PyBIDS Transformation Specification

version 1 (working copy)

This document contains a draft specification for a transformation language for use in BIDS Stats-Models. This is a working document in draft stage and any comments are welcome.

This specification is an extension of BIDS, and general principles are shared. The specification should work for many different settings and facilitate the integration with other imaging methods.

To see the original BIDS Stats-Models specification, see this link.

1 Overview

The PyBIDS Transformations specification provides a way for users to flexibly transform variables available in the workspace of an implementation of BIDS Stats-Models.

1.1 Top-level structure

This specification describes the pybids-transforms Transformer, and MUST be indicated in a BIDS Stats-Model document by the following fields:

Key name	Description
`Transformer`	REQUIRED. String. Value MUST be “pybids-transforms-v1”.
`Instructions`	REQUIRED. Array of transformation objects.

Transformation objects contain the following fields:

Key name	Description
`Name`	REQUIRED. String. The name of the transformation to apply. This MUST exactly match one of the values in the controlled vocabulary of available transformations.
`Input`	REQUIRED. Array of strings. An array of names identifying the variables to pass to the selected transformation operation.
`Output`	OPTIONAL. Array of strings. An array of variable names to use for the output(s) of the operation. By default, every transformation will either output

	to the same variable names provided in the input, or (for transformations that produce output with different dimensions than the inputs) will append a suffix to the existing variable name. The behavior of the output parameter is described below for each transformation. Note that if no output parameter is given, the expected behavior will be to overwrite the input variables in-place. If a single string is passed that contains an asterisk (*), the asterisk will be sequentially replaced with numbers and used as many times as there are output variables (i.e., if output="new_col_*", and there are 3 outputs, the variables will be named new_col_1, new_col_2, and new_col_3).

Each transformation may provide additional REQUIRED or OPTIONAL keys.

1.2 Variable model

Variables may be "sparse" (with onset and duration) or "dense" (with sampling rate). The collection of sparse variables may be considered equivalent to a data frame with onset and duration columns, specified in seconds, along with at least one additional column, each of which is an additional variable. A dense variable may be considered equivalent to a single-column data frame with a timeseries index starting at 0.

1.2 Transformation model

2.0 PyBIDS Transformation details

This appendix contains detailed documentation and usage information for all of the transformations listed in §4.4. Transformations are listed in alphabetical order.

Recap table:

https://docs.google.com/spreadsheets/d/1_BykMB9Uybe9javHDwGtQVBC7jqi36eTGpibJsGC8t4/edit#gid=0

And(Input, Output)
Or(Input, Output)

Each of these transformations takes 2 or more columns as input and performs the corresponding logical operation (inclusive or and conjunction, respectively) returning a single column as output. If non-boolean inputs are passed, it is expected that all zero (for numeric data types) and empty (for strings) values will evaluate to `false`, and all other values will evaluate to `true`.

Arguments:

- Input(list; mandatory): A list of 2 or more column names.
- Output(str; mandatory): The name of the output column.

Assign(**Input, Target, Output=None, InputAttr='value', TargetAttr='value'**)

The *Assign* transformation assigns one or more variables or columns (specified as the *input*) to one or more other columns (specified by target and/or output as described below).

Arguments:

- Input (list; mandatory): the name(s) of the columns from which attribute values are to be drawn (for assignment to the attributes of other columns). Must exactly match the length of the *target* argument.
- Target (list, mandatory): the name(s) of the columns to which the attribute values taken from the inputs are to be assigned. Must exactly match the length of the *input* argument. Names are mapped 1-to-1 from input to target. Note that if no output argument is specified, the columns named in target are modified in-place.
- Output (list; optional): optional names of the columns to output the result of the assignment to. Must exactly match the length of the *input* and *target* arguments. If no output list is provided, columns named in *target* are modified in-place. If an output list is provided, each column in the *target* list is first cloned, then the reassignment from the *input* to the *target* is applied; finally, the new (cloned and modified) column is written out to the column named in *output.*
- InputAttr (str or list; optional): specifies which attribute of the input column to assign. Must be one of 'value', 'onset', or 'duration'. Defaults to 'value'. If a list is passed, its length must exactly match that of the *input* and *target* lists.
- TargetAttr (str or list; optional): specifies which attribute of the output column to assign to. Must be one of 'amplitude', 'onset', or 'duration'. Defaults to value. If a list is passed, its length must exactly match that of the *input* and *target* lists.

Examples:

To reassign the value property of a variable named 'RT' to the duration property of a 'face' variable (as one might do in order to, e.g., model trial-by-trial reaction time differences for a given condition using a varying-epoch approach), and write it out as a new "face_modulated_by_RT" column:

```
{
  "Name": "Assign",
  "Input": ["response_time"],
  "Target": ["face"],
  "TargetAttr": "duration",
  "Output": ["face_modulated_by_RT"]
}
```

Notes:

This transformation is non-destructive with respect to the input column(s). In case where in-place assignment is desired (essentially, renaming a column), either use the *rename* transformation, or set output to the same value as the input.

Convolve(Input, Output=None, Model='spm', Derivative=False, Dispersion=False, FirDelays=None)

Convolves one or more variables with the specified hemodynamic response function.

Arguments:

- Input (list; mandatory): The name(s) of the variable(s) to convolve.
- Output (list; optional): the optional list of column names to write out to. By default, computation is done in-place (i.e., input columnise overwritten).
- Model (str; optional): The name of the HRF model to apply. Must be one of 'spm', 'glover', or 'fir'. Defaults to 'spm'.
    - FSL: 'Double-Gamma HRF', 'Gaussian HRF' (bad practice - probably to exclude), 'Gamma HRF', 'Custom basis set', 'Gamma basis set', 'Sine basis set', 'FIR basis set'
    - SPM: 'Canonical HRF'
    - AFNI: 'BLOCK', 'GAM', 'TWOGAM', 'SPMG1', 'WAV', 'MION' (more info here)
- Derivative (bool; optional): Whether or not to include the temporal derivative. Defaults to False.
- Dispersion (bool; optional): Whether or not to include the dispersion derivative. Defaults to False.
- FIRDelays (list; optional): A list of delays (with each element giving the delay in seconds from event onset) to use if model is 'fir' (ignored otherwise). Spacing between delays must be fixed.

Notes:

- If temporal derivatives, dispersion derivatives, or FIR delays are used, additional variables are added to the design matrix. These are **not** automatically included in contrasts, and must be explicitly added to each contrast as desired.
- Important implementation detail: convolution MUST NEVER cross run boundaries, no matter how else the design matrix is constructed or specified.

Copy(Input, Output)

Clones/copies each of the input columns to a new column with identical values and a different name. Useful as a basis for subsequent transformations that need to modify their inputs in-place.

Arguments:

- input (list; mandatory): A list of column names to copy.
- output (list; mandatory): A list of the names to copy the input columns to. Must be same length as input, and columns are mapped one-to-one from the input list to the output list.

Delete(Input)

Deletes column(s) from further analysis.

Arguments:

- Input (list; mandatory): The name(s) of the variable(s) to delete.

Notes: The “Select” transformation provides the inverse function: selection of columns to keep for subsequent analysis.

Demean(Input, Output=None):

Mean-centers the input column. Effectively just an alias for calling scale(input, output, demean=True, scale=False), so see scale() documentation for further details.

Arguments:

- Input(list; mandatory): The name(s) of the variable to operate on.
- Output (list; optional): the optional list of column names to write out to. By default, computation is done in-place (i.e., input columnise overwritten).

DropNA(Input, Output=None):

Drops all rows with ‘n/a’.

Arguments:

- Input(list; mandatory): The name of the variable to operate on.
- Output (list; optional): the optional list of column names to write out to. By default, computation is done in-place (i.e., input columnise overwritten).

Factor(Input, Prefix=None, Constraint=’none’, RefLevel=None)

Converts a nominal/categorical variable with N unique levels to either N or N-1 binary indicators (i.e., dummy-coding).

Arguments:

- Input (list; mandatory): the name(s) of the variable(s) to dummy-code.
- Prefix (list, optional): an optional list of prefixes when naming levels of the factorized columns. Length of prefix must match the length of the input list, and columns are mapped one-to-one. If left unspecified, each column will be assigned the name of the value, prefixed with the original column name. E.g., a variable named “condition” with levels “A”, “B”, and “C” would produce three binary columns named “condition.A”, “condition.B”, and “condition.C” by default.
- Constraint (string; optional): Constraints to use when creating dummy variables. Must be one of “none”, “drop_one” or “mean_zero”. Value “none” (default) will result in N unique levels creating N dummy variables; “drop_one” will omit one level creating N-1 dummy variables; “mean_zero” will create N-1 dummy variables implementing the constraint that the sum of the N levels is mean zero. “Drop_one” and “mean_zero” require a reference level; for “drop_one” the reference level is the level omitted (this factor level is then modeled with the grand mean of the model); for “mean_zero” the reference level is likewise omitted (it can be estimated as one minus the sum of the other factor level estimates).
- RefLevel (string; optional): For “drop_one” and “mean_zero” constraints, the level used to determine the mapping from N to N-1 factor levels. By default it is the first factor level when sorting in alphabetical order (e.g., if a condition has levels ‘dog’, ‘apple’, and ‘helsinki’, the default reference level will be ‘apple’).

Filter(Input, Query, By=None, Output=None):

Subsets rows using a boolean expression.

Arguments:

- Input(list; mandatory): The name(s) of the variable(s) to operate on.
- Query(str; mandatory): Boolean expression used to filter
- By(str; optional): Name of column to group filter operation by
- Output (list; optional): the optional list of column names to write out to. By default, computation is done in-place (i.e., input columnise overwritten). If provided, the number of values must exactly match the number of input values, and the order will be mapped 1-to-1.

Group(Input, Name):

Creates a new variable group that can be used as an alias for the named variables.

Arguments:

- Input(list, mandatory): The names of the variables to include in the group.
- Name(string, mandatory): The name of the new variable group. Cannot be the same as the name of any existing variable.

Lag(Input, Output=None, Shift=1, Order=3, Mode="nearest", Constant=0, Difference=False):

Returns a variable that is lagged by a specified number of time points. Spline interpolation of the requested order is used for non-integer shifts. Points outside the input are filled according to the given mode. Negative shifts move values toward the beginning of the sequence. If Difference is true, the backward difference is calculated for positive shifts, and the forward difference is calculated for negative shifts.

Arguments:

- Input(list, mandatory): The name of the variable to operate on.
- Output (list, optional): the optional list of column names to write out to. By default, computation is done in-place (i.e., input columnise overwritten).
- Shift (number, optional): The number of places to shift the values. By default, shifts each value one forward in the array.
- Order (integer, optional): The order or spline interpolation. Must be in range 0-5. Default is 3.
- Mode (string, optional): The mode parameter determines how the input variable is extended beyond its boundaries. Default is "nearest". The following values are accepted:
    - "nearest" - (a a a a | a b c d | d d d d) - The input is extended by replicating the boundary values
    - "reflect" - (d c b a | a b c d | d c b a) - The input is extended by reflecting the array over the edge
    - "constant" - (k k k k | a b c d | k k k k) - The input is extended by filling all values beyond the edge with the same constant value, defined by the Constant parameter
- Constant (number, optional): Value to fill past edges of input if Mode is "constant". Default is 0.
- Difference (boolean, optional): Calculate the backward (if shift is positive) or forward (if shift is negative) difference. For the forward difference dx of an array x, dx[i] = x[i+1] - x[i]. For the backward difference dx of an array x, dx[i] = x[i] - x[i-1].

Additional modes may be defined if there is need for them. The scipy shift method provides the reference implementation for all current modes. "Constant" is equivalent to the shift parameter "cval".

Log10(Input, Output=None):

[TODO]

Arguments:

- Input(list, mandatory): The name of the variable to operate on.
- Output (list, optional): the optional list of column names to write out to. By default, computation is done in-place (i.e., input columnise overwritten).

Not(Input, Output=None)

Returns the logical negation of the input column(s). Uses Python-like boolean semantics. That is, for every value that evaluates to True (i.e., all non-zero or non-empty values), return 0, and for every value that evaluates to False (i.e., zero or empty string) return 1.

Arguments:

- Input(list, mandatory): A list containing one or more column names.
- Output(list, optional): An optional list of output column names. Must match the input list in length, and column names will be mapped 1-to-1. If no output argument is provided, defaults to in-place transformation (i.e., each input column will be overwritten).

Or(Input, Output=None)

See And for details

Orthogonalize(Input, Wrt, Output=None, Demean=False)

The orthogonalize transformation orthogonalizes one or more input columns with respect to one or more other columns.

Arguments:

- Input (list, mandatory): the names of the columns to orthogonalize. Note that if more than one value is passed, the same orthogonalization (defined by the terms in the ‘wrt’ argument) will be independently applied to each column named in the list.
- Wrt (list, mandatory): stands for “with respect to”; the names of the columns to orthogonalize each of the input columns with respect to. Note that because orthogonalization is applied independently to each input (see above), *there must not be any overlap between values in* input *and values in* wrt.

- Output (list, optional): the optional list of column names to write out to. By default, orthogonalization is done in-place on the inputs (i.e., input columns are overwritten). If provided, the number of values must exactly match the number of input values, and the order will be mapped 1-to-1.
- Demean (bool, optional): Specifies whether or not to demean both the target column and the columns in the wrt argument before orthogonalization.

Product(Input, Output)

Computes the row-wise product of two or more columns.

Arguments:

- Input(list, mandatory): Names of two or more columns to compute the product of.
- Output(str, mandatory): Name of the newly generated column.

Rename(Input, Output):

Rename a variable.

Arguments:

- Input(list, mandatory): The name(s) of the variable(s) to rename.
- Output (list, mandatory): New column names to output. Must match the length of input column(s), and columns will be mapped 1-to-1 in order.

Replace(Input, Replace, Attribute='value', Output=None)

Replaces values in one or more input columns.

Arguments:

- Input (list, mandatory): Name(s_ of column(s) to search and replace within.
- Replace (object, mandatory): An associative array (dictionary) mapping old values to new values. For example, {"apple": "bee", "elusive": 5} would replace all occurrences of 'apple' in the input columns with the value 'bee', and all occurrences of 'elusive' with the value 5.
- Attribute (string, optional): The column attribute to search/replace. Valid values include 'value' (the default), 'duration', 'onset', and 'all'. In the last case, all three attributes (value, duration, and onset) will be scanned. Note that level names for categorical columns (e.g., 'trial_type') are invariably represented in the value attribute.

- Output (list, optional): Optional names of columns to output. Must match length of input column(s) if provided, and columns will be mapped 1-to-1 in order. If no output values are provided, the replacement transformation is applied in-place to all the inputs.

Resample(Input, SamplingRate, Output=None):

Resamples one or more columns to a specified sampling rate. If the target sampling rate is lower than the sampling rate of the input variable, an anti-aliasing filter must be used.

Arguments:

- Input(list, mandatory): The name of the variable to resample.
- SamplingRate(float, mandatory): Sampling frequency in hertz.
- Output (list, optional): the optional list of column names to write out to. By default, computation is done in-place (i.e., input columnise overwritten).

Scale(Input, Demean=True, Rescale=True, ReplaceNa=None, Output=None)

Scales the values of one or more columns. Semantics mimic scikit-learn, such that demeaning and rescaling are treated as independent arguments, with the default being to apply both (i.e., standardizing each value so that it has zero mean and unit SD).

Arguments:

- Input(list, mandatory): Names of columns to standardize.
- Demean(bool, optional): If True, subtracts the mean from each input column (i.e., applies mean-centering).
- Rescale(bool, optional): If True, divides each column by its standard deviation.
- ReplaceNa(string, optional). Whether/when to replace missing values with 0. If None, no replacement is performed. If 'before', missing values are replaced with 0's before scaling. If 'after', missing values are replaced with 0 after scaling.
- Output(list, optional): Optional names of columns to output. Must match length of input column if provided, and columns will be mapped 1-to-1 in order. If no output values are provided, the scaling transformation is applied in-place to all the inputs.

Select(Input)

The *select* transformation specifies which columns to retain for subsequent analysis. Any columns that are not specified here will be dropped.

Arguments:

- Input (list, mandatory): The names of all columns to keep. Any columns not in this list will be deleted and will not be available to any subsequent transformations or downstream analyses.

Notes: one can think of select as the inverse the“Delete” transformation that removes all named columns from further analysis.

Split(Input, By, Sep='.', Output=None)

Split a variable into N variables as defined by the levels of one or more other variables.

Arguments:

- Input(list, mandatory): The name of the variable(s) to operate on.
- By(list, mandatory): Name(s) for variable(s) to split on.
- Output (list, optional): the optional list of column names to write out to. If an output list is provided, it must have the same number of values as the number of generated columns. If no output list is provided, name components will be separated by a period, and values of variables will be enclosed in square brackets. For example, given a variable `Condition` that we wish to split on two categorical columns `A` and `B`, where a given row has values `A=a` and `B=1`, the generated name will be `Condition.A[a].B[`

Sum(Input, Output, Weights=None)

Computes the (optionally weighted) row-wise sums of two or more columns.

Arguments:

- Input(list, mandatory): Names of two or more columns to sum.
- Output(str, mandatory): Name of the newly generated column.
- Weights(list, optional): Optional list of floats giving the weights of the columns. If provided, length of weights must equal the number of values in input, and weights will be mapped 1-to-1 onto named columns. If no weights are provided, defaults to unit weights (i.e., simple sum).

Threshold(Input, Threshold=0, Binarize=False, Above=True, Signed=True, Output=None)

Thresholds input values at a specified cut-off and optionally binarizes the result.

Arguments:

- Input(list, mandatory): The name(s)of the column(s) to threshold/binarize.
- Threshold(float, optional): The cut-off to use for thresholding. Defaults to 0.
- Binarize(boolean, optional): If True, thresholded values will be binarized (i.e., all non-zero values will be set to 1). Defaults to False.
- Above(boolean, optional): Specifies which values to retain with respect to the cut-off. If True, all value above the threshold will be kept; if False, all values below the threshold will be kept. Defaults to True.
- Signed(boolean, optional): Specifies whether to treat the threshold as signed (default) or unsigned. For example, when passing above=True and threshold=3, if signed=True, all and only values above +3 would be retained. If signed=False, all absolute values > 3 would be retained (i.e.,values in the range -3 < X < 3 would be set to 0).
- Output(list, optional): Optional names of columns to output. Must match length of input column if provided, and columns will be mapped 1-to-1 in order. If no output values are provided, the threshold transformation is applied in-place to all the inputs.

ToDense(Input, SamplingRate=None, Output=None)

Convert one or more variables from a sparse representation to a dense one. This entails taking the onset/duration/values for sparsely-represented variables/conditions, and converting them to a dense format where the number of observations is equal to the number of acquisition volumes, with one value per volume. Note that this transformation is irreversible: one can go from sparse to dense, but not the converse.

Arguments:

- Input (list, mandatory): Name(s) of variable(s) to convert from sparse to dense format.
- SamplingRate (float): Optional sampling rate to use for the densified variable. Specified in Hz. If left unspecified, the default value is left to the implementing software to fill in (with the expectation that it will be something reasonable, e.g., oversampling the TR by a factor of 10 or more.)'
- Output (list, optional): Optional name(s) of output variables. If provided, length must equal the length of the input list, and columns are mapped 1-to-1 in order. If no output argument is passed, variables in the input list are modified in-place.

Not implemented

Add(Input, Value, Output=None)
Divide(Input, Value, Output=None)
Multiply(Input, Value, Output=None)
Subtract(Input, Value, Output=None)

Each of these transformations takes one or more columns, and performs a mathematical operation on the input column and a provided operand. The operations are performed on each column independently.

Arguments:

- Input(list; mandatory): A list of columns to perform operation on.
- Value(float or str; mandatory): The value to perform operation with (i.e. operand)
- Output (list; optional): the optional list of column names to write out to. By default, computation is done in-place on the inputs (i.e., input columns are overwritten). If provided, the number of values must exactly match the number of input values, and the order will be mapped 1-to-1.

Censor(By, Input='*')

Censor (zero-out) input columns where the censoring column(s) are non-zero.

Arguments:

- By(list; mandatory): Name of indicator columns used to censor inputs.
- Input(list; optional): Set of columns to be censored. Default: all columns (except censoring

Derivative(Input, Output=None, Order=[1], Initial=NaN)

Computes the temporal derivative of each input column using backward differences. That is, for every value in the input column, returns the difference between that value and the preceding value.

Arguments:

- Input(list; mandatory): A list containing 1 or more column names.
- Output(list; optional): Optional names of columns to output. Length must be equal to the product of the lengths of `input` and `order` if provided. Columns will be mapped using an outer loop over `order` and an inner loop over `input`. If no output values are provided, the derivative columns will automatically be named by appending `_derivative<order>` to the input column names.
- Order(list(int); optional): The order of temporal derivative to compute via backward differences. For instance, a value of [2,3] will compute the second and third temporal derivatives.
- Initial(float; optional): For a temporal derivative of order n, that derivative cannot be computed for the first n time points. Instead, the new derivative column will be assigned the value specified here (NaN by default).

Formula(Input, Formula, Output=None):

[TODO]

Arguments:

- Input(list, mandatory): The name(s) of the variable(s) to apply formula to.
- Formula(string; mandatory): The formula to apply
- Output (list; optional): the optional list of column names to write out to. By default, computation is done in-place (i.e., input columnise overwritten). If provided, the number of values must exactly match the number of input values, and the order will be mapped 1-to-1.

Image(Input, Output, Mask, Aggregate)

Returns a variable group containing one variable per voxel or region in the input timeseries image. An input mask (specified as discrete labels on a 3D nifti or as fuzzy partitions in a 4D nifti, MNI coordinates + radius, etc.) and aggregation function can be provided that specify which voxels to select and/or how to aggregate them. Note that this operation is not structure-preserving—i.e., once timecourses are loaded, the user should have no expectation of being able to project back into image space.

Arguments:

- Input (dictionary, mandatory): 4D image from which time series will be extracted. The image must be specified as a set of BIDS-compliant selectors that identify one or more images, NOT as a filename. E.g., `{"desc": "myTargetMask", "space": "MNI152Lin"}`. If more than one image is matched, timecourses from images will be concatenated into the same result (i.e., if 2 images are matched, the first has 3 timecourses, and the second has 6 timecourses, the returned data will contain 9 columns in total).
- Output (list, mandatory): Name(s) to use for output columns. If more than one element is provided, length of list must match the number of returned columns. If a single value is provided, it is interpreted as a prefix, and sequential numbers are appended (i.e., “MyTimecourse” will return columns named “MyTimecourse1”, “MyTimecourse2”, etc.).
- Mask (dictionary, optional): Image specifying spatial location(s) from which time series will be extracted. If none is provided, a separate timecourse is returned for every available voxel that contains at least one non-zero value in its timecourse. The image must be specified as a set of BIDS-compliant selectors that identify one or more images, NOT as a filename. E.g., `{"desc": "MyFavoriteROIs"}`
- Aggregate (str or boolean, optional): Method of combining time series within each mask. The following values are valid:
    - “mean” (default): Returns the average of all voxels within each discrete non-zero value found in the image.
    - “pca”: Returns the first principal component of all voxels within each discrete non-zero value found in the image.

    - “none”: No dimensionality reduction; a separate timecourse is returned for each voxel that contains at least one non-zero value in its timecourse.

Mean(Input, Output=None):

Compute mean of a column.

Arguments:

- Input(string, mandatory): The name of the variable to operate on.
- Output (string, optional): the optional list of column names to write out to. By default, computation is done in-place (i.e., input columnise overwritten).

Power(Input, Output, Order=2)

Elevates each value in each input column to a specified power. This can be used, for instance, to generate higher-order terms for confound regression.

Arguments:

- Input(list, mandatory): A list containing 1 or more column names.
- Output(list, optional): Optional names of columns to output. Length must be equal to the product of the lengths of `input` and `order`. If provided, columns will be mapped using an outer loop over `order` and an inner loop over `input`. If no output values are specified, higher-order terms will automatically be named by appending `_power<order>` to the input column names.
- Order(list(int), optional): The order of power to which input terms are elevated. For instance, a value of [2, 3] will compute the second and third powers. By default, only quadratic terms are computed.

PCA(Input, Output=None):

[TODO]

Arguments:

- Input(string, mandatory): The name of the variable to operate on.
- Output (string, optional): the optional list of column names to write out to. By default, computation is done in-place (i.e., input columnise overwritten).

StdDev(Input, Output=None)

Compute the sample standard deviation.

Arguments:

- Input(list, mandatory): The name(s) of the variable(s) to operate on.
- Output (list, optional): Optional names of columns to output. Must match length of input column if provided, and columns will be mapped 1-to-1 in order. If no output values are provided, the transformation is applied in-place to all the inputs.

Variance(Input, SplitBy, NestedIn=None, Constraint='none', Reference=None, Value=1)

The Variance transformation facilitates creation of indicator columns specifying group memberships. It is primarily useful for generating columns of the Z matrix used in mixed-effects models (where $y = X\beta + Zu + \epsilon$). The transformation returns one or more variable groups. These groups can then be passed into the `Z` field of the `model` section (when `AnalysisType='glm'`) to specify variance components.

Arguments:

- `Input (string, mandatory)`: The name(s) of the existing variable that defines the grouping structure. For example, passing "`subject`" would indicate that all observations that share the same value of the `subject` variable are part of the same group. Mandatory.

- `Output (string, mandatory)`: The name to give to the generated variable group(s). If more than one group is generated (e.g., because of the use of `SplitBy`), the `Output` is used as the prefix (see `SplitBy` and `NestedIn` for suffix naming details).

- `SplitBy (string or list)`: Optional name(s) indicating that a separate variable group should be defined for each unique value of the passed variables. For example, passing "`condition`" as the `SplitBy` variable would create a separate variable group for each unique condition. The resulting variable groups will have names that folow the convention "`[Output].[level]`" (e.g., if `Output` is `subject_slope` and the levels of `SplitBy` are `A` and `B`, we will end up with variable groups named `subject_slope.A` and `subject_slope.B`.

- `NestedIn (string or list)`: Optional string(s) naming one or more variables within which the `GroupBy` variable is nested. For most purposes, `NestedIn` is interchangeable with `SplitBy`. However, there are cases where using both options independently can substantially simplify specification of variance components.

- `Constraint (boolean)`: Optional indicator specifying how to estimate the variable groups (and how many to generate) in the case that more than one is to be returned. Must be one of "none", "drop_one" or "mean_zero". Value "none" (default) will result in N unique levels creating N variable groups; "drop_one" will omit one level creating N-1 variable groups; "mean_zero" will create N-1 variable groups implementing the constraint that the sum of the N levels is mean zero. "drop_one" and "mean_zero" require a

reference level (passed as `RefLevel`); for “drop_one” the reference level is the level omitted (this factor level is then modeled with the grand mean of the model); for “mean_zero” the reference level is likewise omitted (it can be estimated as one minus the sum of the other factor level estimates).

- `RefLevel (string)`: For “drop_one” and “mean_zero” constraints, the level used to determine the mapping from N to N-1 factor levels. By default it is the first factor level when sorting in alphabetical order (e.g., if a condition has levels ‘dog’, ‘apple’, and ‘helsinki’, the default reference level will be ‘apple’).

- `Value (int, string)`: Optional value to assign to each non-zero cell in the generated grouping variables. If not explicitly provided, defaults to `1`. If a string is passed, it must give the name of an existing variable containing the values to use.