Crunch Cube Documentation

Release 3.0.35

Crunch.io

Sep 06, 2023

Contents

1 Installation **3**

2 A quick example **5**

3 For developers **7**

3.1 Quick Start . . . 7

3.2 Cube Objects . . . 9

3.3 Partition Objects . . . 12

3.4 Dimension objects . . . 28

4 Indices and tables **31**

Index **33**

Crunch Cube allows you to manipulate cube responses from the Crunch API using Python. We'll refer to these cube responses as *cubes* in the subsequent text. When used in conjunction with *pycrunch*, this library can unlock powerful second-order analytics and visualizations.

A *cube* is obtained from the *Crunch.io* platform as a JSON response to a specific *query* created by a user. The most common usage is to obtain the following:

- Cross correlation between different variables
- Margins of the cross-tab *cube*
- Proportions of the cross-tab *cube* (e.g. proportions of each single element to the entire sample size)

Crunch Cube allows you to access these values from a cube response without dealing with the complexities of the underlying JSON format.

The data in a cube is often best represented in a table-like format. For this reason, many API methods return data as a *numpy.ndarray* object.

CHAPTER 1

Installation

The Crunch Cube package can be installed via *pip install*:

```
pip install cr.cube
```

CHAPTER 2

A quick example

After the *cr.cube* package has been successfully installed, the usage is as simple as:

```
>>> from cr.cube.cube import Cube

>>> ### Obtain the crunch cube JSON payload using app.crunch.io, pycrunch, rcrunch or
↪scrunch
>>> ### And store it in the 'cube_JSON_response' variable

>>> cube = Cube(cube_JSON_response)
>>> print(cube)
Cube(name='MyCube', dimension_types='CAT x CAT')
>>> cube.counts
np.array([[1169, 547],
          [1473, 1261]])
```

CHAPTER 3

For developers

For development mode, Crunch Cube needs to be installed from the local checkout of the *crunch-cube* repository. Navigate to the top-level folder of the repo, on the local file system, and run:

```
$ python setup.py develop
$ py.test tests -cov=cr.cube
```

Note that we are happy to accept pull requests, please be certain that your code has proper coverage before submitting. All pull requests will be tested by travis.

3.1 Quick Start

In the Crunch system, any analysis is also referred to as a `cube`. Cubes are the mechanical means of representing analyses to and from the Crunch system; you can think of them as spreadsheets that might have other than two dimensions. A cube consists of two primary parts: "dimensions" which supply the cube axes, and "measures" which populate the cells. Although both the request and response include dimensions and measures, it is important to distinguish between them. The request supplies expressions for each, while the response has data (and metadata) for each. The request declares what variables to use and what to do with them, while the response includes and describes the results.

At an abstract level, cubes contain arrays (`numpy arrays`) of measures. Measures frequently (although not always!) are simply counts of responses that fall into each cell of the cross-tabulation (also sometimes called contingency tables). Cubes always include the unweighted counts which are important for some analyses, or could contain other measures which are treated differently.

Check out the details here

3.1.1 Installation

The Crunch Cube package can be installed via *pip install*:

```
pip install cr.cube
```

3.1.2 Cube object

Below a quick example on how instanciate and query the counts of a *cube*

```
>>> from cr.cube.cube import Cube

>>> ### Obtain the crunch cube JSON payload using app.crunch.io, pycrunch, rcrunch or␣
↪scrunch
>>> ### And store it in the 'cube_JSON_response' variable

>>> cube = Cube(cube_JSON_response)
>>> print(cube)
Cube(name='MyCube', dimension_types='CAT x CAT')
>>> cube.counts
np.array([[1169, 547],
          [1473, 1261]])
```

If the JSON response includes both `weighted` and `unweighted_counts`, `cube.counts` corresponds to the weighted version of the counts; but we still have both measures:

```
>>> cube.counts
np.array([[1122.345, 234.456,
         1432.2331, 1211.8763]])
>>> cube.unweighted_counts
np.array([[1169, 547],
          [1473, 1261]])
```

3.1.3 Cube Partitions

A `cube` can contain 1 or more partitions according to its dimensionality. For example a CAT_X_CAT cube has a single 2D partition, identified as a *Slice* object in the *cubepart* module, a CA_SUBVAR_X_CA_CAT cube has two 2D partitions that can be represented like:

```
>>> cube.partitions[0]
_Slice(name='pets_array', dimension_types='CA_SUBVAR x CA_CAT')
Showing: COUNT
          not selected    selected
------  --------------  ----------
cat                 13          12
dog                 16          12
wombat              11          12
Available measures: [<CUBE_MEASURE.COUNT: 'count'>]
>>> cube.partitions[1]
_Slice(name='pets_array', dimension_types='CA_SUBVAR x CA_CAT')
Showing: COUNT
          not selected    selected
------  --------------  ----------
cat                 32          22
```

(continues on next page)

(continued from previous page)

```
dog                   24           28
wombat                21           26
Available measures: [<CUBE_MEASURE.COUNT: 'count'>]
```

Let's back to the CAT_X_CAT cube, the example below shows how to access to some of the avilable measures for the analyses.

```
>>> cube = Cube(cube_JSON_response_CAT_X_CAT)
>>> partition = cube.partition[0]
>>> partition.column_proportions
array([[0.5, 0.4],
       [0.5, 0.6]])
>>> partition.column_std_dev
array([[0.5       , 0.48989795],
       [0.5       , 0.48989795]])
>>> partition.columns_scale_mean
array([1.5, 1.6])
```

For the complete measure references visit the Partition API

3.2 Cube Objects

3.2.1 Cube

class cr.cube.cube.**Cube**(*response: Union[str, Dict[KT, VT]], cube_idx: Optional[int] = None, transforms: Optional[Dict[KT, VT]] = None, population: Optional[int] = None, mask_size: int = 0*)

Provides access to individual slices on a cube-result.

It also provides some attributes of the overall cube-result.

cube_idx must be *None* (or omitted) for a single-cube CubeSet. This indicates the CubeSet contains only a single cube and influences behaviors like CA-as-0th.

available_measures
frozenset of available CUBE_MEASURE members in the cube response.

counts_with_missings
ndarray of weighted, unweighted or valid counts including missing values.

The difference from .counts is that this property includes value for missing categories.

covariance
Optional float64 ndarray of the cube_covariance if the measure exists.

cube_index
Offset of this cube within its CubeSet.

description
Return the description of the cube.

dimension_types
Tuple of DIMENSION_TYPE member for each dimension of cube.

dimensions
List of visible dimensions.

A cube involving a multiple-response (MR) variable has two dimensions for that variable (subvariables and categories dimensions), but is "collapsed" into a single effective dimension for cube-user purposes (its categories dimension is supressed). This collection will contain a single dimension for each MR variable and therefore may have fewer dimensions than appear in the cube response.

has_weighted_counts
True if cube response has weighted count data.

inflate() → cr.cube.cube.Cube
Return new Cube object with rows-dimension added.

A multi-cube (tabbook) response formed from a function (e.g. mean()) on a numeric variable arrives without a rows-dimension.

means
Optional float64 ndarray of the cube_means if the measure exists.

missing
Get missing count of a cube.

n_responses
Total (int) number of responses considered.

name
Return the name of the cube.

If the cube has 2 diensions, return the name of the second one. In case of a different number of dimensions, default to returning the name of the last one. In case of no dimensions, return the empty string.

ndim
int count of dimensions for this cube.

overlaps
Optional float64 ndarray of cube_overlaps if the measure exists.

The array has as many dimensions as there are defined in the cube query, plus the extra subvariables dimension as the last dimension.

partitions
Sequence of _Slice, _Strand, or _Nub objects from this cube-result.

population_fraction
The filtered/unfiltered ratio for cube response.

This value is required for properly calculating population on a cube where a filter has been applied. Returns 1.0 for an unfiltered cube. Returns *np.nan* if the unfiltered count is zero, which would otherwise result in a divide-by-zero error.

stddev
Optional float64 ndarray of the cube_stddev if the measure exists.

sums
Optional float64 ndarray of the cube_sum if the measure exists.

title
str alternate-name given to cube-result.

This value is suitable for naming a Strand when displayed as a column. In this use-case it is a stand-in for the columns-dimension name since a strand has no columns dimension.

unweighted_counts
ndarray of unweighted counts, valid elements only.

Unweighted counts are drawn from the *result.counts* field of the cube result. These counts are always present, even when the measure is numeric and there are no count measures. These counts are always unweighted, regardless of whether the cube is "weighted".

In case of presence of valid counts in the cube response the counts are replaced with the valid counts measure.

unweighted_valid_counts
Optional float64 ndarray of unweighted_valid_counts if the measure exists.

valid_counts_summary_range
Optional (min, max) tuple of summary valid counts

valid_overlaps
Optional float64 ndarray of cube_valid_overlaps if the measure exists.

The array has as many dimensions as there are defined in the cube query, plus the extra subvariables dimension as the last dimension.

weighted_counts
ndarray of weighted counts, valid elements only.

In case of presence of valid counts in the cube response the weighted counts are replaced with the valid counts measure.

weighted_valid_counts
Optional float64 ndarray of weighted_valid_counts if the measure exists.

3.2.2 CubeSet

class cr.cube.cube.**CubeSet**(*cube_responses: List[Dict[KT, VT]], transforms: Dict[KT, VT], population: int, min_base: int*)

Represents a multi-cube cube-response.

Also works just fine for a single cube-response passed inside a sequence, allowing uniform handling of single and multi-cube responses.

cube_responses is a sequence of cube-response dicts received from Crunch. The sequence can contain a single item, such as a cube-response for a slide, but it must be contained in a sequence. A tabbook cube-response sequence can be passed as it was received.

transforms is a sequence of transforms dicts corresponding in order to the cube-responses. *population* is the estimated target population and is used when a population-projection measure is requested. *min_base* is an integer representing the minimum sample-size used for indicating values that are unreliable by reason of insufficient sample (base).

available_measures
frozenset of available measures of the first cube in this set.

can_show_pairwise
True if all 2D cubes in a multi-cube set can provide pairwise comparison.

description
str description of first cube in this set.

has_weighted_counts
True if cube-responses include a weighted-count measure.

is_ca_as_0th
True for multi-cube when first cube represents a categorical-array.

A "CA-as-0th" tabbook tab is "3D" in the sense it is "sliced" into one table (partition-set) for each of the CA subvariables.

missing_count
The number of missing values from first cube in this set.

n_responses
Total number of responses considered from first cube in this set.

name
str name of first cube in this set.

partition_sets
Sequence of cube-partition collections across all cubes of this cube-set.

This value might look like the following for a ca-as-0th tabbook. For example:

```
(
    (_Strand, _Slice, _Slice),
    (_Strand, _Slice, _Slice),
    (_Strand, _Slice, _Slice),
)
```

and might often look like this for a typical slide:

```
((_Slice,))
```

Each partition set represents the partitions for a single "stacked" table. A 2D slide has a single partition-set of a single _Slice object, as in the second example above. A 3D slide would have multiple partition sets, each of a single _Slice. A tabook will have multiple partitions in each set, the first being a _Strand and the rest being _Slice objects. Multiple partition sets only arise for a tabbook in the CA-as-0th case.

population_fraction
The filtered/unfiltered ratio for this cube-set.

This value is required for properly calculating population on a cube where a filter has been applied. Returns 1.0 for an unfiltered cube. Returns *np.nan* if the unfiltered count is zero, which would otherwise result in a divide-by-zero error.

valid_counts_summary_range
The valid count summary values from first cube in this set.

3.3 Partition Objects

3.3.1 CubePartition

class cr.cube.cubepart.**CubePartition**(*cube*, *transforms=None*)
A slice, a strand, or a nub drawn from a cube-response.

These represent 2, 1, or 0 dimensions of a cube, respectively.

cube_index
Offset of this partition's cube in its CubeSet.

Used to differentiate certain partitions like a filtered rows-summary strand.

dimension_types
Sequence of member of *cr.cube.enum.DIMENSION_TYPE* for each dimension.

Items appear in rows-dimension, columns-dimension order.

classmethod factory(*cube*, *slice_idx=0*, *transforms=None*, *population=None*, *ca_as_0th=None*, *mask_size=0*)
Return slice, strand, or nub object appropriate to passed parameters.

ndim
int count of dimensions for this partition.

population_fraction
population fraction of the cube

selected_category_labels
Tuple of str: names of any and all underlying categories in 'Selected'.

shape
Tuple of int vector counts for this partition.

Not to be confused with *numpy.ndarray.shape*, this represent the count of rows and columns respectively, in this partition. It does not necessarily represent the shape of any underlying *numpy.ndarray* object that may arise in the implementation of the cube partition. In particular, the value of any count in the shape can be zero.

A _Slice has a shape like (2, 3) representing (row-count, col-count). A _Strand has a shape like (5,) which represents its row-count. The shape of a _Nub is unconditionally () (an empty tuple).

variable_name
str representing the name of the superheading variable.

3.3.2 Slice

class cr.cube.cubepart.**_Slice**(*cube*, *slice_idx*, *transforms*, *population*, *mask_size*)
2D cube partition.

A slice represents the cross-tabulation of two dimensions, often, but not necessarily contributed by two different variables. A single CA variable has two dimensions which can be crosstabbed in a slice.

column_aliases
1D str ndarray of alias for each column, for use as column headings.

column_codes
1D int ndarray of code for each column, for use as column headings.

column_index
2D np.float64 ndarray of column-index "percentage".

The index values represent the difference of the percentages to the corresponding baseline values. The baseline values are the univariate percentages of the rows variable.

column_labels
1D str ndarray of name for each column, for use as column headings.

column_proportion_variances
2D ndarray of np.float64 column-proportion variance for each matrix cell.

column_proportions
2D np.float64 ndarray of column-proportion for each matrix cell.

This is the proportion of the weighted-N (aka. weighted base) of its column that the *weighted-count* in each cell represents, generally a number between 0.0 and 1.0. Note that within an inserted subtotal vector involving differences, the values can range between -1.0 and 1.0.

column_proportions_moe
1D/2D np.float64 ndarray of margin-of-error (MoE) for columns proportions.

The values are represented as fractions, analogue to the *column_proportions* property. This means that the value of 3.5% will have the value 0.035. The values can be np.nan when the corresponding percentage is also np.nan, which happens when the respective columns margin is 0.

column_share_sum
2D optional np.float64 ndarray of column share sum value for each table cell.

Raises *ValueError* if the cube-result does not include a sum cube-measure.

Column share of sum is the sum of each subvar item divided by the TOTAL number of column items.

column_std_dev
standard deviation for column percentages

std_deviation = sqrt(variance)

column_std_err
standard error for column percentages

std_error = sqrt(variance/N)

column_unweighted_bases
2D np.float64 ndarray of unweighted col-proportion denominator per cell.

column_weighted_bases
2D np.float64 ndarray of column-proportion denominator for each cell.

columns_base
1D/2D np.float64 ndarray of unweighted-N for each column/cell of slice.

This array is 2D (a distinct base for each cell) when the rows dimension is MR, because each MR-subvariable has its own unweighted N. This is because not every possible response is necessarily offered to every respondent.

In all other cases, the array is 1D, containing one value for each column.

columns_dimension_description
str description assigned to columns-dimension.

columns_dimension_name
str name assigned to columns-dimension.

Reflects the resolved dimension-name transform cascade.

columns_dimension_type
Member of *cr.cube.enum.DIMENSION_TYPE* describing columns dimension.

columns_margin
1D or 2D np.float64 ndarray of weighted-N for each column of slice.

This array is 2D (a distinct margin value for each cell) when the rows dimension is MR, because each MR-subvariable has its own weighted N. This is because not every possible response is necessarily offered to every respondent.

In all other cases, the array is 1D, containing one value for each column.

columns_margin_proportion
1D or 2D np.float64 ndarray of weighted-proportion for each column of slice.

This array is 2D (a distinct margin value for each cell) when the rows dimension is MR, because each MR-subvariable has its own weighted N. This is because not every possible response is necessarily offered to every respondent.

In all other cases, the array is 1D, containing one value for each column.

columns_scale_mean
Optional 1D np.float64 ndarray of scale mean for each column.

The returned vector is to be interpreted as a summary *row*. Also note that the underlying scale values are based on the numeric values of the opposing *rows-dimension* elements.

This value is *None* if no row element has an assigned numeric value.

columns_scale_mean_margin
Optional float overall mean of column-scale values.

This value is the "margin" of the *.columns_scale_mean* vector and might typically appear in the cell immediately to the right of the *.columns_scale_mean* summary-row. It is similar to a "table-total" value, in that it is a scalar that might appear in the lower right-hand corner of a table, but note that it does *not* represent the overall table in that *.rows_scale_mean_margin* will not have the same value (except by chance). This value derives from the numeric values of the row elements whereas its counterpart *.rows_scale_mean_margin* derives from the numeric values of the column elements.

This value is *None* if no row has an assigned numeric-value.

columns_scale_mean_pairwise_indices
Sequence of column-idx tuples indicating pairwise-t result of scale-means.

The sequence contains one tuple for each column. The indicies in a column's tuple each identify *another* of the columns who's scale-mean is pairwise-significant to that of the tuple's column. Pairwise significance is computed based on the more restrictive (lesser-value) threshold specified in the analysis.

columns_scale_mean_pairwise_indices_alt
Optional sequence of column-idx tuples indicating pairwise-t of scale-means.

This value is *None* if no secondary threshold value (alpha) was specified in the analysis. Otherwise, it is the same calculation as *.columns_scale_mean_pairwise_indices* computed using the less restrictive (greater-valued) threshold.

columns_scale_mean_stddev
Optional 1D np.float64 ndarray of scale-mean std-deviation for each column.

The returned vector (1D array) is to be interpreted as a summary *row*. Also note that the underlying scale values are based on the numeric values of the opposing *rows-dimension* elements.

This value is *None* if no row element has been assigned a numeric value.

columns_scale_mean_stderr
Optional 1D np.float64 ndarray of scale-mean standard-error for each row.

The returned vector is to be interpreted as a summary *row*. Also note that the underlying scale values are based on the numeric values of the opposing *rows-dimension* elements.

This value is *None* if no row element has a numeric value assigned or if the columns-weighted-base is *None* (eg an array variable in the row dim).

columns_scale_median
Optional 1D np.float64 ndarray of scale median for each column.

The returned vector is to be interpreted as a summary *row*. Also note that the underlying scale values are based on the numeric values of the opposing *rows-dimension* elements.

This value is *None* if no row element has been assigned a numeric value.

columns_scale_median_margin
Optional scalar numeric median of all column-scale values.

This value is the "margin" of the *.columns_scale_median* vector and might typically appear in the cell immediately to the right of the *.columns_scale_median* summary-row. It is similar to a "table-total" value, in that it is a scalar that might appear in the lower right-hand corner of a table, but note that it does *not* represent the overall table in that *.rows_scale_median_margin* will not have the same value (except by chance). This value derives from the numeric values of the row elements whereas its counterpart *.rows_scale_median_margin* derives from the numeric values of the column elements.

This value is *None* if no row has an assigned numeric-value.

counts
2D np.float64 ndarray of weighted cube counts.

derived_column_idxs
tuple of int index of each derived column-element in slice.

An element is derived if it's a subvariable of a multiple response dimension, which has been produced by the zz9, and inserted into the response data.

All other elements, including regular MR and CA subvariables, as well as categories of CAT dimensions, are not derived. Subtotals are also not derived in this sense, because they're not even part of the data (elements).

derived_row_idxs
tuple of int index of each derived row-element in slice.

An element is derived if it's a subvariable of a multiple response dimension, which has been produced by the zz9, and inserted into the response data.

All other elements, including regular MR and CA subvariables, as well as categories of CAT dimensions, are not derived. Subtotals are also not derived in this sense, because they're not even part of the data (elements).

description
str description of this slice, which it takes from its rows-dimension.

diff_column_idxs
tuple of int index of each difference column-element in slice.

diff_row_idxs
tuple of int index of each difference row-element in slice.

has_scale_means
True if the slice has valid columns scale mean.

inserted_column_idxs
tuple of int index of each subtotal column in slice.

inserted_row_idxs
tuple of int index of each subtotal row in slice.

means
2D optional np.float64 ndarray of mean value for each table cell.

Cell value is *np.nan* for each cell corresponding to an inserted subtotal (mean of addend cells cannot simply be added to get the mean of the subtotal).

Raises *ValueError* if the cube-result does not include a means cube-measure.

name
str name assigned to this slice.

A slice takes the name of its rows-dimension.

pairwise_indices
2D ndarray of tuple of int column-idxs meeting pairwise-t threshold.

Like:

```
[
   [(1, 3, 4), (), (0,), (), ()],
   [(2,), (1, 2), (), (), (0, 3)],
   [(), (), (), (), ()],
]
```

Has the same shape as *.counts*. Each int represents the offset of another column in the same row with a confidence interval meeting the threshold defined for this analysis.

pairwise_indices_alt
2D ndarray of tuple of int column-idxs meeting alternate threshold.

This value is None if no alternate threshold has been defined.

pairwise_means_indices
Optional 2D ndarray of tuple column-idxs significance threshold for mean.

Like:

```
[
   [(1, 3, 4), (), (0,), (), ()],
   [(2,), (1, 2), (), (), (0, 3)],
   [(), (), (), (), ()],
]
```

Has the same shape as *.means*. Each int represents the offset of another column in the same row with a confidence interval meeting the threshold defined for this analysis.

pairwise_means_indices_alt
2D ndarray of tuple of column-idxs meeting alternate threshold for mean.

This value is None if no alternate threshold has been defined.

pairwise_significance_means_p_vals(*column_idx*)
Optional 2D ndarray of means significance p-vals matrices for column idx.

pairwise_significance_means_t_stats(*column_idx*)
Optional 2D ndarray of means significance t-stats matrices for column idx.

pairwise_significance_p_vals(*column_idx*)
2D ndarray of pairwise-significance p-vals matrices for column idx.

pairwise_significance_t_stats(*column_idx*)
return 2D ndarray of pairwise-significance t-stats for selected column.

pairwise_significance_tests
tuple of _ColumnPairwiseSignificance tests.

Result has as many elements as there are columns in the slice. Each significance test contains *p_vals* and *t_stats* (ndarrays that represent probability values and statistical scores).

payload_order
1D np.int64 ndarray of signed int idx respecting the payload order.

Positive integers indicate the 1-indexed position in payload of regular elements, while negative integers are the subtotal insertions.

Needed for reordering color palette in exporter.

population_counts
2D np.float64 ndarray of population counts per cell.

The (estimated) population count is computed based on the *population* value provided when the Slice is created (*._population*). It is also adjusted to account for any filters that were applied as part of the query (*._cube.population_fraction*).

._population and *_cube.population_fraction* are both scalars and so do not affect sort order.

population_counts_moe
2D np.float64 ndarray of population-count margin-of-error (MoE) per cell.

The values are represented as population estimates, analogue to the *population_counts* property. This means that the values will be presented by actual estimated counts of the population. The values can be np.nan when the corresponding percentage is also np.nan, which happens when the respective margin is 0.

When calculating the estimates of categorical dates, the total populatioin is not "divided" between its categories, but rather considered constant for all categorical dates (or waves). Hence, the different standard errors will be applied in these specific cases (like the *row_std_err* or *column_std_err*). If categorical dates are not involved, the standard *table_std_err* is used.

population_proportions
2D np.float64 ndarray of proportions

The proportion used to calculate proportion counts depends on the dimension types.

population_std_err
2D np.float64 ndarray of standard errors

The proportion used to calculate proportion counts depends on the dimension types.

pvals
2D optional np.float64 ndarray of p-value for each cell.

A p-value is a measure of the probability that an observed difference could have occurred just by random chance. The lower the p-value, the greater the statistical significance of the observed difference.

A cell value of np.nan indicates a meaningful p-value could not be computed for that cell.

pvalues
2D optional np.float64 ndarray of p-value for each cell.

A p-value is a measure of the probability that an observed difference could have occurred just by random chance. The lower the p-value, the greater the statistical significance of the observed difference.

A cell value of np.nan indicates a meaningful p-value could not be computed for that cell.

residual_test_stats
Exposes pvals and zscores (with HS) stacked together

Public method used as cube_method for the SOA API

row_aliases
1D str ndarray of row alias for each matrix row.

These are suitable for use as row headings; alias for subtotal rows appear in the sequence and alias are ordered to correspond with their respective data row.

row_codes
1D int ndarray of row codes for each matrix row.

These are suitable for use as row headings; codes for subtotal rows appear in the sequence and codes are ordered to correspond with their respective data row.

row_labels
1D str ndarray of row name for each matrix row.

These are suitable for use as row headings; labels for subtotal rows appear in the sequence and labels are ordered to correspond with their respective data row.

row_order (*format=<ORDER_FORMAT.SIGNED_INDEXES: 0>*)
1D np.int64 ndarray of idx for each assembled row of matrix.

If order format is *SIGNED_INDEXES* negative values represent inserted subtotal-row locations; for *BOGUS_IDS* insertios are represented by *ins_{insertion_id}* string.

Indices appear in the order rows are to appear in the final result.

Needed for reordering color palette in exporter.

row_proportion_variances
2D ndarray of np.float64 row-proportion variance for each matrix cell.

row_proportions
2D np.float64 ndarray of row-proportion for each matrix cell.

This is the proportion of the weighted-N (aka. weighted base) of its row that the *weighted-count* in each cell represents, generally a number between 0.0 and 1.0. Note that within an inserted subtotal vector involving differences, the values can range between -1.0 and 1.0.

row_proportions_moe
2D np.float64 ndarray of margin-of-error (MoE) for rows proportions.

The values are represented as percentage-fractions, analogue to the *row_proportions* property. This means that the value of 3.5% will have the value 0.035. The values can be np.nan when the corresponding percentage is also np.nan, which happens when the respective table margin is 0.

row_share_sum
2D optional np.float64 ndarray of row share sum value for each table cell.

Raises *ValueError* if the cube-result does not include a sum cube-measure.

Row share of sum is the sum of each subvar item divided by the TOTAL number of row items.

row_std_dev
2D np.float64 ndarray of standard deviation for row percentages.

row_std_err
2D np.float64 ndarray of standard errors for row percentages.

row_unweighted_bases
2D np.float64 ndarray of unweighted row-proportion denominator per cell.

row_weighted_bases
2D np.float64 ndarray of row-proportion denominator for each table cell.

rows_base
1D/2D np.float64 ndarray of unweighted-N for each row/cell of slice.

This array is 2D (a distinct base for each cell) when the columns dimension is MR, because each MR-subvariable has its own unweighted N. This is because not every possible response is necessarily offered to every respondent.

In all other cases, the array is 1D, containing one value for each column.

rows_dimension_alias
str alias assigned to rows-dimension.

rows_dimension_description
str description assigned to rows-dimension.

Reflects the resolved dimension-description transform cascade.

rows_dimension_fills
tuple of optional RGB str like "#def032" fill color for each row in slice.

The values reflect the resolved element-fill transform cascade. The length and ordering of the sequence correspond to the rows in the slice, including accounting for insertions and hidden rows. A value of *None* indicates the default fill, possibly determined by a theme or template.

rows_dimension_name
str name assigned to rows-dimension.

Reflects the resolved dimension-name transform cascade.

rows_dimension_type
Member of *cr.cube.enum.DIMENSION_TYPE* specifying type of rows dimension.

rows_margin
1D or 2D np.float64 ndarray of weighted-N for each column of slice.

This array is 2D (a distinct margin value for each cell) when the columns dimension is MR, because each MR-subvariable has its own weighted N. This is because not every possible response is necessarily offered to every respondent.

In all other cases, the array is 1D, containing one value for each column.

rows_margin_proportion
1D or 2D np.float64 ndarray of weighted-proportion for each column of slice.

This array is 2D (a distinct margin value for each cell) when the columns dimension is MR, because each MR-subvariable has its own weighted N. This is because not every possible response is necessarily offered to every respondent.

In all other cases, the array is 1D, containing one value for each column.

rows_scale_mean
Optional 1D np.float64 ndarray of scale mean for each row.

The returned vector is to be interpreted as a summary *column*. Also note that the underlying scale values are based on the numeric values of the opposing *columns-dimension* elements.

This value is *None* if no column element has an assigned numeric value.

rows_scale_mean_margin
Optional float overall mean of row-scale values.

This value is the "margin" of the *.rows_scale_mean* vector and might typically appear in the cell immediately below the *.rows_scale_mean* summary-column. It is similar to a "table-total" value, in that it is a scalar that might appear in the lower right-hand corner of a table, but note that it does *not* represent the overall table in that *.columns_scale_mean_margin* will not have the same value (except by chance). This value derives from the numeric values of the column elements whereas its counterpart *.columns_scale_mean_margin* derives from the numeric values of the row elements.

This value is *None* if no column has an assigned numeric-value.

rows_scale_mean_stddev
Optional 1D np.float64 ndarray of std-deviation of scale-mean for each row.

The returned vector (1D array) is to be interpreted as a summary *column*. Also note that the underlying scale values are based on the numeric values of the opposing *columns-dimension* elements.

This value is *None* if no column elements have an assigned numeric value.

rows_scale_mean_stderr
Optional 1D np.float64 ndarray of standard-error of scale-mean for each row.

The returned vector is to be interpreted as a summary *column*. Also note that the underlying scale values are based on the numeric values of the opposing *columns-dimension* elements.

This value is *None* if no column element has a numeric value assigned or if the rows-weighted-base is *None* (eg an array variable in the column dim).

rows_scale_median
Optional 1D np.float64 ndarray of scale median for each row.

The returned vector is to be interpreted as a summary *column*. Also note that the underlying scale values are based on the numeric values of the opposing *columns-dimension* elements.

This value is *None* if no column element has an assigned numeric value.

rows_scale_median_margin
Optional scalar numeric median of all row-scale values.

This value is the "margin" of the *.rows_scale_median* vector and might typically appear in the cell immediately below the *.rows_scale_median* summary-column. It is similar to a "table-total" value, in that it is a scalar that might appear in the lower right-hand corner of a table, but note that it does *not* represent the overall table in that *.columns_scale_mean_margin* will not have the same value (except by chance). This value derives from the numeric values of the column elements whereas its counterpart *.columns_scale_median_margin* derives from the numeric values of the row elements.

This value is *None* if no column has an assigned numeric-value.

smoothed_column_index
2D np.float64 ndarray of smoothed column-index "percentage".

If cube has smoothing specification in the transforms it will return the column index smoothed according to the algorithm and the parameters specified, otherwise it fallbacks to unsmoothed values.

smoothed_column_percentages
2D np.float64 ndarray of smoothed column-percentages for each matrix cell.

If cube has smoothing specification in the transforms it will return the column percentages smoothed according to the algorithm and the parameters specified, otherwise it fallbacks to unsmoothed values.

smoothed_column_proportions
2D np.float64 ndarray of smoothed column-proportion for each matrix cell.

This is the proportion of the weighted-count for cell to the weighted-N of the column the cell appears in (aka. column-margin). Generally a number between 0.0 and 1.0 inclusive, but subtotal differences can be between -1.0 and 1.0 inclusive.

If cube has smoothing specification in the transforms it will return the column proportions smoothed according to the algorithm and the parameters specified, otherwise it fallbacks to unsmoothed values.

smoothed_columns_scale_mean
Optional 1D np.float64 ndarray of smoothed scale mean for each column.

If cube has smoothing specification in the transforms it will return the column scale mean smoothed according to the algorithm and the parameters specified, otherwise it fallbacks to unsmoothed values.

smoothed_means
2D optional np.float64 ndarray of smoothed mean value for each table cell.

If cube has smoothing specification in the transforms it will return the smoothed means according to the algorithm and the parameters specified, otherwise it fallbacks to unsmoothed values.

stddev
2D optional np.float64 ndarray of stddev value for each table cell.

Raises *ValueError* if the cube-result does not include a stddev cube-measure.

sums
2D optional np.float64 ndarray of sum value for each table cell.

Raises *ValueError* if the cube-result does not include a sum cube-measure.

tab_alias
Subvar alias of slice id if first dimension is a CA_SUBVAR, '"' otherwise.

tab_label
Subvar label of slice id if first dimension is a CA_SUBVAR, '"' otherwise.

table_base
Scalar or 1D/2D np.float64 ndarray of unweighted-N for table.

This value is scalar when the slice has no MR dimensions, 1D when the slice has one MR dimension (either MR_X or X_MR), and 2D for an MR_X_MR slice.

The caller must know the dimensionality of the slice in order to correctly interpret a 1D value for this property.

This value has four distinct forms, depending on the slice dimensions:

- ARR_X_ARR - 2D ndarray with a distinct table-base value per cell.
- ARR_X - 1D ndarray of value per *row* when only rows dimension is ARR.
- X_ARR - 1D ndarray of value per *column* when only col dimension is ARR
- CAT_X_CAT - scalar float value when slice has no MR dimension.

table_base_range
[min, max] np.float64 ndarray range of the table_base (table-unweighted-base)

A CAT_X_CAT has a scalar for all table-unweighted-bases, but arrays have more than one table-weighted-base. This collapses all the values them to the range, and it is "unpruned", meaning that it is calculated before any hiding or removing of empty rows/columns.

table_margin
Scalar or 1D/2D np.float64 ndarray of weighted-N table.

This value is scalar when the slice has no MR dimensions, 1D when the slice has one MR dimension (either MR_X or X_MR), and 2D for an MR_X_MR slice.

The caller must know the dimensionality of the slice in order to correctly interpret a 1D value for this property.

This value has four distinct forms, depending on the slice dimensions:

- CAT_X_CAT - scalar float value when slice has no ARRAY dimension.
- ARRAY_X - 1D ndarray of value per *row* when only rows dimension is ARRAY.
- X_ARRAY - 1D ndarray of value per *column* when only column is ARRAY.
- ARRAY_X_ARRAY - 2D ndarray with a distinct table-margin value per cell.

table_margin_range
[min, max] np.float64 ndarray range of the table_margin (table-weighted-base)

A CAT_X_CAT has a scalar for all table-weighted-bases, but arrays have more than one table-weighted-base. This collapses all of the values to a range, and it is "unpruned", meaning that it is calculated before any hiding or removing of empty rows/columns.

table_name
Optional table name for this Slice

Provides differentiated name for each stacked table of a 3D cube.

table_proportion_variances
2D ndarray of np.float64 table-proportion variance for each matrix cell.

table_proportions
2D ndarray of np.float64 fraction of table count each cell contributes.

This is the proportion of the weighted-count for cell to the weighted-N of the row the cell appears in (aka. table-margin). Generally a number between 0.0 and 1.0 inclusive, but subtotal differences can be between -1.0 and 1.0 inclusive.

table_proportions_moe
1D/2D np.float64 ndarray of margin-of-error (MoE) for table proportions.

The values are represented as fractions, analogue to the *table_proportions* property. This means that the value of 3.5% will have the value 0.035. The values can be np.nan when the corresponding percentage is also np.nan, which happens when the respective table margin is 0.

table_std_dev
2D np.float64 ndarray of std-dev of table-percent for each table cell.

table_std_err
2D optional np.float64 ndarray of std-error of table-percent for each cell.

A cell value can be np.nan under certain conditions.

table_unweighted_bases
2D np.float64 ndarray of unweighted table-proportion denominator per cell.

table_weighted_bases
2D np.float64 ndarray of table-proportion denominator for each cell.

total_share_sum
2D optional np.float64 ndarray of total share sum value for each table cell.

Raises *ValueError* if the cube-result does not include a sum cube-measure.

Total share of sum is the sum of each subvar item divided by the TOTAL of items.

unweighted_counts
2D np.float64 ndarray of unweighted count for each slice matrix cell.

weighted_counts
2D np.float64 ndarray of weighted cube counts.

zscores
2D np.float64 ndarray of std-res value for each cell of matrix.

A z-score is also known as a *standard score* and is the number of standard deviations above (positive) or below (negative) the population mean a cell's value is.

3.3.3 Strand

class cr.cube.cubepart.**_Strand**(*cube*, *transforms*, *population*, *ca_as_0th*, *slice_idx*, *mask_size*)
1D cube-partition.

A strand can arise from a 1D cube (non-CA univariate), or as a partition of a CA-cube (CAs are 2D) into a sequence of 1D partitions, one for each subvariable.

counts
1D np.float64 ndarray of weighted count for each row of strand.

The values are int when the underlying cube-result has no weighting.

derived_row_idxs
tuple of int index of each derived row-element in this strand.

Subtotals cannot be derived

An element is derived if it's a subvariable of a multiple response dimension, which has been produced by the zz9, and inserted into the response data.

All other elements, including regular MR and CA subvariables, as well as categories of CAT dimensions, are not derived. Subtotals are also not derived in this sense, because they're not even part of the data (elements).

diff_row_idxs
tuple of int index of each difference row-element in this strand.

Valid elements are cannot be differences, only some subtotals can.

has_scale_means
True if the strand has valid scale means.

inserted_row_idxs
tuple of int index of each inserted row in this strand.

Suitable for use in applying different formatting (e.g. Bold) to inserted rows. Provided index values correspond to measure values as-delivered by this strand, after any insertion of subtotals, re-ordering, and hiding/pruning of rows specified in a transform has been applied.

Provided index values correspond rows after any insertion of subtotals, re-ordering, and hiding/pruning.

means
1D np.float64 ndarray of mean for each row of strand.

Raises ValueError when accessed on a cube-result that does not contain a means cube-measure.

min_base_size_mask
1D bool ndarray of True for each row that fails to meet min-base spec.

The "base" is the physical (unweighted) count of respondents to the question. When this is lower than a specified threshold, the reliability of the value is too low to be meaningful. The threshold is defined by the caller (user).

payload_order
1D np.int64 ndarray of signed int idx respecting the payload order.

Positive integers indicate the 1-indexed position in payload of regular elements, while negative integers are the subtotal insertions.

Needed for reordering color palette in exporter.

population_counts
1D np.float64 ndarray of population count for each row of strand.

The (estimated) population count is computed based on the *population* value provided when the Strand is created. It is also adjusted to account for any filters that were applied as part of the query.

population_counts_moe
1D np.float64 ndarray of population margin-of-error (MoE) for table percents.

The values are represented as population estimates, analogue to the *population_counts* property. This means that the values will be presented by actual estimated counts of the population The values can be np.nan when the corresponding percentage is also np.nan, which happens when the respective table margin is 0.

population_proportion_stderrs
1D np.float64 population-proportion-standard-error for each row

Generally equal to the table_proprotion_standard_error, but because we don't divide the population when the row is a CAT_DATE, can also be all 0s. Used to calculate the population_counts_moe.

population_proportions
1D np.float64 population-proportion for each row

Generally equal to the table_proprotions, but because we don't divide the population when the row is a CAT_DATE, can also be all 1s. Used to calculate the population_counts.

row_aliases
1D str ndarray of alias for each row, for use as row headings.

row_codes
1D int ndarray of code for each row, for use as row headings.

row_count
int count of rows in a returned measure or marginal.

This count includes inserted rows but not rows that have been hidden/pruned.

row_labels
1D str ndarray of name for each row, suitable for use as row headings.

row_order (*format=<ORDER_FORMAT.SIGNED_INDEXES: 0>*)
1D np.int64 ndarray of idx for each assembled row of stripe.

If order format is *SIGNED_INDEXES* negative values represent inserted subtotal-row locations; for *BOGUS_IDS* insertios are represented by *ins_{insertion_id}* string. Indices appear in the order rows are to appear in the final result.

Needed for reordering color palette in exporter.

rows_base
1D np.float64 ndarray of unweighted-N for each row of slice.

rows_dimension_alias
str alias assigned to rows-dimension.

rows_dimension_description
str description assigned to rows-dimension.

Reflects the resolved dimension-description transform cascade.

rows_dimension_fills
tuple of optional RGB str like "#def032" fill color for each strand row.

Each value reflects the resolved element-fill transform cascade. The length and ordering of the sequence correspond to the rows in the slice, including accounting for insertions, ordering, and hidden rows. A fill value is *None* when no explicit fill color is defined for that row, indicating the default fill color for that row should be used, probably coming from a caller-defined theme.

rows_dimension_name
str name assigned to rows-dimension.

Reflects the resolved dimension-name transform cascade.

rows_dimension_type
Member of DIMENSION_TYPE enum describing type of rows dimension.

rows_margin
1D np.float64 ndarray of weighted-N for each row of slice.

scale_mean
Optional float mean of row numeric-values (scale).

This value is *None* when no row-elements have a numeric-value assigned. The numeric value (aka. "scale") for a row is its count multiplied by the numeric-value of its element. For example, if 100 women responded "Very Likely" and the numeric-value of the "Very Likely" response (element) was 4, then the scale for that row would be 400. The scale mean is the average of those scale values over the total count of responses.

scale_median
Optional int/float median of scaled weighted-counts.

This value is *None* when no rows have a numeric-value assigned.

scale_std_dev
Optional np.float64 standard-deviation of scaled weighted counts.

This value is *None* when no rows have a numeric-value assigned.

scale_std_err
Optional np.float64 standard-error of scaled weighted counts.

This value is *None* when no rows have a numeric-value assigned. The value has the same units as the assigned numeric values and indicates the dispersion of the scaled-count distribution from its mean (scale-mean).

scale_stddev
Optional np.float64 standard-deviation of scaled weighted counts.

This value is *None* when no rows have a numeric-value assigned.

scale_stderr
Optional np.float64 standard-error of scaled weighted counts.

This value is *None* when no rows have a numeric-value assigned. The value has the same units as the assigned numeric values and indicates the dispersion of the scaled-count distribution from its mean (scale-mean).

shape
Tuple of int vector counts for this partition.

A _Strand has a shape like (5,) which represents its row-count.

Not to be confused with *numpy.ndarray.shape*, this represent the count of rows in this strand. It does not necessarily represent the shape of any underlying *numpy.ndarray* object In particular, the value of its row-count can be zero.

share_sum
1D np.float64 ndarray of share of sum for each row of strand.

Raises *ValueError* if the cube-result does not include a sum cube-measure.

Share of sum is the sum of each subvar item divided by the TOTAL number of items.

smoothed_means
1D np.float64 ndarray of smoothed mean for each row of strand.

If cube has smoothing specification in the transforms it will return the smoothed means according to the algorithm and the parameters specified, otherwise it fallbacks to unsmoothed values.

stddev
1D np.float64 ndarray of stddev for each row of strand.

Raises ValueError when accessed on a cube-result that does not contain a stddev cube-measure.

sums
1D np.float64 ndarray of sum for each row of strand.

Raises ValueError when accessed on a cube-result that does not contain a sum cube-measure.

tab_alias
Subvar alias of strand if first dimension is a CA_SUBVAR, "''" otherwise.

tab_label
Subvar label of strand if first dimension is a CA_SUBVAR, "''" otherwise.

table_base_range
[min, max] np.float64 ndarray range of unweighted-N for this stripe.

A non-MR stripe will have a single base, represented by min and max being the same value. Each row of an MR stripe has a distinct base, which is reduced to a range in that case.

table_margin_range
[min, max] np.float64 ndarray range of (total) weighted-N for this stripe.

A non-MR stripe will have a single margin, represented by min and max being the same value. Each row of an MR stripe has a distinct base, which is reduced to a range in that case.

table_name
Optional table name for this strand

Only for CA-as-0th case, provides differentiated names for stacked tables.

table_percentages
1D np.float64 ndarray of table-percentage for each row.

Table-percentage is the fraction of the table weighted-N contributed by each row, expressed as a percentage (float between 0.0 and 100.0 inclusive).

table_proportion_moes
1D np.float64 ndarray of table-proportion margin-of-error (MoE) for each row.

The values are represented as fractions, analogue to the *table_proportions* property. This means that the value of 3.5% will have the value 0.035. The values can be np.nan when the corresponding proportion is also np.nan, which happens when the respective columns margin is 0.

table_proportion_stddevs
1D np.float64 ndarray of table-proportion std-deviation for each row.

table_proportion_stderrs
1D np.float64 ndarray of table-proportion std-error for each row.

table_proportions
1D np.float64 ndarray of fraction of weighted-N contributed by each row.

The proportion is expressed as a float between 0.0 and 1.0 inclusive.

title
The str display name of this strand, suitable for use as a column heading.

Strand.name is the rows-dimension name, which is suitable for use as a title of the row-headings. However, a strand can also appear as a *column* and this value is a suitable name for such a column.

unweighted_bases
1D np.float64 ndarray of base count for each row, before weighting.

When the rows dimension is multiple-response (MR), each value is different, reflecting the base for that individual subvariable. In all other cases, the table base is repeated for each row.

unweighted_counts
1D np.float64 ndarray of unweighted count for each row of stripe.

weighted_bases
1D np.float64 ndarray of table-proportion denominator for each row.

For a non-MR strand, all values in the array are the same. For an MR strand, each value may be different, reflecting the fact that not all response options were necessarily presented to all respondents.

weighted_counts
1D np.float64 ndarray of weighted count for each row of strand.

The values are int when the underlying cube-result has no weighting.

3.3.4 Nub

class `cr.cube.cubepart.`**_Nub**(*cube*, *transforms=None*)
0D slice.

is_empty
True if the partition has no counts, False otherwise

means
Float scalar representing the mean.

table_base
Int scalar of the unweighted N of the table.

unweighted_count
Integer scalar of total unweighted count of the table

3.4 Dimension objects

class `cr.cube.dimension.`**Dimension**(*dimension_dict*, *dimension_type*, *dimension_transforms=None*)
Represents one dimension of a cube response.

Each dimension represents one of the variables in a cube response. For example, a query to cross-tabulate snack-food preference against region will have two variables (snack-food preference and region) and will produce a two-dimensional (2D) cube response. That cube will have two of these dimension objects, which are accessed using `CrunchCube.dimensions`.

alias
Return the alias for the dimension if it exists, None otherwise.

all_elements
Elements object providing cats or subvars of this dimension.

Elements in this sequence appear in cube-result order.

apply_transforms(*dimension_transforms*) → cr.cube.dimension.Dimension
Return a new *Dimension* object with *dimension_transforms* applied.

The new dimension object is the same as this one in all other respects.

description
str description of this dimension.

element_aliases
tuple of string element-aliases for each valid element in this dimension.

Element-aliases appear in the order defined in the cube-result.

element_ids
tuple of int element-id for each valid element in this dimension.

Element-ids appear in the order defined in the cube-result.

element_labels
tuple of string element-labels for each valid element in this dimension.

Element-labels appear in the order defined in the cube-result.

hidden_idxs
tuple of int element-idx for each hidden valid element in this dimension.

An element is hidden when a "hide" transform is applied to it in its transforms dict.

insertion_ids
tuple of int insertion-id for each insertion in this dimension.

Insertion-ids appear in the order insertions are defined in the dimension.

name
str name of this dimension, the empty string ("") if not specified.

numeric_values
tuple of numeric values for valid elements of this dimension.

Each category of a categorical variable can be assigned a *numeric value*. For example, one might assign *like=1, dislike=-1, neutral=0*. These numeric mappings allow quantitative operations (such as mean) to be applied to what now forms a *scale* (in this example, a scale of preference).

The numeric values appear in the same order as the categories/elements of this dimension. Each element is represented by a value, but an element with no numeric value appears as *np.nan* in the returned list.

order_spec
_OrderSpec proxy object for dimension.transforms.order dict from payload.

prune
True if empty elements should be automatically hidden on this dimension.

selected_categories
List of selected categories specified for this dimension.

shape
int count of *all* elements in this dimension, both valid and missing.

smoothing_dict
Optional dict of smoothing specifications.

subtotal_aliases
tuple of string element-aliases for each subtotal in this dimension.

Element-aliases appear in the order defined in the cube-result.

subtotal_labels
tuple of string element-labels for each subtotal in this dimension.

Element-labels appear in the order defined in the cube-result.

subtotals
_Subtotals sequence object for this dimension.

Each item in the sequence is a _Subtotal object specifying a subtotal, including its addends and anchor.

subtotals_in_payload_order
_Subtotals sequence object for this dimension respecting the payload order.

Each item in the sequence is a _Subtotal object specifying a subtotal, including its addends and anchor.

translate_element_id(*_id*) → Optional[str]
Optional string that is the translation of various ids to subvariable alias

This is needed for the opposing dimension's sort by opposing element, because when creating a dimension, we don't have access to the other dimension's ids to transform it. Therefore, the id for opposing element sort by value transforms is not translated at creation time.

0) If dimension is not a subvariables dimension, return the _id.
1) If id matches an alias, then just use it.
2) If id matches a subvariable id, translate to corresponding alias.
3) If id matches an element id, translate to corresponding alias.
4) If id can be parsed to int and matches an element id, translate to alias.
5) If id is int (or can be parsed to int) and can be used as index (eg in range 0-# of elements), use _id'th alias.
6) If all of these fail, return None.

valid_elements
Elements object providing access to non-missing elements.

Any categories or subvariables representing missing data are excluded from the collection; this sequence represents a subset of that provided by *.all_elements*.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index

Symbols

_Nub (*class in cr.cube.cubepart*), 28
_Slice (*class in cr.cube.cubepart*), 13
_Strand (*class in cr.cube.cubepart*), 23

A

alias (*cr.cube.dimension.Dimension attribute*), 28
all_elements (*cr.cube.dimension.Dimension attribute*), 28
apply_transforms() (*cr.cube.dimension.Dimension method*), 28
available_measures (*cr.cube.cube.Cube attribute*), 9
available_measures (*cr.cube.cube.CubeSet attribute*), 11

C

can_show_pairwise (*cr.cube.cube.CubeSet attribute*), 11
column_aliases (*cr.cube.cubepart._Slice attribute*), 13
column_codes (*cr.cube.cubepart._Slice attribute*), 13
column_index (*cr.cube.cubepart._Slice attribute*), 13
column_labels (*cr.cube.cubepart._Slice attribute*), 13
column_proportion_variances (*cr.cube.cubepart._Slice attribute*), 13
column_proportions (*cr.cube.cubepart._Slice attribute*), 13
column_proportions_moe (*cr.cube.cubepart._Slice attribute*), 13
column_share_sum (*cr.cube.cubepart._Slice attribute*), 14
column_std_dev (*cr.cube.cubepart._Slice attribute*), 14
column_std_err (*cr.cube.cubepart._Slice attribute*), 14
column_unweighted_bases (*cr.cube.cubepart._Slice attribute*), 14
column_weighted_bases (*cr.cube.cubepart._Slice attribute*), 14
columns_base (*cr.cube.cubepart._Slice attribute*), 14
columns_dimension_description (*cr.cube.cubepart._Slice attribute*), 14
columns_dimension_name (*cr.cube.cubepart._Slice attribute*), 14
columns_dimension_type (*cr.cube.cubepart._Slice attribute*), 14
columns_margin (*cr.cube.cubepart._Slice attribute*), 14
columns_margin_proportion (*cr.cube.cubepart._Slice attribute*), 14
columns_scale_mean (*cr.cube.cubepart._Slice attribute*), 15
columns_scale_mean_margin (*cr.cube.cubepart._Slice attribute*), 15
columns_scale_mean_pairwise_indices (*cr.cube.cubepart._Slice attribute*), 15
columns_scale_mean_pairwise_indices_alt (*cr.cube.cubepart._Slice attribute*), 15
columns_scale_mean_stddev (*cr.cube.cubepart._Slice attribute*), 15
columns_scale_mean_stderr (*cr.cube.cubepart._Slice attribute*), 15
columns_scale_median (*cr.cube.cubepart._Slice attribute*), 15
columns_scale_median_margin (*cr.cube.cubepart._Slice attribute*), 15
counts (*cr.cube.cubepart._Slice attribute*), 16
counts (*cr.cube.cubepart._Strand attribute*), 24
counts_with_missings (*cr.cube.cube.Cube attribute*), 9
covariance (*cr.cube.cube.Cube attribute*), 9
Cube (*class in cr.cube.cube*), 9
cube_index (*cr.cube.cube.Cube attribute*), 9
cube_index (*cr.cube.cubepart.CubePartition attribute*), 12
CubePartition (*class in cr.cube.cubepart*), 12
CubeSet (*class in cr.cube.cube*), 11

D

derived_column_idxs (*cr.cube.cubepart._Slice attribute*), 16
derived_row_idxs (*cr.cube.cubepart._Slice attribute*), 16
derived_row_idxs (*cr.cube.cubepart._Strand attribute*), 24
description (*cr.cube.cube.Cube attribute*), 9
description (*cr.cube.cube.CubeSet attribute*), 11
description (*cr.cube.cubepart._Slice attribute*), 16
description (*cr.cube.dimension.Dimension attribute*), 29
diff_column_idxs (*cr.cube.cubepart._Slice attribute*), 16
diff_row_idxs (*cr.cube.cubepart._Slice attribute*), 16
diff_row_idxs (*cr.cube.cubepart._Strand attribute*), 24
Dimension (*class in cr.cube.dimension*), 28
dimension_types (*cr.cube.cube.Cube attribute*), 9
dimension_types (*cr.cube.cubepart.CubePartition attribute*), 12
dimensions (*cr.cube.cube.Cube attribute*), 9

E

element_aliases (*cr.cube.dimension.Dimension attribute*), 29
element_ids (*cr.cube.dimension.Dimension attribute*), 29
element_labels (*cr.cube.dimension.Dimension attribute*), 29

F

factory() (*cr.cube.cubepart.CubePartition class method*), 13

H

has_scale_means (*cr.cube.cubepart._Slice attribute*), 16
has_scale_means (*cr.cube.cubepart._Strand attribute*), 24
has_weighted_counts (*cr.cube.cube.Cube attribute*), 10
has_weighted_counts (*cr.cube.cube.CubeSet attribute*), 11
hidden_idxs (*cr.cube.dimension.Dimension attribute*), 29

I

inflate() (*cr.cube.cube.Cube method*), 10
inserted_column_idxs (*cr.cube.cubepart._Slice attribute*), 16
inserted_row_idxs (*cr.cube.cubepart._Slice attribute*), 16
inserted_row_idxs (*cr.cube.cubepart._Strand attribute*), 24
insertion_ids (*cr.cube.dimension.Dimension attribute*), 29
is_ca_as_0th (*cr.cube.cube.CubeSet attribute*), 11
is_empty (*cr.cube.cubepart._Nub attribute*), 28

M

means (*cr.cube.cube.Cube attribute*), 10
means (*cr.cube.cubepart._Nub attribute*), 28
means (*cr.cube.cubepart._Slice attribute*), 16
means (*cr.cube.cubepart._Strand attribute*), 24
min_base_size_mask (*cr.cube.cubepart._Strand attribute*), 24
missing (*cr.cube.cube.Cube attribute*), 10
missing_count (*cr.cube.cube.CubeSet attribute*), 12

N

n_responses (*cr.cube.cube.Cube attribute*), 10
n_responses (*cr.cube.cube.CubeSet attribute*), 12
name (*cr.cube.cube.Cube attribute*), 10
name (*cr.cube.cube.CubeSet attribute*), 12
name (*cr.cube.cubepart._Slice attribute*), 16
name (*cr.cube.dimension.Dimension attribute*), 29
ndim (*cr.cube.cube.Cube attribute*), 10
ndim (*cr.cube.cubepart.CubePartition attribute*), 13
numeric_values (*cr.cube.dimension.Dimension attribute*), 29

O

order_spec (*cr.cube.dimension.Dimension attribute*), 29
overlaps (*cr.cube.cube.Cube attribute*), 10

P

pairwise_indices (*cr.cube.cubepart._Slice attribute*), 16
pairwise_indices_alt (*cr.cube.cubepart._Slice attribute*), 17
pairwise_means_indices (*cr.cube.cubepart._Slice attribute*), 17
pairwise_means_indices_alt (*cr.cube.cubepart._Slice attribute*), 17
pairwise_significance_means_p_vals() (*cr.cube.cubepart._Slice method*), 17
pairwise_significance_means_t_stats() (*cr.cube.cubepart._Slice method*), 17
pairwise_significance_p_vals() (*cr.cube.cubepart._Slice method*), 17
pairwise_significance_t_stats() (*cr.cube.cubepart._Slice method*), 17
pairwise_significance_tests (*cr.cube.cubepart._Slice attribute*), 17

partition_sets (*cr.cube.cube.CubeSet attribute*), 12
partitions (*cr.cube.cube.Cube attribute*), 10
payload_order (*cr.cube.cubepart._Slice attribute*), 17
payload_order (*cr.cube.cubepart._Strand attribute*), 24
population_counts (*cr.cube.cubepart._Slice attribute*), 17
population_counts (*cr.cube.cubepart._Strand attribute*), 24
population_counts_moe (*cr.cube.cubepart._Slice attribute*), 18
population_counts_moe (*cr.cube.cubepart._Strand attribute*), 24
population_fraction (*cr.cube.cube.Cube attribute*), 10
population_fraction (*cr.cube.cube.CubeSet attribute*), 12
population_fraction (*cr.cube.cubepart.CubePartition attribute*), 13
population_proportion_stderrs (*cr.cube.cubepart._Strand attribute*), 25
population_proportions (*cr.cube.cubepart._Slice attribute*), 18
population_proportions (*cr.cube.cubepart._Strand attribute*), 25
population_std_err (*cr.cube.cubepart._Slice attribute*), 18
prune (*cr.cube.dimension.Dimension attribute*), 29
pvals (*cr.cube.cubepart._Slice attribute*), 18
pvalues (*cr.cube.cubepart._Slice attribute*), 18

R

residual_test_stats (*cr.cube.cubepart._Slice attribute*), 18
row_aliases (*cr.cube.cubepart._Slice attribute*), 18
row_aliases (*cr.cube.cubepart._Strand attribute*), 25
row_codes (*cr.cube.cubepart._Slice attribute*), 18
row_codes (*cr.cube.cubepart._Strand attribute*), 25
row_count (*cr.cube.cubepart._Strand attribute*), 25
row_labels (*cr.cube.cubepart._Slice attribute*), 18
row_labels (*cr.cube.cubepart._Strand attribute*), 25
row_order() (*cr.cube.cubepart._Slice method*), 19
row_order() (*cr.cube.cubepart._Strand method*), 25
row_proportion_variances (*cr.cube.cubepart._Slice attribute*), 19
row_proportions (*cr.cube.cubepart._Slice attribute*), 19
row_proportions_moe (*cr.cube.cubepart._Slice attribute*), 19
row_share_sum (*cr.cube.cubepart._Slice attribute*), 19
row_std_dev (*cr.cube.cubepart._Slice attribute*), 19
row_std_err (*cr.cube.cubepart._Slice attribute*), 19
row_unweighted_bases (*cr.cube.cubepart._Slice attribute*), 19
row_weighted_bases (*cr.cube.cubepart._Slice attribute*), 19
rows_base (*cr.cube.cubepart._Slice attribute*), 19
rows_base (*cr.cube.cubepart._Strand attribute*), 25
rows_dimension_alias (*cr.cube.cubepart._Slice attribute*), 19
rows_dimension_alias (*cr.cube.cubepart._Strand attribute*), 25
rows_dimension_description (*cr.cube.cubepart._Slice attribute*), 19
rows_dimension_description (*cr.cube.cubepart._Strand attribute*), 25
rows_dimension_fills (*cr.cube.cubepart._Slice attribute*), 20
rows_dimension_fills (*cr.cube.cubepart._Strand attribute*), 25
rows_dimension_name (*cr.cube.cubepart._Slice attribute*), 20
rows_dimension_name (*cr.cube.cubepart._Strand attribute*), 25
rows_dimension_type (*cr.cube.cubepart._Slice attribute*), 20
rows_dimension_type (*cr.cube.cubepart._Strand attribute*), 26
rows_margin (*cr.cube.cubepart._Slice attribute*), 20
rows_margin (*cr.cube.cubepart._Strand attribute*), 26
rows_margin_proportion (*cr.cube.cubepart._Slice attribute*), 20
rows_scale_mean (*cr.cube.cubepart._Slice attribute*), 20
rows_scale_mean_margin (*cr.cube.cubepart._Slice attribute*), 20
rows_scale_mean_stddev (*cr.cube.cubepart._Slice attribute*), 20
rows_scale_mean_stderr (*cr.cube.cubepart._Slice attribute*), 21
rows_scale_median (*cr.cube.cubepart._Slice attribute*), 21
rows_scale_median_margin (*cr.cube.cubepart._Slice attribute*), 21

S

scale_mean (*cr.cube.cubepart._Strand attribute*), 26
scale_median (*cr.cube.cubepart._Strand attribute*), 26
scale_std_dev (*cr.cube.cubepart._Strand attribute*), 26
scale_std_err (*cr.cube.cubepart._Strand attribute*), 26
scale_stddev (*cr.cube.cubepart._Strand attribute*), 26

scale_stderr (*cr.cube.cubepart._Strand attribute*), 26
selected_categories (*cr.cube.dimension.Dimension attribute*), 29
selected_category_labels (*cr.cube.cubepart.CubePartition attribute*), 13
shape (*cr.cube.cubepart._Strand attribute*), 26
shape (*cr.cube.cubepart.CubePartition attribute*), 13
shape (*cr.cube.dimension.Dimension attribute*), 29
share_sum (*cr.cube.cubepart._Strand attribute*), 26
smoothed_column_index (*cr.cube.cubepart._Slice attribute*), 21
smoothed_column_percentages (*cr.cube.cubepart._Slice attribute*), 21
smoothed_column_proportions (*cr.cube.cubepart._Slice attribute*), 21
smoothed_columns_scale_mean (*cr.cube.cubepart._Slice attribute*), 21
smoothed_means (*cr.cube.cubepart._Slice attribute*), 21
smoothed_means (*cr.cube.cubepart._Strand attribute*), 26
smoothing_dict (*cr.cube.dimension.Dimension attribute*), 29
stddev (*cr.cube.cube.Cube attribute*), 10
stddev (*cr.cube.cubepart._Slice attribute*), 21
stddev (*cr.cube.cubepart._Strand attribute*), 27
subtotal_aliases (*cr.cube.dimension.Dimension attribute*), 29
subtotal_labels (*cr.cube.dimension.Dimension attribute*), 29
subtotals (*cr.cube.dimension.Dimension attribute*), 29
subtotals_in_payload_order (*cr.cube.dimension.Dimension attribute*), 30
sums (*cr.cube.cube.Cube attribute*), 10
sums (*cr.cube.cubepart._Slice attribute*), 22
sums (*cr.cube.cubepart._Strand attribute*), 27

T

tab_alias (*cr.cube.cubepart._Slice attribute*), 22
tab_alias (*cr.cube.cubepart._Strand attribute*), 27
tab_label (*cr.cube.cubepart._Slice attribute*), 22
tab_label (*cr.cube.cubepart._Strand attribute*), 27
table_base (*cr.cube.cubepart._Nub attribute*), 28
table_base (*cr.cube.cubepart._Slice attribute*), 22
table_base_range (*cr.cube.cubepart._Slice attribute*), 22
table_base_range (*cr.cube.cubepart._Strand attribute*), 27
table_margin (*cr.cube.cubepart._Slice attribute*), 22
table_margin_range (*cr.cube.cubepart._Slice attribute*), 22
table_margin_range (*cr.cube.cubepart._Strand attribute*), 27
table_name (*cr.cube.cubepart._Slice attribute*), 23
table_name (*cr.cube.cubepart._Strand attribute*), 27
table_percentages (*cr.cube.cubepart._Strand attribute*), 27
table_proportion_moes (*cr.cube.cubepart._Strand attribute*), 27
table_proportion_stddevs (*cr.cube.cubepart._Strand attribute*), 27
table_proportion_stderrs (*cr.cube.cubepart._Strand attribute*), 27
table_proportion_variances (*cr.cube.cubepart._Slice attribute*), 23
table_proportions (*cr.cube.cubepart._Slice attribute*), 23
table_proportions (*cr.cube.cubepart._Strand attribute*), 27
table_proportions_moe (*cr.cube.cubepart._Slice attribute*), 23
table_std_dev (*cr.cube.cubepart._Slice attribute*), 23
table_std_err (*cr.cube.cubepart._Slice attribute*), 23
table_unweighted_bases (*cr.cube.cubepart._Slice attribute*), 23
table_weighted_bases (*cr.cube.cubepart._Slice attribute*), 23
title (*cr.cube.cube.Cube attribute*), 10
title (*cr.cube.cubepart._Strand attribute*), 27
total_share_sum (*cr.cube.cubepart._Slice attribute*), 23
translate_element_id() (*cr.cube.dimension.Dimension method*), 30

U

unweighted_bases (*cr.cube.cubepart._Strand attribute*), 27
unweighted_count (*cr.cube.cubepart._Nub attribute*), 28
unweighted_counts (*cr.cube.cube.Cube attribute*), 10
unweighted_counts (*cr.cube.cubepart._Slice attribute*), 23
unweighted_counts (*cr.cube.cubepart._Strand attribute*), 28
unweighted_valid_counts (*cr.cube.cube.Cube attribute*), 11

V

valid_counts_summary_range (*cr.cube.cube.Cube attribute*), 11

valid_counts_summary_range (*cr.cube.cube.CubeSet attribute*), 12
valid_elements (*cr.cube.dimension.Dimension attribute*), 30
valid_overlaps (*cr.cube.cube.Cube attribute*), 11
variable_name (*cr.cube.cubepart.CubePartition attribute*), 13

W

weighted_bases (*cr.cube.cubepart._Strand attribute*), 28
weighted_counts (*cr.cube.cube.Cube attribute*), 11
weighted_counts (*cr.cube.cubepart._Slice attribute*), 23
weighted_counts (*cr.cube.cubepart._Strand attribute*), 28
weighted_valid_counts (*cr.cube.cube.Cube attribute*), 11

Z

zscores (*cr.cube.cubepart._Slice attribute*), 23