    "sourceColumn": "my_url",
    "targetColumn": "URL_PARAMS Column 1"
  }
}
```

Other functions

Following, find reference topics for other functions that work with recipe actions.

Topics

- [COALESCE](#)
- [GET_ACTION_RESULT](#)
- [GET_STEP_DATAFRAME](#)

COALESCE

Returns in a new column the first non-null value found in the array of columns. The order of the columns listed in the function determines the order in which they're searched.

Parameters

- `sourceColumns` – A JSON-encoded string representing list of existing columns.
- `targetColumn` – The name of the new column to be created.

Example

```
{
  "RecipeAction": {
    "Operation": "COALESCE",
    "Parameters": {
      "sourceColumns": "[\"nation_position\",\"joined\"]",
      "targetColumn": "COALESCE Column 1"
    }
  }
}
```

```
}
```

GET_ACTION_RESULT

Fetches the result of a previously submitted action. Only for use in the interactive experience.

Parameters

- `actionId` – The `ActionId` returned in the original `SendProjectSessionAction` response.

Example

```
{
  "RecipeAction": {
    "Operation": "GET_ACTION_RESULT",
    "Parameters": {
      "actionId": "7",
    }
  }
}
```

GET_STEP_DATAFRAME

Fetches the data frame from a step in the project's recipe. Only for use in the interactive experience. Used with the `ViewFrame` parameter to paginate across a large data frame.

Parameters

- `stepIndex` – The index of the step in the project's recipe for which to fetch the data frame.

Example

```
{
  "RecipeAction": {
    "Operation": "GET_STEP_DATAFRAME",
    "Parameters": {
      "stepIndex": "0"
    }
  }
}
```

```
}
```

Quotas for AWS Glue DataBrew

You can view your DataBrew service quotas in the [AWS Service Quotas](#) console. You can also request a quota increase, for any quota that's adjustable.

Document history for AWS Glue DataBrew Developer Guide

Current API version: databrew-2017-07-25

The following table describes the documentation for this release of AWS Glue DataBrew. If you want to be notified when the *AWS Glue DataBrew Developer Guide* is updated, you can subscribe to the RSS feed.

Change	Description	Date
glue:GetCustomEntityType added to AWS managed policies	This permission is required to execute AWS Glue DataBrew profile jobs with PII-identification enabled. For more information, see AWS Glue DataBrew updates to AWS managed policies .	March 20, 2024
Support for multiple hashing algorithms in the CRYPTOGRAPHIC_HASH transformation	You can now specify a hashing algorithm when hashing values in a column. For more information, see CRYPTOGRAPHIC_HASH .	August 11, 2023
glue:BatchGetCustomEntityTypes added to AWS managed policies	This permission is required to execute AWS Glue DataBrew profile jobs with PII-identification enabled. For more information, see AWS Glue DataBrew updates to AWS managed policies .	May 9, 2022
Support for Apache ORC file format	DataBrew now supports Apache ORC as a file format for DataBrew data sources and outputs. For more	March 31, 2022

information, see [Supported file types for data sources](#).

[Support for cross-account AWS Glue Data Catalog Amazon S3 access](#)

You can now access AWS Glue Data Catalog S3 tables from other AWS accounts if an appropriate resource policy is created in the AWS Glue console. After creating a policy, the relevant Data Catalog S3 tables can be selected as input sources when creating a DataBrew dataset. For more information, see [Supported connections for data sources and outputs](#).

March 11, 2022

[Support for native console integration with Amazon AppFlow](#)

DataBrew now has native console integration with Amazon AppFlow. This integration means that you can connect to data from Salesforce, Zendesk, Slack, ServiceNow, and other software-as-a-service (SaaS) applications. You can also connect to data from AWS services such as Amazon S3 and Amazon Redshift. For more information, see [Supported connections for data sources and outputs](#).

November 18, 2021

Support for data quality rules	DataBrew now supports the creation of data quality rules, which are customizable validation checks that define business requirements for specific data. For more information, see Validating data quality in AWS Glue DataBrew .	November 18, 2021
Support for custom SQL statements	DataBrew now supports custom SQL statements for retrieving data from Amazon Redshift and Snowflake. This support means that you can use a purpose-built query to select and limit the data returned from large tables. For more information, see Supported connections for data sources and outputs .	November 18, 2021
Support for PII detection	DataBrew now supports detection of personally identifiable information (PII). This gives you the option of masking PII during data preparation. For more information, see Identifying and handling personally identifiable information (PII) .	November 18, 2021
Support for additional AWS Regions	DataBrew now supports additional AWS Regions. For a list of supported Regions, see AWS Glue DataBrew endpoints and quotas .	October 5, 2021

[Support for writing data to Lake Formation-based Amazon S3 tables](#)

DataBrew now supports writing data into AWS Glue Data Catalog S3 tables based on AWS Lake Formation. DataBrew also now supports writing data into Tableau Hyper format. For more information, see [Creating and working with AWS Glue DataBrew recipe jobs](#).

August 13, 2021

[Support for writing data into JDBC destinations](#)

DataBrew now supports writing data directly into JDBC-supported databases and data warehouses. These include Amazon Redshift, Snowflake, Microsoft SQL Server, MySQL, Oracle Database, and PostgreSQL. For more information, see [Creating and working with AWS Glue DataBrew recipe jobs](#).

July 23, 2021

[Support for specifying which data quality statistics are generated for a profile job](#)

DataBrew now supports specifying which data quality statistics are autogenerated for datasets in a profile job. For more information, see [Creating and working with AWS Glue DataBrew recipe jobs](#).

July 23, 2021

[Support for writing datasets into the AWS Glue Data Catalog](#)

DataBrew now includes support for writing datasets directly into the AWS Glue Data Catalog. You can choose to store datasets created from jobs that run your data preparation recipes in Amazon S3, Amazon Redshift, and Amazon RDS tables in the Data Catalog. The RDS tables supported include those for Amazon Aurora, RDS for Oracle, RDS for Microsoft SQL Server, RDS for MySQL, and RDS for PostgreSQL.

June 30, 2021

[Support for identifying advanced data types](#)

DataBrew now includes support to automatically identify and mark advanced data types for columns, which makes it easier to normalize columns that contain certain types of data. These types of data include Social Security number, email address, phone number, gender, credit card, URL, IP address, date and time, currency, ZIP code, country, region, state, and city.

June 30, 2021

[Support for using Amazon AppFlow to transfer data from SAAS applications](#)

DataBrew now supports using Amazon AppFlow to transfer data into Amazon S3 from third-party software-as-a-service (SaaS) applications such as Salesforce, Zendesk, Slack, and ServiceNow. For more information, see [Supported connections for data sources and outputs](#).

April 29, 2021

[Support for creating DataBrew datasets with input from JDBC databases](#)

DataBrew now supports creating datasets from data in JDBC-supported databases and data warehouses, including Amazon Redshift, Snowflake, Microsoft SQL Server, MySQL, Oracle Database, and PostgreSQL. For more information, see [Supported connections for data sources and outputs](#).

April 2, 2021

[Support for additional AWS Regions](#)

DataBrew now supports additional AWS Regions. For a list of supported Regions, see [AWS Glue DataBrew endpoints and quotas](#).

January 28, 2021

[New transforms for handling duplication](#)

Four new transforms for handling duplication have been added to the DataBrew console and API. For more information, see [DELETE_DUPLICATE_ROWS](#), [FLAG_DUPLICATE_ROWS](#), [FLAG_DUPLICATES_IN_COLUMN](#), and [REMOVE_DUPLICATES](#) in [Data quality recipe steps](#).

January 28, 2021

[Additional CSV delimiters](#)

DataBrew now supports additional delimiters besides commas in comma-separated value (CSV) files used to create DataBrew datasets. For more information, see [Creating and using AWS Glue DataBrew datasets](#).

January 28, 2021

[DataBrew extension for JupyterLab](#)

Now you can use AWS Glue DataBrew as an extension in JupyterLab. For more information, see [Using DataBrew as an extension in JupyterLab](#).

November 20, 2020

[New data preparation tool: AWS Glue DataBrew](#)

This is the first release of the *AWS Glue DataBrew Developer Guide*.

November 11, 2020

AWS Glossary

For the latest AWS terminology, see the [AWS glossary](#) in the *AWS Glossary Reference*.