

Package: opendp (via r-universe)

October 11, 2026

Version 0.15.1

Title R Bindings for the OpenDP Library

Description The OpenDP Library is a modular collection of statistical algorithms that adhere to the definition of differential privacy. It can be used to build applications of privacy-preserving computations, using a number of different models of privacy.

License MIT + file LICENSE

Depends R (≥ 4.2)

Suggests testthat ($\geq 3.0.0$), jsonlite

NeedsCompilation yes

URL <https://opendp.org/>, <https://github.com/opendp/opendp>,
<https://docs.opendp.org/en/stable/index.html>

BugReports <https://github.com/opendp/opendp/issues>

Encoding UTF-8

Roxygen list(markdown = TRUE)

SystemRequirements Cargo (Rust's package manager), rustc

Config/testthat/edition 3

Config/opendp/MSRV 1.64.0

Collate 'mod.R' 'extras.R' 'typing.R' 'internal.R' 'accuracy.R'
'combinators.R' 'core.R' 'data.R' 'domains.R' 'measurements.R'
'measures.R' 'metrics.R' 'opendp-package.R' 'transformations.R'

Config/roxygen2/version 8.1.0

Config/pak/sysreqs libclang-dev

Repository <https://opendp-dev.r-universe.dev>

Date/Publication 2026-10-07 12:16:40 UTC

RemoteUrl <https://github.com/opendp/opendp>

RemoteRef r-universe-dev

RemoteSha e9c24d3366d3556b1ecc0cfa0ddeaf335fa60aa7

RemoteSubdir R/opendp

Contents

absolute_distance	8
accuracy_to_discrete_gaussian_scale	8
accuracy_to_discrete_laplacian_scale	9
accuracy_to_gaussian_scale	10
accuracy_to_laplacian_scale	10
approximate	11
atom_domain	12
binary_search	12
binary_search_chain	13
binary_search_param	14
BitVector	15
bitvector_domain	15
bool	15
change_one_distance	16
choose_branching_factor	16
disable_features	17
discrete_distance	17
discrete_gaussian_scale_to_accuracy	17
discrete_laplacian_scale_to_accuracy	18
domain_carrier_type	19
domain_debug	19
domain_type	20
enable_features	20
ExtrinsicObject	20
f32	21
f64	21
fixed_smoothed_max_divergence	21
function_eval	22
gaussian_scale_to_accuracy	22
hamming_distance	23
hashitems	23
i128	24
i16	24
i32	24
i64	24
i8	25
insert_delete_distance	25
l01inf_distance	25
l02inf_distance	26
l1_distance	26
l2_distance	27
laplacian_scale_to_accuracy	27
linf_distance	28
make_adaptive_composition	28
make_alp_queryable	29
make_approximate	31
make_b_ary_tree	31

make_basic_composition	32
make_bounded_float_checked_sum	33
make_bounded_float_ordered_sum	34
make_bounded_int_monotonic_sum	35
make_bounded_int_ordered_sum	36
make_bounded_int_split_sum	37
make_canonical_noise	38
make_cast	39
make_cast_default	40
make_cast_inherent	41
make_cdf	42
make_chain_mt	42
make_chain_pm	43
make_chain_tt	44
make_clamp	44
make_composition	45
make_consistent_b_ary_tree	46
make_count	47
make_count_by	48
make_count_by_categories	49
make_count_distinct	50
make_create_dataframe	51
make_df_cast_default	51
make_df_is_equal	52
make_drop_null	53
make_find	54
make_find_bin	55
make_fix_delta	56
make_fixed_approxDP_to_approxDP	56
make_fully_adaptive_composition	57
make_gaussian	58
make_gaussian_threshold	59
make_geometric	61
make_identity	62
make_impute_constant	63
make_impute_uniform_float	64
make_index	64
make_is_equal	65
make_is_null	66
make_laplace	67
make_laplace_threshold	68
make_lipschitz_float_mul	69
make_mean	70
make_metric_bounded	71
make_metric_unbounded	72
make_noise	73
make_noise_threshold	74
make_noisy_max	75

make_noisy_top_k	76
make_ordered_random	77
make_population_amplification	78
make_privacy_filter	78
make_private_quantile	79
make_pureDP_to_zCDP	81
make_quantile_score_candidates	81
make_quantiles_from_counts	82
make_randomized_response	83
make_randomized_response_bitvec	84
make_randomized_response_bool	85
make_report_noisy_max_gumbel	86
make_resize	87
make_select_column	88
make_select_private_candidate	88
make_sequential_composition	89
make_sized_bounded_float_checked_sum	91
make_sized_bounded_float_ordered_sum	92
make_sized_bounded_int_checked_sum	93
make_sized_bounded_int_monotonic_sum	94
make_sized_bounded_int_ordered_sum	95
make_sized_bounded_int_split_sum	96
make_split_dataframe	97
make_split_lines	97
make_split_records	98
make_subset_by	99
make_sum	99
make_sum_of_squared_deviations	100
make_unordered	101
make_user_measurement	102
make_user_transformation	103
make_variance	105
make_zCDP_to_approxDP	106
map_domain	106
max_divergence	107
measure_debug	107
measure_distance_type	108
measure_type	108
measurement_check	109
measurement_function	109
measurement_input_carrier_type	110
measurement_input_distance_type	110
measurement_input_domain	111
measurement_input_metric	111
measurement_invoke	112
measurement_map	112
measurement_output_distance_type	113
measurement_output_measure	113

metric_debug	114
metric_distance_type	114
metric_type	115
new_domain	115
new_function	116
new_function_internal	116
new_hashtab	117
new_measure	117
new_measurement	118
new_metric	118
new_odometer	118
new_odometer_queryable_internal	119
new_privacy_profile	119
new_privacy_profile_internal	120
new_queryable	120
new_queryable_internal	121
new_transformation	121
object_type	122
odometer_input_carrier_type	122
odometer_input_domain	123
odometer_input_metric	123
odometer_invoke	124
odometer_output_measure	124
odometer_queryable_invoke	125
odometer_queryable_invoke_type	125
odometer_queryable_privacy_loss	126
odometer_queryable_privacy_loss_type	126
opendp	127
option_domain	128
parse_or_infer	128
privacy_profile_delta	129
privacy_profile_epsilon	129
queryable_eval	130
queryable_query_type	130
renyi_divergence	131
rt_infer	131
rt_parse	132
smoothed_max_divergence	132
String	133
symmetric_distance	133
then_adaptive_composition	134
then_alp_queryable	134
then_approximate	135
then_b_ary_tree	136
then_basic_composition	136
then_bounded_float_checked_sum	137
then_bounded_float_ordered_sum	137
then_bounded_int_monotonic_sum	138

then_bounded_int_ordered_sum	138
then_bounded_int_split_sum	139
then_canonical_noise	139
then_cast	140
then_cast_default	140
then_cast_inherent	141
then_cdf	141
then_chain_mt	142
then_chain_pm	142
then_chain_tt	143
then_clamp	143
then_composition	144
then_consistent_b_ary_tree	144
then_count	145
then_count_by	145
then_count_by_categories	146
then_count_distinct	146
then_create_dataframe	147
then_df_cast_default	147
then_df_is_equal	148
then_drop_null	148
then_find	149
then_find_bin	149
then_fix_delta	150
then_fixed_approxDP_to_approxDP	150
then_fully_adaptive_composition	151
then_gaussian	151
then_gaussian_threshold	152
then_geometric	153
then_identity	153
then_impute_constant	154
then_impute_uniform_float	154
then_index	155
then_is_equal	155
then_is_null	156
then_laplace	156
then_laplace_threshold	157
then_lipschitz_float_mul	157
then_mean	158
then_metric_bounded	158
then_metric_unbounded	159
then_noise	159
then_noise_threshold	160
then_noisy_max	160
then_noisy_top_k	161
then_ordered_random	161
then_population_amplification	162
then_postprocess	162

then_privacy_filter	163
then_private_quantile	163
then_pureDP_to_zCDP	164
then_quantile_score_candidates	164
then_quantiles_from_counts	165
then_randomized_response	165
then_randomized_response_bitvec	166
then_randomized_response_bool	167
then_report_noisy_max_gumbel	167
then_resize	168
then_select_column	168
then_select_private_candidate	169
then_sequential_composition	169
then_sized_bounded_float_checked_sum	170
then_sized_bounded_float_ordered_sum	170
then_sized_bounded_int_checked_sum	171
then_sized_bounded_int_monotonic_sum	171
then_sized_bounded_int_ordered_sum	172
then_sized_bounded_int_split_sum	172
then_split_dataframe	173
then_split_lines	173
then_split_records	174
then_subset_by	174
then_sum	175
then_sum_of_squared_deviations	175
then_unordered	176
then_user_measurement	176
then_user_transformation	177
then_variance	178
then_zCDP_to_approxDP	178
to_str.default	179
to_str.hashtab	179
to_then	180
transformation_check	180
transformation_function	181
transformation_input_carrier_type	181
transformation_input_distance_type	182
transformation_input_domain	182
transformation_input_metric	183
transformation_invoke	183
transformation_map	184
transformation_output_distance_type	184
transformation_output_domain	185
transformation_output_metric	185
u128	186
u16	186
u32	186
u64	186

u8	187
user_distance	187
user_divergence	188
user_domain	188
usize	189
vector_domain	189
zero_concentrated_divergence	190

Index	191
-------	-----

absolute_distance	Construct an instance of the AbsoluteDistance metric.
-------------------	---

Description

[absolute_distance in Rust documentation.](#)

Usage

absolute_distance(.T)

Arguments

.T undocumented

Value

Metric

accuracy_to_discrete_gaussian_scale	Convert a desired accuracy (tolerance) into a discrete gaussian noise scale at a statistical significance level alpha.
-------------------------------------	--

Description

[accuracy_to_discrete_gaussian_scale in Rust documentation.](#)

Usage

accuracy_to_discrete_gaussian_scale(accuracy, alpha, .T = NULL)

Arguments

accuracy	Desired accuracy. A tolerance for how far values may diverge from the input to the mechanism.
alpha	Statistical significance, level-alpha, or (1. - alpha)100% confidence. Must be within (0, 1].
.T	Data type of accuracy and alpha

Details

Proof Definition:

([Proof Document](#))

accuracy_to_discrete_laplacian_scale

*Convert a desired **accuracy** (tolerance) into a discrete Laplacian noise scale at a statistical significance level **alpha**.*

Description

[accuracy_to_discrete_laplacian_scale](#) in Rust documentation.

Usage

```
accuracy_to_discrete_laplacian_scale(accuracy, alpha, .T = NULL)
```

Arguments

accuracy	Desired accuracy. A tolerance for how far values may diverge from the input to the mechanism.
alpha	Statistical significance, level- alpha , or (1. - alpha)100% confidence. Must be within (0, 1].
.T	Data type of accuracy and alpha

Details

Proof Definition:

([Proof Document](#))

Value

Discrete laplacian noise scale that meets the **accuracy** requirement at a given level-**alpha**.

accuracy_to_gaussian_scale

*Convert a desired accuracy (tolerance) into a gaussian noise scale at a statistical significance level **alpha**.*

Description

[accuracy_to_gaussian_scale in Rust documentation.](#)

Usage

```
accuracy_to_gaussian_scale(accuracy, alpha, .T = NULL)
```

Arguments

accuracy	Desired accuracy. A tolerance for how far values may diverge from the input to the mechanism.
alpha	Statistical significance, level- alpha , or (1. - alpha)100% confidence. Must be within (0, 1].
.T	Data type of accuracy and alpha

accuracy_to_laplacian_scale

*Convert a desired accuracy (tolerance) into a Laplacian noise scale at a statistical significance level **alpha**.*

Description

[accuracy_to_laplacian_scale in Rust documentation.](#)

Usage

```
accuracy_to_laplacian_scale(accuracy, alpha, .T = NULL)
```

Arguments

accuracy	Desired accuracy. A tolerance for how far values may diverge from the input to the mechanism.
alpha	Statistical significance, level- alpha , or (1. - alpha)100% confidence. Must be within (0, 1].
.T	Data type of accuracy and alpha

Value

Laplacian noise scale that meets the **accuracy** requirement at a given level-**alpha**.

<code>approximate</code>	<i>Privacy measure used to define δ-approximate PM-differential privacy.</i>
--------------------------	--

Description

In the following definition, d corresponds to privacy parameters (d', δ) when also quantified over all adjacent datasets (d' is the privacy parameter corresponding to privacy measure PM). That is, (d', δ) is no smaller than d (by product ordering), over all pairs of adjacent datasets x, x' where $Y \sim M(x)$, $Y' \sim M(x')$. $M(\cdot)$ is a measurement (commonly known as a mechanism). The measurement's input metric defines the notion of adjacency, and the measurement's input domain defines the set of possible datasets.

Usage

```
approximate(measure)
```

Arguments

`measure` inner privacy measure

Details

[approximate in Rust documentation.](#)

Proof Definition:

For any two distributions Y, Y' and 2-tuple $d = (d', \delta)$, where d' is the distance with respect to privacy measure PM, Y, Y' are d -close under the approximate PM measure whenever, for any choice of $\delta \in [0, 1]$, there exist events E (depending on Y) and E' (depending on Y') such that $\Pr[E] \geq 1 - \delta$, $\Pr[E'] \geq 1 - \delta$, and

$$D_{\text{PM}}^\delta(Y|_E, Y'|_{E'}) = D_{\text{PM}}(Y|_E, Y'|_{E'})$$

where $Y|_E$ denotes the distribution of Y conditioned on the event E .

Note that this δ is not privacy parameter δ until quantified over all adjacent datasets, as is done in the definition of a measurement.

Value

ApproximateDivergence

<code>atom_domain</code>	<i>Construct an instance of <code>AtomDomain</code>.</i>
--------------------------	--

Description

The domain defaults to unbounded if `bounds` is `None`, If `T` is float, `nan` defaults to `true`.

Usage

```
atom_domain(bounds = NULL, nan = NULL, .T = NULL)
```

Arguments

<code>bounds</code>	Optional bounds of elements in the domain, if the data type is numeric.
<code>nan</code>	Whether the domain may contain NaN, if the data type is float.
<code>.T</code>	The type of the atom.

Details

[atom_domain in Rust documentation.](#)

Value

`AtomDomain`

Examples

```
atom_domain(.T = "i32")
```

<code>binary_search</code>	<i>Find the closest passing value to the decision boundary of predicate</i>
----------------------------	---

Description

Missing bounds are inferred. Use `NULL` for either element of `bounds` to conduct a one-sided search, or omit `bounds` to conduct an exponential search.

Usage

```
binary_search(predicate, bounds = NULL, .T = NULL, return_sign = FALSE)
```

Arguments

predicate	a monotonic unary function from a number to a boolean
bounds	a two-element list of optional lower and upper bounds on the input of predicate ; use NULL for either bound to infer it
.T	type of argument to predicate , one of float or int; when set, takes precedence over the type inferred from bounds
return_sign	if True, also return the direction away from the decision boundary

Value

the discovered parameter within the bounds

Examples

```
binary_search(\(x) x <= -5L, bounds = list(-10L, NULL))
```

binary_search_chain	<i>Find the highest-utility (d_in, d_out)-close Transformation or Measurement.</i>
----------------------------	--

Description

Searches for the numeric parameter to **make_chain** that results in a computation that most tightly satisfies **d_out** when datasets differ by at most **d_in**, then returns the Transformation or Measurement corresponding to said parameter.

Usage

```
binary_search_chain(make_chain, d_in, d_out, bounds = NULL, .T = NULL)
```

Arguments

make_chain	a function that takes a number and returns a Transformation or Measurement
d_in	how far apart input datasets can be
d_out	how far apart output datasets or distributions can be
bounds	a two-element list of optional lower and upper bounds on the input of make_chain ; use NULL for either bound to infer it
.T	type of argument to make_chain , either "float" or "int"; when set, takes precedence over the type inferred from bounds

Details

See **binary_search_param** to retrieve the discovered parameter instead of the complete computation chain.

Value

a Transformation or Measurement (chain) that is (**d_in**, **d_out**)-close.

Examples

```
enable_features("contrib")
# create a sum transformation over the space of float vectors
s_vec <- c(vector_domain(atom_domain(.T = "float", nan = FALSE)), symmetric_distance())
t_sum <- s_vec |> then_clamp(c(0., 1.)) |> then_sum()

# find a measurement that satisfies epsilon = 1 when datasets differ by at most one record
m_sum <- binary_search_chain(\(s) t_sum |> then_laplace(s), d_in = 1L, d_out = 1.)
```

binary_search_param *Solve for the ideal constructor argument to make_chain*

Description

Searches for the numeric parameter to **make_chain** that results in a computation that most tightly satisfies **d_out** when datasets differ by at most **d_in**.

Usage

```
binary_search_param(make_chain, d_in, d_out, bounds = NULL, .T = NULL)
```

Arguments

make_chain	a function that takes a number and returns a Transformation or Measurement
d_in	how far apart input datasets can be
d_out	how far apart output datasets or distributions can be
bounds	a two-element list of optional lower and upper bounds on the input of make_chain ; use NULL for either bound to infer it
.T	type of argument to make_chain , either "float" or "int"; when set, takes precedence over the type inferred from bounds

Value

the parameter to **make_chain** that results in a (**d_in**, **d_out**)-close Transformation or Measurement

BitVector	<i>type signature for a BitVector</i>
-----------	---------------------------------------

Description

type signature for a BitVector

Usage

BitVector

bitvector_domain	<i>Construct an instance of BitVectorDomain.</i>
------------------	--

Description

Construct an instance of BitVectorDomain.

Usage

bitvector_domain(max_weight = NULL)

Arguments

max_weight The maximum number of positive bits.

Value

Domain

bool	<i>type signature for a boolean</i>
------	-------------------------------------

Description

type signature for a boolean

Usage

bool

`change_one_distance` *Construct an instance of the `ChangeOneDistance` metric.*

Description

Construct an instance of the `ChangeOneDistance` metric.

Usage

```
change_one_distance()
```

Value

Metric

`choose_branching_factor`

Returns an approximation to the ideal `branching_factor` for a dataset of a given size, that minimizes error in cdf and quantile estimates based on b-ary trees.

Description

Required features: `contrib`

Usage

```
choose_branching_factor(size_guess)
```

Arguments

`size_guess` A guess at the size of your dataset.

Details

[choose_branching_factor](#) in Rust documentation.

Citations:

- [QYL13 Understanding Hierarchical Methods for Differentially Private Histograms](#)

Value

int

disable_features	Disable features in the <i>opendp</i> package.
------------------	--

Description

Disable features in the *opendp* package.

Usage

```
disable_features(...)
```

Arguments

... features to disable

discrete_distance	Construct an instance of the <code>DiscreteDistance</code> metric.
-------------------	--

Description

Construct an instance of the `DiscreteDistance` metric.

Usage

```
discrete_distance()
```

Value

Metric

discrete_gaussian_scale_to_accuracy	Convert a discrete gaussian scale into an accuracy estimate (tolerance) at a statistical significance level <code>alpha</code> .
-------------------------------------	--

Description

[discrete_gaussian_scale_to_accuracy](#) in Rust documentation.

Usage

```
discrete_gaussian_scale_to_accuracy(scale, alpha, .T = NULL)
```

Arguments

<code>scale</code>	Gaussian noise scale.
<code>alpha</code>	Statistical significance, level- <code>alpha</code> , or $(1. - \text{alpha})100\%$ confidence. Must be within $(0, 1]$.
<code>.T</code>	Data type of <code>scale</code> and <code>alpha</code>

Details**Proof Definition:**

([Proof Document](#))

`discrete_laplacian_scale_to_accuracy`

Convert a discrete Laplacian scale into an accuracy estimate (tolerance) at a statistical significance level `alpha`.

Description

$\alpha = P[Y \geq \text{accuracy}]$, where $Y = |X - z|$, and $X \sim \mathcal{L}_Z(0, \text{scale})$. That is, X is a discrete Laplace random variable and Y is the distribution of the errors.

Usage

```
discrete_laplacian_scale_to_accuracy(scale, alpha, .T = NULL)
```

Arguments

<code>scale</code>	Discrete Laplacian noise scale.
<code>alpha</code>	Statistical significance, level- <code>alpha</code> , or $(1. - \text{alpha})100\%$ confidence. Must be within $(0, 1]$.
<code>.T</code>	Data type of <code>scale</code> and <code>alpha</code>

Details

This function returns a float accuracy. You can take the floor without affecting the coverage probability.

[discrete_laplacian_scale_to_accuracy](#) in Rust documentation.

Proof Definition:

([Proof Document](#))

domain_carrier_type	<i>Get the carrier type of a domain.</i>
---------------------	--

Description

Get the carrier type of a domain.

Usage

```
domain_carrier_type(this)
```

Arguments

this	The domain to retrieve the carrier type from.
------	---

Value

str

domain_debug	<i>Debug a domain.</i>
--------------	------------------------

Description

Debug a domain.

Usage

```
domain_debug(this)
```

Arguments

this	The domain to debug (stringify).
------	----------------------------------

Value

str

<code>domain_type</code>	<i>Get the type of a domain.</i>
--------------------------	----------------------------------

Description

Get the type of a domain.

Usage

```
domain_type(this)
```

Arguments

<code>this</code>	The domain to retrieve the type from.
-------------------	---------------------------------------

Value

str

<code>enable_features</code>	<i>Enable features for the <code>opendp</code> package.</i>
------------------------------	---

Description

See <https://docs.opendp.org/en/stable/api/user-guide/#feature-listing> for available features.

Usage

```
enable_features(...)
```

Arguments

<code>...</code>	features to enable
------------------	--------------------

<code>ExtrinsicObject</code>	<i>type signature for an arbitrary R object preserved across FFI</i>
------------------------------	--

Description

type signature for an arbitrary R object preserved across FFI

Usage

```
ExtrinsicObject
```

f32	<i>type signature for a 32-bit floating point number</i>
-----	--

Description

type signature for a 32-bit floating point number

Usage

f32

f64	<i>type signature for a 64-bit floating point number</i>
-----	--

Description

type signature for a 64-bit floating point number

Usage

f64

<code>fixed_smoothed_max_divergence</code>	<i>Privacy measure used to define (ϵ, δ)-approximate differential privacy.</i>
--	---

Description

In the following definition, d corresponds to (ϵ, δ) when also quantified over all adjacent datasets. That is, (ϵ, δ) is no smaller than d (by product ordering), over all pairs of adjacent datasets x, x' where $Y \sim M(x)$, $Y' \sim M(x')$. $M(\cdot)$ is a measurement (commonly known as a mechanism). The measurement's input metric defines the notion of adjacency, and the measurement's input domain defines the set of possible datasets.

Usage

`fixed_smoothed_max_divergence()`

Details**Proof Definition:**

For any two distributions Y, Y' and any 2-tuple d of non-negative numbers ϵ and δ , Y, Y' are d -close under the fixed smoothed max divergence measure whenever

$$D_{\infty}^{\delta}(Y, Y') = \max_{S \subseteq \text{Supp}(Y)} \left[\ln \frac{\Pr[Y \in S] + \delta}{\Pr[Y' \in S]} \right] \leq \epsilon.$$

Note that this ϵ and δ are not privacy parameters ϵ and δ until quantified over all adjacent datasets, as is done in the definition of a measurement.

Value

Measure

<code>function_eval</code>	<i>Eval the function with <code>arg</code>.</i>
----------------------------	---

Description

Eval the function with `arg`.

Usage

```
function_eval(this, arg, TI = NULL)
```

Arguments

<code>this</code>	Function to invoke.
<code>arg</code>	Input data to supply to the measurement. A member of the measurement's input domain.
<code>TI</code>	Input Type.

<code>gaussian_scale_to_accuracy</code>	<i>Convert a gaussian scale into an accuracy estimate (tolerance) at a statistical significance level <code>alpha</code>.</i>
---	---

Description

[gaussian_scale_to_accuracy in Rust documentation.](#)

Usage

```
gaussian_scale_to_accuracy(scale, alpha, .T = NULL)
```

Arguments

<code>scale</code>	Gaussian noise scale.
<code>alpha</code>	Statistical significance, level- <code>alpha</code> , or $(1. - \text{alpha})100\%$ confidence. Must be within $(0, 1]$.
<code>.T</code>	Data type of <code>scale</code> and <code>alpha</code>

<code>hamming_distance</code>	<i>Construct an instance of the <code>HammingDistance</code> metric.</i>
-------------------------------	--

Description

Construct an instance of the `HammingDistance` metric.

Usage

```
hamming_distance()
```

Value

Metric

<code>hashitems</code>	<i>extract heterogeneously typed keys and values from a hashtable</i>
------------------------	---

Description

extract heterogeneously typed keys and values from a hashtable

Usage

```
hashitems(data, type_name)
```

Arguments

<code>data</code>	a hashtable
<code>type_name</code>	the expected <code>runtime_type</code> of the hashtable

i128	<i>type signature for a 128-bit signed integer</i>
------	--

Description

type signature for a 128-bit signed integer

Usage

i128

i16	<i>type signature for a 16-bit signed integer</i>
-----	---

Description

type signature for a 16-bit signed integer

Usage

i16

i32	<i>type signature for a 32-bit signed integer</i>
-----	---

Description

type signature for a 32-bit signed integer

Usage

i32

i64	<i>type signature for a 64-bit signed integer</i>
-----	---

Description

type signature for a 64-bit signed integer

Usage

i64

i8	<i>type signature for an 8-bit signed integer</i>
----	---

Description

type signature for an 8-bit signed integer

Usage

i8

insert_delete_distance	<i>Construct an instance of the InsertDeleteDistance metric.</i>
------------------------	--

Description

Construct an instance of the InsertDeleteDistance metric.

Usage

insert_delete_distance()

Value

Metric

l01inf_distance	<i>Construct an instance of the L01InfDistance metric.</i>
-----------------	--

Description

[l01inf_distance](#) in Rust documentation.

Usage

l01inf_distance(metric)

Arguments

metric The metric used to compute distance between partitions.

Value

Metric

<code>l02inf_distance</code>	<i>Construct an instance of the <code>L02InfDistance</code> metric.</i>
------------------------------	---

Description

[l02inf_distance](#) in Rust documentation.

Usage

```
l02inf_distance(metric)
```

Arguments

<code>metric</code>	The metric used to compute distance between partitions.
---------------------	---

Value

Metric

<code>l1_distance</code>	<i>Construct an instance of the <code>L1Distance</code> metric.</i>
--------------------------	---

Description

[l1_distance](#) in Rust documentation.

Usage

```
l1_distance(.T)
```

Arguments

<code>.T</code>	undocumented
-----------------	--------------

Value

Metric

l2_distance	Construct an instance of the L2Distance metric.
-------------	---

Description

[l2_distance](#) in Rust documentation.

Usage

```
l2_distance(.T)
```

Arguments

.T	undocumented
----	--------------

Value

Metric

laplacian_scale_to_accuracy	Convert a Laplacian scale into an accuracy estimate (tolerance) at a statistical significance level alpha .
-----------------------------	--

Description

[laplacian_scale_to_accuracy](#) in Rust documentation.

Usage

```
laplacian_scale_to_accuracy(scale, alpha, .T = NULL)
```

Arguments

scale	Laplacian noise scale.
alpha	Statistical significance, level- alpha , or (1. - alpha)100% confidence. Must be within (0, 1].
.T	Data type of scale and alpha

<code>linf_distance</code>	<i>Construct an instance of the LInfDistance metric.</i>
----------------------------	--

Description

[linf_distance in Rust documentation.](#)

Usage

```
linf_distance(.T, monotonic = FALSE)
```

Arguments

<code>.T</code>	The type of the distance.
<code>monotonic</code>	set to true if non-monotonicity implies infinite distance

Value

Metric

<code>make_adaptive_composition</code>	<i>adaptive composition constructor</i>
--	---

Description

Construct a Measurement that when invoked, returns a queryable that interactively composes measurements.

Usage

```
make_adaptive_composition(
  input_domain,
  input_metric,
  output_measure,
  d_in,
  d_mids
)
```

Arguments

<code>input_domain</code>	indicates the space of valid input datasets
<code>input_metric</code>	how distances are measured between members of the input domain
<code>output_measure</code>	how privacy is measured
<code>d_in</code>	maximum distance between adjacent input datasets
<code>d_mids</code>	maximum privacy expenditure of each query

Details

Required features: `contrib`

[make_adaptive_composition](#) in Rust documentation.

Citations:

- [LW22 Composition Theorems for Interactive Differential Privacy](#)
- [VW21 Concurrent Composition of Differential Privacy](#)
- [HSTVVXZ23 Concurrent Composition for Interactive Differential Privacy with Adaptive Privacy-Loss Parameters](#)

Composition Properties

- sequential: all measurements are applied to the same dataset
- basic: the composition is the linear sum of the privacy usage of each query
- interactive: mechanisms can be specified based on answers to previous queries
- compositor: all privacy parameters specified up-front

If the privacy measure supports concurrency, this compositor allows you to spawn multiple interactive mechanisms and interleave your queries amongst them.

Supporting Elements:

- Input Domain: `DI`
- Output Type: `MI`
- Input Metric: `MO`
- Output Measure: `Queryable<Measurement<DI, MI, MO, TO>, TO>`

Value

Measurement

`make_alp_queryable` *alp queryable constructor*

Description

Measurement to release a queryable containing a DP projection of bounded sparse data.

Usage

```
make_alp_queryable(
    input_domain,
    input_metric,
    scale,
    total_limit,
    value_limit = NULL,
    size_factor = 50L,
    alpha = 4L
)
```

Arguments

<code>input_domain</code>	Domain of input data
<code>input_metric</code>	Metric on input domain
<code>scale</code>	Privacy loss parameter. This is equal to epsilon/sensitivity.
<code>total_limit</code>	Either the true value or an upper bound estimate of the sum of all values in the input.
<code>value_limit</code>	Upper bound on individual values (referred to as <code>beta</code>). Entries above <code>beta</code> are clamped.
<code>size_factor</code>	Optional multiplier (default of 50) for setting the size of the projection.
<code>alpha</code>	Optional parameter (default of 4) for scaling and determining <code>p</code> in randomized response step.

Details

The size of the projection is $O(\text{total} * \text{size_factor} * \text{scale} / \alpha)$. The evaluation time of post-processing is $O(\beta * \text{scale} / \alpha)$.

`size_factor` is an optional multiplier (defaults to 50) for setting the size of the projection. There is a memory/utility trade-off. The value should be sufficiently large to limit hash collisions.

Required features: `contrib`

[make_alp_queryable in Rust documentation.](#)

Citations:

- [ALP21 Differentially Private Sparse Vectors with Low Error, Optimal Space, and Fast Access](#) Algorithm 4

Supporting Elements:

- Input Domain: `MapDomain<AtomDomain<K>, AtomDomain<CI>>`
- Output Type: `L01InfDistance<AbsoluteDistance<CI>>`
- Input Metric: `MaxDivergence`
- Output Measure: `Queryable<K, f64>`

Value

Measurement

make_approximate	<i>approximate constructor</i>
------------------	--------------------------------

Description

Constructs a new output measurement where the output measure is `-approximate`, where `=0`.

Usage

```
make_approximate(measurement)
```

Arguments

`measurement` a measurement with a privacy measure to be casted

Details

Required features: `contrib`
[make_approximate in Rust documentation.](#)

Value

Measurement

make_b_ary_tree	<i>b ary tree constructor</i>
-----------------	-------------------------------

Description

Expand a vector of counts into a b-ary tree of counts, where each branch is the sum of its `b` immediate children.

Usage

```
make_b_ary_tree(input_domain, input_metric, leaf_count, branching_factor)
```

Arguments

`input_domain` Domain of input data
`input_metric` Metric on input domain
`leaf_count` The number of leaf nodes in the b-ary tree.
`branching_factor` The number of children on each branch of the resulting tree. Larger branching factors result in shallower trees.

Details

Required features: `contrib`

`make_b_ary_tree` in [Rust documentation](#).

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<TA>>`
- Output Domain: `M`
- Input Metric: `VectorDomain<AtomDomain<TA>>`
- Output Metric: `M`

Value

Transformation

`make_basic_composition`

basic composition constructor

Description

Construct the DP composition `[measurement0, measurement1, ...]`. Returns a `Measurement` that when invoked, computes `[measurement0(x), measurement1(x), ...]`

Usage

```
make_basic_composition(measurements)
```

Arguments

`measurements` A vector of `Measurements` to compose.

Details

All metrics and domains must be equivalent.

Composition Properties

- sequential: all measurements are applied to the same dataset
- basic: the composition is the linear sum of the privacy usage of each query
- noninteractive: all mechanisms specified up-front (but each can be interactive)
- compositor: all privacy parameters specified up-front (via the map)

Required features: `contrib`

Value

`Measurement`

```
make_bounded_float_checked_sum
```

bounded float checked sum constructor

Description

Make a Transformation that computes the sum of bounded data with known dataset size.

Usage

```
make_bounded_float_checked_sum(size_limit, bounds, .S = "Pairwise<.T>")
```

Arguments

<code>size_limit</code>	Upper bound on number of records to keep in the input data.
<code>bounds</code>	Tuple of lower and upper bounds for data in the input domain.
<code>.S</code>	Summation algorithm to use over some data type T (T is shorthand for <code>S::Item</code>)

Details

This uses a restricted-sensitivity proof that takes advantage of known dataset size for better utility. Use `make_clamp` to bound data and `make_resize` to establish dataset size.

S (summation algorithm)	input type
<code>Sequential<S::Item></code>	<code>Vec<S::Item></code>
<code>Pairwise<S::Item></code>	<code>Vec<S::Item></code>

`S::Item` is the type of all of the following: each bound, each element in the input data, the output data, and the output sensitivity.

For example, to construct a transformation that pairwise-sums `f32` half-precision floats, set `S` to `Pairwise<f32>`.

Required features: `contrib`

[make_bounded_float_checked_sum in Rust documentation.](#)

Citations:

- [CSVW22 Widespread Underestimation of Sensitivity...](#)
- [DMNS06 Calibrating Noise to Sensitivity in Private Data Analysis](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<S::Item>>`
- Output Domain: `SymmetricDistance`
- Input Metric: `AtomDomain<S::Item>`
- Output Metric: `AbsoluteDistance<S::Item>`

Value

Transformation

`make_bounded_float_ordered_sum`*bounded float ordered sum constructor*

Description

Make a Transformation that computes the sum of bounded floats with known ordering.

Usage

```
make_bounded_float_ordered_sum(size_limit, bounds, .S = "Pairwise<T>")
```

Arguments

<code>size_limit</code>	Upper bound on the number of records in input data. Used to bound sensitivity.
<code>bounds</code>	Tuple of lower and upper bounds for data in the input domain.
<code>.S</code>	Summation algorithm to use over some data type T (T is shorthand for <code>S::Item</code>)

Details

Only useful when `make_bounded_float_checked_sum` returns an error due to potential for overflow. You may need to use `make_ordered_random` to impose an ordering on the data. The utility loss from overestimating the `size_limit` is small.

S (summation algorithm)	input type
<code>Sequential<S::Item></code>	<code>Vec<S::Item></code>
<code>Pairwise<S::Item></code>	<code>Vec<S::Item></code>

`S::Item` is the type of all of the following: each bound, each element in the input data, the output data, and the output sensitivity.

For example, to construct a transformation that pairwise-sums `f32` half-precision floats, set `S` to `Pairwise<f32>`.

Required features: `contrib`

[make_bounded_float_ordered_sum](#) in Rust documentation.

Citations:

- [CSVW22 Widespread Underestimation of Sensitivity...](#)
- [DMNS06 Calibrating Noise to Sensitivity in Private Data Analysis](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<S::Item>>`
- Output Domain: `InsertDeleteDistance`
- Input Metric: `AtomDomain<S::Item>`
- Output Metric: `AbsoluteDistance<S::Item>`

Value

Transformation

`make_bounded_int_monotonic_sum`
*bounded int monotonic sum constructor***Description**

Make a Transformation that computes the sum of bounded ints, where all values share the same sign.

Usage

```
make_bounded_int_monotonic_sum(bounds, .T = NULL)
```

Arguments

<code>bounds</code>	Tuple of lower and upper bounds for data in the input domain.
<code>.T</code>	Atomic Input Type and Output Type

Details

Required features: `contrib`

[make_bounded_int_monotonic_sum](#) in Rust documentation.

Citations:

- [CSVW22 Widespread Underestimation of Sensitivity...](#)
- [DMNS06 Calibrating Noise to Sensitivity in Private Data Analysis](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<T>>`
- Output Domain: `SymmetricDistance`
- Input Metric: `AtomDomain<T>`
- Output Metric: `AbsoluteDistance<T>`

Value

Transformation

```
make_bounded_int_ordered_sum
    bounded int ordered sum constructor
```

Description

Make a Transformation that computes the sum of bounded ints. You may need to use `make_ordered_random` to impose an ordering on the data.

Usage

```
make_bounded_int_ordered_sum(bounds, .T = NULL)
```

Arguments

<code>bounds</code>	Tuple of lower and upper bounds for data in the input domain.
<code>.T</code>	Atomic Input Type and Output Type

Details

Required features: `contrib`

[make_bounded_int_ordered_sum](#) in Rust documentation.

Citations:

- [CSVW22 Widespread Underestimation of Sensitivity...](#)
- [DMNS06 Calibrating Noise to Sensitivity in Private Data Analysis](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<T>>`
- Output Domain: `InsertDeleteDistance`
- Input Metric: `AtomDomain<T>`
- Output Metric: `AbsoluteDistance<T>`

Value

Transformation

`make_bounded_int_split_sum`
bounded int split sum constructor

Description

Make a Transformation that computes the sum of bounded ints. Adds the saturating sum of the positives to the saturating sum of the negatives.

Usage

```
make_bounded_int_split_sum(bounds, .T = NULL)
```

Arguments

<code>bounds</code>	Tuple of lower and upper bounds for data in the input domain.
<code>.T</code>	Atomic Input Type and Output Type

Details

Required features: `contrib`

[make_bounded_int_split_sum](#) in Rust documentation.

Citations:

- [CSVW22 Widespread Underestimation of Sensitivity...](#)
- [DMNS06 Calibrating Noise to Sensitivity in Private Data Analysis](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<T>>`
- Output Domain: `SymmetricDistance`
- Input Metric: `AtomDomain<T>`
- Output Metric: `AbsoluteDistance<T>`

Value

Transformation

`make_canonical_noise` *canonical noise constructor*

Description

Make a Measurement that adds noise from a canonical noise distribution. The implementation is tailored towards approximate-DP, resulting in noise sampled from the Tulap distribution.

Usage

```
make_canonical_noise(input_domain, input_metric, d_in, d_out)
```

Arguments

<code>input_domain</code>	Domain of the input.
<code>input_metric</code>	Metric of the input.
<code>d_in</code>	Sensitivity
<code>d_out</code>	Privacy parameters (,)

Details

Required features: `contrib`

[make_canonical_noise](#) in Rust documentation.

Citations:

- [AV23 Canonical Noise Distributions and Private Hypothesis Tests](#)

Supporting Elements:

- Input Domain: `AtomDomain<f64>`
- Output Type: `AbsoluteDistance<f64>`
- Input Metric: `Approximate<MaxDivergence>`
- Output Measure: `f64`

Proof Definition:

[\(Proof Document\)](#)

Value

Measurement

make_cast	cast constructor
-----------	------------------

Description

Make a Transformation that casts a vector of data from type TIA to type TOA. For each element, failure to parse results in `None`, else `Some(out)`.

Usage

```
make_cast(input_domain, input_metric, .TOA)
```

Arguments

input_domain	Domain of input data
input_metric	Metric on input domain
.TOA	Atomic Output Type to cast into

Details

Can be chained with `make_impute_constant` or `make_drop_null` to handle nullity.

Required features: `contrib`

[make_cast in Rust documentation.](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<TIA>>`
- Output Domain: `M`
- Input Metric: `VectorDomain<OptionDomain<AtomDomain<TOA>>>`
- Output Metric: `M`

Value

Transformation

make_cast_default	<i>cast default constructor</i>
-------------------	---------------------------------

Description

Make a Transformation that casts a vector of data from type TIA to type TOA. Any element that fails to cast is filled with default.

Usage

```
make_cast_default(input_domain, input_metric, .TOA)
```

Arguments

input_domain	Domain of input data
input_metric	Metric on input domain
.TOA	Atomic Output Type to cast into

Details

TIA	TIA::default()
float	0.
int	0
string	""
bool	false

Required features: `contrib`

[make_cast_default in Rust documentation.](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<TIA>>`
- Output Domain: `M`
- Input Metric: `VectorDomain<AtomDomain<TOA>>`
- Output Metric: `M`

Value

Transformation

make_cast_inherent	<i>cast inherent constructor</i>
--------------------	----------------------------------

Description

Make a Transformation that casts a vector of data from type `TIA` to a type that can represent nullity `TOA`. If cast fails, fill with `TOA`'s null value.

Usage

```
make_cast_inherent(input_domain, input_metric, .TOA)
```

Arguments

<code>input_domain</code>	Domain of input data
<code>input_metric</code>	Metric on input domain
<code>.TOA</code>	Atomic Output Type to cast into

Details

<code>TIA</code>	<code>TIA::default()</code>
<code>float</code>	<code>NaN</code>

Required features: `contrib`

[make_cast_inherent in Rust documentation.](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<TIA>>`
- Output Domain: `M`
- Input Metric: `VectorDomain<AtomDomain<TOA>>`
- Output Metric: `M`

Value

Transformation

make_cdf	<i>cdf constructor</i>
----------	------------------------

Description

Postprocess a noisy array of float summary counts into a cumulative distribution.

Usage

```
make_cdf(.TA = "float")
```

Arguments

.TA Atomic Type. One of f32 or f64

Details

Required features: `contrib`

[make_cdf in Rust documentation.](#)

Supporting Elements:

- Input Type: `Vec<TA>`
- Output Type: `Vec<TA>`

Value

Function

make_chain_mt	<i>chain mt constructor</i>
---------------	-----------------------------

Description

Construct the functional composition (`measurement1 transformation0`). Returns a Measurement that when invoked, computes `measurement1(transformation0(x))`.

Usage

```
make_chain_mt(measurement1, transformation0)
```

Arguments

measurement1 outer mechanism
transformation0 inner transformation

Details

Required features: `contrib`

[make_chain_mt](#) in Rust documentation.

Value

Measurement

make_chain_pm	<i>chain pm constructor</i>
---------------	-----------------------------

Description

Construct the functional composition (`postprocess1 measurement0`). Returns a Measurement that when invoked, computes `postprocess1(measurement0(x))`. Used to represent non-interactive postprocessing.

Usage

```
make_chain_pm(postprocess1, measurement0)
```

Arguments

`postprocess1` outer postprocessor

`measurement0` inner measurement/mechanism

Details

Required features: `contrib`

[make_chain_pm](#) in Rust documentation.

Value

Measurement

make_chain_tt	<i>chain tt constructor</i>
---------------	-----------------------------

Description

Construct the functional composition (`transformation1 transformation0`). Returns a Transformation that when invoked, computes `transformation1(transformation0(x))`.

Usage

```
make_chain_tt(transformation1, transformation0)
```

Arguments

<code>transformation1</code>	outer transformation
<code>transformation0</code>	inner transformation

Details

Required features: `contrib`

[make_chain_tt in Rust documentation.](#)

Value

Transformation

make_clamp	<i>clamp constructor</i>
------------	--------------------------

Description

Make a Transformation that clamps numeric data in `Vec<TA>` to `bounds`.

Usage

```
make_clamp(input_domain, input_metric, bounds)
```

Arguments

<code>input_domain</code>	Domain of input data.
<code>input_metric</code>	Metric on input domain.
<code>bounds</code>	Tuple of inclusive lower and upper bounds.

Details

If datum is less than lower, let datum be lower. If datum is greater than upper, let datum be upper.

Required features: `contrib`

[make_clamp in Rust documentation.](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<TA>>`
- Output Domain: `M`
- Input Metric: `VectorDomain<AtomDomain<TA>>`
- Output Metric: `M`

Proof Definition:

[\(Proof Document\)](#)

Value

Transformation

<code>make_composition</code>	<i>composition constructor</i>
-------------------------------	--------------------------------

Description

Construct the DP composition `[measurement0, measurement1, ...]`. Returns a Measurement that when invoked, computes `[measurement0(x), measurement1(x), ...]`

Usage

```
make_composition(measurements)
```

Arguments

`measurements` A vector of Measurements to compose.

Details

All metrics and domains must be equivalent.

Composition Properties

- sequential: all measurements are applied to the same dataset
- basic: the composition is the linear sum of the privacy usage of each query
- noninteractive: all mechanisms specified up-front (but each can be interactive)
- compositor: all privacy parameters specified up-front (via the map)

Required features: `contrib`

Value

Measurement

```
make_consistent_b_ary_tree
```

consistent b ary tree constructor

Description

Postprocessor that makes a noisy b-ary tree internally consistent, and returns the leaf layer.

Usage

```
make_consistent_b_ary_tree(branching_factor, .TIA = "int", .TOA = "float")
```

Arguments

<code>branching_factor</code>	the maximum number of children
<code>.TIA</code>	Atomic type of the input data. Should be an integer type.
<code>.TOA</code>	Atomic type of the output data. Should be a float type.

Details

The input argument of the function is a balanced **b**-ary tree implicitly stored in breadth-first order. Tree is assumed to be complete, as in, all leaves on the last layer are on the left. Non-existent leaves are assumed to be zero.

The output remains consistent even when leaf nodes are missing. This is due to an adjustment to the original algorithm to apportion corrections to children relative to their variance.

Required features: `contrib`

[make_consistent_b_ary_tree](#) in Rust documentation.

Citations:

- [HRMS09 Boosting the Accuracy of Differentially Private Histograms Through Consistency, section 4.1](#)

Supporting Elements:

- Input Type: `Vec<TIA>`
- Output Type: `Vec<TOA>`

Value

Function

make__count	<i>count constructor</i>
-------------	--------------------------

Description

Make a Transformation that computes a count of the number of records in data.

Usage

```
make_count(input_domain, input_metric, .T0 = "int")
```

Arguments

<code>input_domain</code>	Domain of the data type to be released.
<code>input_metric</code>	Metric of the data type to be released.
<code>.T0</code>	Output Type. Must be numeric.

Details

Required features: `contrib`

[make__count in Rust documentation.](#)

Citations:

- [GRS12 Universally Utility-Maximizing Privacy Mechanisms](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<TIA>>`
- Output Domain: `SymmetricDistance`
- Input Metric: `AtomDomain<T0>`
- Output Metric: `AbsoluteDistance<T0>`

Proof Definition:

[\(Proof Document\)](#)

Value

Transformation

make_count_by	<i>count by constructor</i>
---------------	-----------------------------

Description

Make a Transformation that computes the count of each unique value in data. This assumes that the category set is unknown.

Usage

```
make_count_by(input_domain, input_metric, .TV = "int")
```

Arguments

<code>input_domain</code>	Domain of input data
<code>input_metric</code>	Metric on input domain
<code>.TV</code>	Type of Value. Express counts in terms of this integral type.

Details

Required features: `contrib`

[make_count_by](#) in Rust documentation.

Citations:

- [BV17 Differential Privacy on Finite Computers](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<TK>>`
- Output Domain: `SymmetricDistance`
- Input Metric: `MapDomain<AtomDomain<TK>, AtomDomain<TV>>`
- Output Metric: `L01InfDistance<AbsoluteDistance<TV>>`

Value

The carrier type is `HashMap<TK, TV>`, a hashmap of the count (TV) for each unique data input (TK).

```
make_count_by_categories
    count by categories constructor
```

Description

Make a Transformation that computes the number of times each category appears in the data. This assumes that the category set is known.

Usage

```
make_count_by_categories(
    input_domain,
    input_metric,
    categories,
    null_category = TRUE,
    .MO = "L1Distance<int>",
    .TOA = "int"
)
```

Arguments

<code>input_domain</code>	Domain of input data
<code>input_metric</code>	Metric on input domain
<code>categories</code>	The set of categories to compute counts for.
<code>null_category</code>	Include a count of the number of elements that were not in the category set at the end of the vector.
<code>.MO</code>	Output Metric.
<code>.TOA</code>	Atomic Output Type that is numeric.

Details

Required features: `contrib`

[make_count_by_categories in Rust documentation.](#)

Citations:

- [GRS12 Universally Utility-Maximizing Privacy Mechanisms](#)
- [BV17 Differential Privacy on Finite Computers](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<TIA>>`
- Output Domain: `SymmetricDistance`
- Input Metric: `VectorDomain<AtomDomain<TOA>>`
- Output Metric: `MO`

Value

The carrier type is `Vec<TOA>`, a vector of the counts (`TOA`).

<code>make_count_distinct</code>	<i>count distinct constructor</i>
----------------------------------	-----------------------------------

Description

Make a Transformation that computes a count of the number of unique, distinct records in data.

Usage

```
make_count_distinct(input_domain, input_metric, .TO = "int")
```

Arguments

<code>input_domain</code>	Domain of input data
<code>input_metric</code>	Metric on input domain
<code>.TO</code>	Output Type. Must be numeric.

Details

Required features: `contrib`

[make_count_distinct](#) in Rust documentation.

Citations:

- [GRS12 Universally Utility-Maximizing Privacy Mechanisms](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<TIA>>`
- Output Domain: `SymmetricDistance`
- Input Metric: `AtomDomain<TO>`
- Output Metric: `AbsoluteDistance<TO>`

Value

Transformation

```
make_create_dataframe
```

create dataframe constructor

Description

Make a Transformation that constructs a dataframe from a `Vec<Vec<String>>` (a vector of records).

Usage

```
make_create_dataframe(col_names, .K = NULL)
```

Arguments

<code>col_names</code>	Column names for each record entry.
<code>.K</code>	categorical/hashable data type of column names

Details

Required features: `contrib`

[make_create_dataframe in Rust documentation.](#)

Supporting Elements:

- Input Domain: `VectorDomain<VectorDomain<AtomDomain<String>>>`
- Output Domain: `SymmetricDistance`
- Input Metric: `DataFrameDomain<K>`
- Output Metric: `SymmetricDistance`

Value

Transformation

```
make_df_cast_default
```

df cast default constructor

Description

Make a Transformation that casts the elements in a column in a dataframe from type `TIA` to type `TOA`. If cast fails, fill with default.

Usage

```
make_df_cast_default(input_domain, input_metric, column_name, .TIA, .TOA)
```

Arguments

input_domain	Domain of input data
input_metric	Metric on input domain
column_name	column name to be transformed
.TIA	Atomic Input Type to cast from
.TOA	Atomic Output Type to cast into

Details

TIA	TIA::default()
float	0.
int	0
string	""
bool	false

Required features: contrib
make_df_cast_default in Rust documentation.

Supporting Elements:

- Input Domain: DataFrameDomain<TK>
- Output Domain: M
- Input Metric: DataFrameDomain<TK>
- Output Metric: M

Value

Transformation

make_df_is_equal	<i>df is equal constructor</i>
------------------	--------------------------------

Description

Make a Transformation that checks if each element in a column in a dataframe is equivalent to value.

Usage

make_df_is_equal(input_domain, input_metric, column_name, value, .TIA = NULL)

Arguments

input_domain	Domain of input data
input_metric	Metric on input domain
column_name	Column name to be transformed
value	Value to check for equality
.TIA	Atomic Input Type to cast from

Details

Required features: `contrib`
`make_df_is_equal` in [Rust documentation](#).

Supporting Elements:

- Input Domain: `DataFrameDomain<TK>`
- Output Domain: `M`
- Input Metric: `DataFrameDomain<TK>`
- Output Metric: `M`

Value

Transformation

make_drop_null	<i>drop null constructor</i>
----------------	------------------------------

Description

Make a Transformation that drops null values.

Usage

```
make_drop_null(input_domain, input_metric)
```

Arguments

input_domain	Domain of input data
input_metric	Metric on input domain

Details

```

input_domain
vector_domain(option_domain(atom_domain(TA)))
vector_domain(atom_domain(TA))

```

Required features: `contrib`

[make_drop_null](#) in Rust documentation.

Supporting Elements:

- Input Domain: `VectorDomain<DIA>`
- Output Domain: `M`
- Input Metric: `VectorDomain<AtomDomain<DIA::Imputed>>`
- Output Metric: `M`

Value

Transformation

<code>make_find</code>	<i>find constructor</i>
------------------------	-------------------------

Description

Find the index of a data value in a set of categories.

Usage

```
make_find(input_domain, input_metric, categories)
```

Arguments

<code>input_domain</code>	The domain of the input vector.
<code>input_metric</code>	The metric of the input vector.
<code>categories</code>	The set of categories to find indexes from.

Details

For each value in the input vector, finds the index of the value in `categories`. If an index is found, returns `Some(index)`, else `None`. Chain with `make_impute_constant` or `make_drop_null` to handle nullity.

Required features: `contrib`

[make_find](#) in Rust documentation.

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<TIA>>`
- Output Domain: `M`
- Input Metric: `VectorDomain<OptionDomain<AtomDomain<usize>>>`
- Output Metric: `M`

Value

Transformation

<code>make_find_bin</code>	<i>find bin constructor</i>
----------------------------	-----------------------------

Description

Make a transformation that finds the bin index in a monotonically increasing vector of edges.

Usage

```
make_find_bin(input_domain, input_metric, edges)
```

Arguments

<code>input_domain</code>	The domain of the input vector.
<code>input_metric</code>	The metric of the input vector.
<code>edges</code>	The set of edges to split bins by.

Details

For each value in the input vector, finds the index of the bin the value falls into. `edges` splits the entire range of `TIA` into bins. The first bin at index zero ranges from negative infinity to the first edge, non-inclusive. The last bin at index `edges.len()` ranges from the last bin, inclusive, to positive infinity.

To be valid, `edges` must be unique and ordered. `edges` are left inclusive, right exclusive.

Required features: `contrib`

[make_find_bin in Rust documentation.](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<TIA>>`
- Output Domain: `M`
- Input Metric: `VectorDomain<AtomDomain<usize>>`
- Output Metric: `M`

Value

Transformation

make_fix_delta	<i>fix delta constructor</i>
----------------	------------------------------

Description

Fix the delta parameter in the privacy map of a `measurement` with a `SmoothedMaxDivergence` output measure.

Usage

```
make_fix_delta(measurement, delta)
```

Arguments

<code>measurement</code>	a measurement with a privacy curve to be fixed
<code>delta</code>	parameter to fix the privacy curve with

Details

Required features: `contrib`
[make_fix_delta in Rust documentation.](#)

Value

Measurement

make_fixed_approxDP_to_approxDP	<i>fixed approxDP to approxDP constructor</i>
---------------------------------	---

Description

Constructs a new output measurement where the output measure is casted from `Approximate<MaxDivergence>` to `SmoothedMaxDivergence`.

Usage

```
make_fixed_approxDP_to_approxDP(measurement)
```

Arguments

<code>measurement</code>	a measurement with a privacy measure to be casted
--------------------------	---

Details

Required features: `contrib`
[make_fixed_approxDP_to_approxDP in Rust documentation.](#)

Value

Measurement

```
make_fully_adaptive_composition
```

fully adaptive composition constructor

Description

Construct an odometer that can spawn a compositor queryable.

Usage

```
make_fully_adaptive_composition(input_domain, input_metric, output_measure)
```

Arguments

`input_domain` indicates the space of valid input datasets
`input_metric` how distances are measured between members of the input domain
`output_measure` how privacy is measured

Details

Required features: `contrib`

[make_fully_adaptive_composition](#) in Rust documentation.

Citations:

- [WRRW23 Fully Adaptive Composition in Differential Privacy](#)
- [VZ23 Concurrent Composition Theorems for Differential Privacy](#)
- [HSTVVXZ23 Concurrent Composition for Interactive Differential Privacy with Adaptive Privacy-Loss Parameters](#)

Supporting Elements:

- Input Domain `DI`
- Input Metric `MI`
- Output Measure `MO`
- Query Measurement `<DI, MI, MO, T0>`
- Answer `T0`

Proof Definition:

[\(Proof Document\)](#)

Value

Odometer

make_gaussian	<i>gaussian constructor</i>
---------------	-----------------------------

Description

Make a Measurement that adds noise from the Gaussian(scale) distribution to the input.

Usage

```
make_gaussian(
  input_domain,
  input_metric,
  scale,
  k = NULL,
  .MO = "ZeroConcentratedDivergence"
)
```

Arguments

input_domain	Domain of the data type to be released.
input_metric	Metric of the data type to be released.
scale	Noise scale parameter for the gaussian distribution. <code>scale == standard_deviation</code> .
k	The noise granularity in terms of 2^k .
.MO	Output Measure. The only valid measure is <code>ZeroConcentratedDivergence</code> .

Details

Valid inputs for `input_domain` and `input_metric` are:

input_domain	input type	input_metric
atom_domain(T)	T	absolute_distance(QI)
vector_domain(atom_domain(T))	Vec<T>	l2_distance(QI)

Required features: `contrib`

[make_gaussian in Rust documentation.](#)

Supporting Elements:

- Input Domain: `DI`
- Output Type: `MI`
- Input Metric: `MO`
- Output Measure: `DI::Carrier`

Proof Definition:

[\(Proof Document\)](#)

Value

Measurement

Examples

```
library(opendp)
enable_features("contrib")
gaussian <- make_gaussian(
  atom_domain(.T = f64, nan = FALSE),
  absolute_distance(.T = f64),
  scale = 1.0)
gaussian(arg = 100.0)

# Or, more readably, define the space and then chain:
space <- c(atom_domain(.T = f64, nan = FALSE), absolute_distance(.T = f64))
gaussian <- space |> then_gaussian(scale = 1.0)
gaussian(arg = 100.0)

# Sensitivity of this measurement:
gaussian(d_in = 1)
gaussian(d_in = 2)
gaussian(d_in = 4)

# Typically will be used with vectors rather than individual numbers:
space <- c(vector_domain(atom_domain(.T = i32)), l2_distance(.T = i32))
gaussian <- space |> then_gaussian(scale = 1.0)
gaussian(arg = c(10L, 20L, 30L))
```

```
make_gaussian_threshold
      gaussian threshold constructor
```

Description

Make a Measurement that uses propose-test-release to release a hashmap of counts.

Usage

```
make_gaussian_threshold(
  input_domain,
  input_metric,
  scale,
  threshold,
  k = NULL,
  .MO = "Approximate<ZeroConcentratedDivergence>"
)
```

Arguments

<code>input_domain</code>	Domain of the input.
<code>input_metric</code>	Metric for the input domain.
<code>scale</code>	Noise scale parameter for the laplace distribution. <code>scale == standard_deviation / sqrt(2)</code> .
<code>threshold</code>	Exclude pairs with values whose distance from zero exceeds this value.
<code>k</code>	The noise granularity in terms of 2^k .
<code>.M0</code>	Output Measure.

Details

This function takes a noise granularity in terms of 2^k . Larger granularities are more computationally efficient, but have a looser privacy map. If `k` is not set, `k` defaults to the smallest granularity.

Required features: `contrib`

[make_gaussian_threshold in Rust documentation.](#)

Citations:

- [Rogers23 A Unifying Privacy Analysis Framework for Unknown Domain Algorithms in Differential Privacy](#)
- [CKS20 The Discrete Gaussian for Differential Privacy](#)

Supporting Elements:

- Input Domain: `DI`
- Output Type: `MI`
- Input Metric: `M0`
- Output Measure: `DI::Carrier`

Proof Definition:

[\(Proof Document\)](#)

Value

Measurement

make_geometric	<i>geometric constructor</i>
----------------	------------------------------

Description

Equivalent to `make_laplace` but restricted to an integer support. Can specify `bounds` to run the algorithm in near constant-time.

Usage

```
make_geometric(
  input_domain,
  input_metric,
  scale,
  bounds = NULL,
  .M0 = "MaxDivergence"
)
```

Arguments

<code>input_domain</code>	Domain of the data type to be released.
<code>input_metric</code>	Metric of the data type to be released.
<code>scale</code>	Noise scale parameter for the distribution. <code>scale == standard_deviation / sqrt(2)</code> .
<code>bounds</code>	Set bounds on the count to make the algorithm run in constant-time.
<code>.M0</code>	Measure used to quantify privacy loss. Valid values are just <code>MaxDivergence</code>

Details

Required features: `contrib`

[make_geometric in Rust documentation.](#)

Citations:

- [GRS12 Universally Utility-Maximizing Privacy Mechanisms](#)

Supporting Elements:

- Input Domain: `DI`
- Output Type: `MI`
- Input Metric: `M0`
- Output Measure: `DI::Carrier`

Proof Definition:

[\(Proof Document\)](#)

Value

Measurement

make_identity	<i>identity constructor</i>
---------------	-----------------------------

Description

Make a Transformation representing the identity function.

Usage

```
make_identity(domain, metric)
```

Arguments

domain	Domain of input data
metric	Metric on input domain

Details

WARNING: In Python, this function does not ensure that the domain and metric form a valid metric space. However, if the domain and metric do not form a valid metric space, then the resulting Transformation won't be chainable with any valid Transformation, so it cannot be used to introduce an invalid metric space into a chain of valid Transformations.

Required features: `contrib`, `honest-but-curious`

[make_identity in Rust documentation.](#)

Why honest-but-curious?:

For the result to be a valid transformation, the `input_domain` and `input_metric` pairing must form a valid metric space. For instance, the symmetric distance metric and atom domain do not form a valid metric space, because the metric cannot be used to measure distances between any two elements of an atom domain. Whereas, the symmetric distance metric and vector domain, or absolute distance metric and atom domain on a scalar type, both form valid metric spaces.

Supporting Elements:

- Input Domain: D
- Output Domain: M
- Input Metric: D
- Output Metric: M

Value

Transformation

`make_impute_constant` *impute constant constructor*

Description

Make a Transformation that replaces null/None data with `constant`.

Usage

```
make_impute_constant(input_domain, input_metric, constant)
```

Arguments

<code>input_domain</code>	Domain of the input data. See table above.
<code>input_metric</code>	Metric of the input data. A dataset metric.
<code>constant</code>	Value to replace nulls with.

Details

If chaining after a `make_cast`, the input type is `Option<Vec<TA>>`. If chaining after a `make_cast_inherent`, the input type is `Vec<TA>`, where TA may take on float NaNs.

<code>input_domain</code>	Input Data Type
<code>vector_domain(option_domain(atom_domain(TA)))</code>	<code>Vec<Option<TA>></code>
<code>vector_domain(atom_domain(TA))</code>	<code>Vec<TA></code>

Required features: `contrib`

[make_impute_constant in Rust documentation.](#)

Supporting Elements:

- Input Domain: `VectorDomain<DIA>`
- Output Domain: `M`
- Input Metric: `VectorDomain<AtomDomain<DIA::Imputed>>`
- Output Metric: `M`

Value

Transformation

make_impute_uniform_float	<i>impute uniform float constructor</i>
---------------------------	---

Description

Make a Transformation that replaces NaN values in `Vec<TA>` with uniformly distributed floats within `bounds`.

Usage

```
make_impute_uniform_float(input_domain, input_metric, bounds)
```

Arguments

<code>input_domain</code>	Domain of the input.
<code>input_metric</code>	Metric of the input.
<code>bounds</code>	Tuple of inclusive lower and upper bounds.

Details

Required features: `contrib`

[make_impute_uniform_float in Rust documentation.](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<TA>>`
- Output Domain: `M`
- Input Metric: `VectorDomain<AtomDomain<TA>>`
- Output Metric: `M`

Value

Transformation

make_index	<i>index constructor</i>
------------	--------------------------

Description

Make a transformation that treats each element as an index into a vector of categories.

Usage

```
make_index(input_domain, input_metric, categories, null, .TOA = NULL)
```

Arguments

input_domain	The domain of the input vector.
input_metric	The metric of the input vector.
categories	The set of categories to index into.
null	Category to return if the index is out-of-range of the category set.
.TOA	Atomic Output Type. Output data will be <code>Vec<TOA></code> .

Details

Required features: `contrib`
[make_index in Rust documentation.](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<usize>>`
- Output Domain: `M`
- Input Metric: `VectorDomain<AtomDomain<TOA>>`
- Output Metric: `M`

Value

Transformation

make_is_equal	<i>is equal constructor</i>
---------------	-----------------------------

Description

Make a Transformation that checks if each element is equal to `value`.

Usage

```
make_is_equal(input_domain, input_metric, value)
```

Arguments

input_domain	Domain of input data
input_metric	Metric on input domain
value	value to check against

Details

Required features: `contrib`

[make_is_equal in Rust documentation.](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<TIA>>`
- Output Domain: `M`
- Input Metric: `VectorDomain<AtomDomain<bool>>`
- Output Metric: `M`

Proof Definition:

([Proof Document](#))

Value

Transformation

<code>make_is_null</code>	<i>is null constructor</i>
---------------------------	----------------------------

Description

Make a Transformation that checks if each element in a vector is null or nan.

Usage

```
make_is_null(input_domain, input_metric)
```

Arguments

<code>input_domain</code>	Domain of input data
<code>input_metric</code>	Metric on input domain

Details

Required features: `contrib`

[make_is_null in Rust documentation.](#)

Supporting Elements:

- Input Domain: `VectorDomain<DIA>`
- Output Domain: `M`
- Input Metric: `VectorDomain<AtomDomain<bool>>`
- Output Metric: `M`

Value

Transformation

make_laplace	<i>laplace constructor</i>
--------------	----------------------------

Description

Make a Measurement that adds noise from the Laplace(scale) distribution to the input.

Usage

```
make_laplace(
  input_domain,
  input_metric,
  scale,
  k = NULL,
  .MO = "MaxDivergence"
)
```

Arguments

<code>input_domain</code>	Domain of the data type to be released.
<code>input_metric</code>	Metric of the data type to be released.
<code>scale</code>	Noise scale parameter for the Laplace distribution. <code>scale == standard_deviation / sqrt(2)</code> .
<code>k</code>	The noise granularity in terms of 2^k , only valid for domains over floats.
<code>.MO</code>	Measure used to quantify privacy loss. Valid values are just <code>MaxDivergence</code>

Details

Valid inputs for `input_domain` and `input_metric` are:

<code>input_domain</code>	input type	<code>input_metric</code>
<code>atom_domain(T)</code> (default)	<code>T</code>	<code>absolute_distance(T)</code>
<code>vector_domain(atom_domain(T))</code>	<code>Vec<T></code>	<code>l1_distance(T)</code>

Internally, all sampling is done using the discrete Laplace distribution.

Required features: `contrib`

[make_laplace in Rust documentation.](#)

Citations:

- [GRS12 Universally Utility-Maximizing Privacy Mechanisms](#)
- [CKS20 The Discrete Gaussian for Differential Privacy](#)

Supporting Elements:

- Input Domain: `DI`

- Output Type: MI
- Input Metric: MO
- Output Measure: `DI::Carrier`

Proof Definition:

([Proof Document](#))

Value

Measurement

```
make_laplace_threshold
    laplace threshold constructor
```

Description

Make a Measurement that uses propose-test-release to release a hashmap of counts.

Usage

```
make_laplace_threshold(
    input_domain,
    input_metric,
    scale,
    threshold,
    k = NULL,
    .MO = "Approximate<MaxDivergence>"
)
```

Arguments

<code>input_domain</code>	Domain of the input.
<code>input_metric</code>	Metric for the input domain.
<code>scale</code>	Noise scale parameter for the laplace distribution. <code>scale == standard_deviation / sqrt(2)</code> .
<code>threshold</code>	Exclude counts that are less than this minimum value.
<code>k</code>	The noise granularity in terms of 2^k .
<code>.MO</code>	Output Measure.

Details

This function takes a noise granularity in terms of 2^k . Larger granularities are more computationally efficient, but have a looser privacy map. If k is not set, k defaults to the smallest granularity.

Required features: `contrib`

[make_laplace_threshold](#) in Rust documentation.

Citations:

- [Rogers23 A Unifying Privacy Analysis Framework for Unknown Domain Algorithms in Differential Privacy](#)
- [CKS20 The Discrete Gaussian for Differential Privacy](#)

Supporting Elements:

- Input Domain: `DI`
- Output Type: `MI`
- Input Metric: `M0`
- Output Measure: `DI::Carrier`

Proof Definition:

[\(Proof Document\)](#)

Value

Measurement

`make_lipschitz_float_mul`
lipschitz float mul constructor

Description

Make a transformation that multiplies an aggregate by a constant.

Usage

```
make_lipschitz_float_mul(input_domain, input_metric, constant, bounds)
```

Arguments

<code>input_domain</code>	The domain of the input.
<code>input_metric</code>	The metric of the input.
<code>constant</code>	The constant to multiply aggregates by.
<code>bounds</code>	Tuple of inclusive lower and upper bounds.

Details

The bounds clamp the input, in order to bound the increase in sensitivity from float rounding.

Required features: `contrib`

[make_lipschitz_float_mul](#) in Rust documentation.

Supporting Elements:

- Input Domain: `AtomDomain<TA>`
- Output Domain: `AbsoluteDistance<TA>`
- Input Metric: `AtomDomain<TA>`
- Output Metric: `AbsoluteDistance<TA>`

Value

Transformation

<code>make_mean</code>	<i>mean constructor</i>
------------------------	-------------------------

Description

Make a Transformation that computes the mean of bounded data.

Usage

```
make_mean(input_domain, input_metric)
```

Arguments

`input_domain` Domain of input data
`input_metric` Metric on input domain

Details

This uses a restricted-sensitivity proof that takes advantage of known dataset size. Use `make_clamp` to bound data and `make_resize` to establish dataset size.

Required features: `contrib`

[make_mean](#) in Rust documentation.

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<T>>`
- Output Domain: `MI`
- Input Metric: `AtomDomain<T>`
- Output Metric: `AbsoluteDistance<T>`

Value

Transformation

make_metric_bounded	<i>metric bounded constructor</i>
---------------------	-----------------------------------

Description

Make a Transformation that converts the unbounded dataset metric `MI` to the respective bounded dataset metric with a no-op.

Usage

```
make_metric_bounded(input_domain, input_metric)
```

Arguments

<code>input_domain</code>	Domain of input data
<code>input_metric</code>	Metric on input domain

Details

The constructor enforces that the input domain has known size, because it must have known size to be valid under a bounded dataset metric.

<code>MI</code>	<code>MI::BoundedMetric</code>
<code>SymmetricDistance</code>	<code>ChangeOneDistance</code>
<code>InsertDeleteDistance</code>	<code>HammingDistance</code>

Required features: `contrib`
[make_metric_bounded in Rust documentation.](#)

Supporting Elements:

- Input Domain: `D`
- Output Domain: `MI`
- Input Metric: `D`
- Output Metric: `MI::BoundedMetric`

Value

Transformation

```
make_metric_unbounded
```

metric unbounded constructor

Description

Make a Transformation that converts the bounded dataset metric MI to the respective unbounded dataset metric with a no-op.

Usage

```
make_metric_unbounded(input_domain, input_metric)
```

Arguments

<code>input_domain</code>	Domain of input data
<code>input_metric</code>	Metric on input domain

Details

MI	MI::UnboundedMetric
ChangeOneDistance	SymmetricDistance
HammingDistance	InsertDeleteDistance

Required features: `contrib`

[make_metric_unbounded in Rust documentation.](#)

Supporting Elements:

- Input Domain: D
- Output Domain: MI
- Input Metric: D
- Output Metric: MI::UnboundedMetric

Value

Transformation

make_noise	noise constructor
------------	-------------------

Description

Make a Measurement that adds noise from the appropriate distribution to the input.

Usage

```
make_noise(input_domain, input_metric, output_measure, scale, k = NULL)
```

Arguments

input_domain	Domain of the data type to be released.
input_metric	Metric of the data type to be released.
output_measure	Privacy measure. Either <code>MaxDivergence</code> or <code>ZeroConcentratedDivergence</code> .
scale	Noise scale parameter.
k	The noise granularity in terms of 2^k .

Details

Valid inputs for `input_domain` and `input_metric` are:

input_domain	input type	input_metric
<code>atom_domain(T)</code>	<code>T</code>	<code>absolute_distance(QI)</code>
<code>vector_domain(atom_domain(T))</code>	<code>Vec<T></code>	<code>l2_distance(QI)</code>

Required features: `contrib`

[make_noise in Rust documentation.](#)

Supporting Elements:

- Input Domain: `DI`
- Output Type: `MI`
- Input Metric: `MO`
- Output Measure: `DI::Carrier`

Value

Measurement

`make_noise_threshold` *noise threshold constructor*

Description

Make a Measurement that uses propose-test-release to release a hashmap of counts.

Usage

```
make_noise_threshold(
    input_domain,
    input_metric,
    output_measure,
    scale,
    threshold,
    k = NULL
)
```

Arguments

<code>input_domain</code>	Domain of the input.
<code>input_metric</code>	Metric for the input domain.
<code>output_measure</code>	Privacy measure. Either <code>MaxDivergence</code> or <code>ZeroConcentratedDivergence</code> .
<code>scale</code>	Noise scale parameter for the laplace distribution. <code>scale == standard_deviation / sqrt(2)</code> .
<code>threshold</code>	Exclude counts that are less than this minimum value.
<code>k</code>	The noise granularity in terms of 2^k .

Details

This function takes a noise granularity in terms of 2^k . Larger granularities are more computationally efficient, but have a looser privacy map. If `k` is not set, `k` defaults to the smallest granularity.

Required features: `contrib`

[make_noise_threshold](#) in Rust documentation.

Citations:

- [Rogers23 A Unifying Privacy Analysis Framework for Unknown Domain Algorithms in Differential Privacy](#)
- [CKS20 The Discrete Gaussian for Differential Privacy](#)

Supporting Elements:

- Input Domain: `DI`
- Output Type: `MI`
- Input Metric: `Approximate<MO>`
- Output Measure: `DI::Carrier`

Value

Measurement

make_noisy_max	<i>noisy max constructor</i>
----------------	------------------------------

Description

Make a Measurement that takes a vector of scores and privately selects the index of the highest score.

Usage

```
make_noisy_max(
    input_domain,
    input_metric,
    output_measure,
    scale,
    negate = FALSE
)
```

Arguments

input_domain	Domain of the input vector. Must be a non-nullable <code>VectorDomain</code>
input_metric	Metric on the input domain. Must be <code>LInfDistance</code>
output_measure	One of <code>MaxDivergence</code> , <code>ZeroConcentratedDivergence</code>
scale	Scale for the noise distribution
negate	Set to true to return min

Details

Required features: `contrib`

[make_noisy_max](#) in Rust documentation.

Citations:

- [MS20 Permute-and-Flip: A New Mechanism for Differentially Private Selection](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<TIA>>`
- Output Type: `LInfDistance<TIA>`
- Input Metric: `M0`
- Output Measure: `usize`

Value

Measurement

make_noisy_top_k	<i>noisy top k constructor</i>
------------------	--------------------------------

Description

Make a Measurement that takes a vector of scores and privately selects the index of the highest score.

Usage

```
make_noisy_top_k(
  input_domain,
  input_metric,
  output_measure,
  k,
  scale,
  negate = FALSE
)
```

Arguments

input_domain	Domain of the input vector. Must be a non-nullable VectorDomain.
input_metric	Metric on the input domain. Must be LInfDistance
output_measure	One of MaxDivergence or ZeroConcentratedDivergence
k	Number of indices to select.
scale	Scale for the noise distribution.
negate	Set to true to return bottom k

Details

Required features: `contrib`

[make_noisy_top_k](#) in Rust documentation.

Citations:

- [MS20 Permute-and-Flip: A New Mechanism for Differentially Private Selection](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<TIA>>`
- Output Type: `LInfDistance<TIA>`
- Input Metric: `M0`
- Output Measure: `Vec<usize>`

Proof Definition:

[\(Proof Document\)](#)

Value

Measurement

make_ordered_random	<i>ordered random constructor</i>
---------------------	-----------------------------------

Description

Make a Transformation that converts the unordered dataset metric `SymmetricDistance` to the respective ordered dataset metric `InsertDeleteDistance` by assigning a random permutation.

Usage

```
make_ordered_random(input_domain, input_metric)
```

Arguments

- `input_domain` Domain of input data
- `input_metric` Metric on input domain

Details

MI	<code>MI::OrderedMetric</code>
<code>SymmetricDistance</code>	<code>InsertDeleteDistance</code>
<code>ChangeOneDistance</code>	<code>HammingDistance</code>

Required features: `contrib`
[make_ordered_random in Rust documentation.](#)

Supporting Elements:

- Input Domain: `D`
- Output Domain: `MI`
- Input Metric: `D`
- Output Metric: `MI::OrderedMetric`

Value

Transformation

`make_population_amplification` *population amplification constructor*

Description

Construct an amplified measurement from a `measurement` with privacy amplification by subsampling. This measurement does not perform any sampling. It is useful when you have a dataset on-hand that is a simple random sample from a larger population.

Usage

```
make_population_amplification(measurement, population_size)
```

Arguments

`measurement` the computation to amplify
`population_size` the size of the population from which the input dataset is a simple sample

Details

The DIA, DO, MI and MO between the input measurement and amplified output measurement all match.

Required features: `contrib`, `honest-but-curious`

[make_population_amplification in Rust documentation.](#)

Why honest-but-curious?:

The privacy guarantees are only valid if the input dataset is a simple sample from a population with `population_size` records.

Value

Measurement

`make_privacy_filter` *privacy filter constructor*

Description

Combinator that limits the privacy loss of an odometer.

Usage

```
make_privacy_filter(odometer, d_in, d_out)
```

Arguments

<code>odometer</code>	A privacy odometer
<code>d_in</code>	Upper bound on the distance between adjacent datasets
<code>d_out</code>	Upper bound on the privacy loss

Details

Adjusts the queryable returned by the odometer to reject any query that would increase the total privacy loss above the privacy guarantee of the mechanism.

Required features: `contrib`

[make_privacy_filter](#) in Rust documentation.

Citations:

- [RRUV21 Privacy Odometers and Filters: Pay-as-you-Go Composition](#)
- [HSTVVXZ23 Concurrent Composition for Interactive Differential Privacy with Adaptive Privacy-Loss Parameters](#)

Supporting Elements:

- Input Domain: `DI`
- Output Type: `MI`
- Input Metric: `MO`
- Output Measure: `OdometerQueryable<Q, A, MI::Distance, MO::Distance>`

Proof Definition:

[\(Proof Document\)](#)

Value

Measurement

`make_private_quantile`
private quantile constructor

Description

Make a Measurement that computes the quantile of a dataset.

Usage

```
make_private_quantile(
    input_domain,
    input_metric,
    output_measure,
    candidates,
    alpha,
    scale
)
```

Arguments

<code>input_domain</code>	Uses a tighter sensitivity when the size of vectors in the input domain is known.
<code>input_metric</code>	Either <code>SymmetricDistance</code> or <code>InsertDeleteDistance</code> .
<code>output_measure</code>	Either <code>MaxDivergence</code> or <code>ZeroConcentratedDivergence</code> .
<code>candidates</code>	Potential quantiles to score
<code>alpha</code>	a value in $[0, 1]$. Choose 0.5 for median
<code>scale</code>	the scale of the noise added

Details

Required features: `contrib`

[make_private_quantile in Rust documentation.](#)

Citations:

- [Smith11 Privacy-Preserving Statistical Estimation with Optimal Convergence Rates](#)
- [MS20 Permute-and-Flip: A New Mechanism for Differentially Private Selection](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<T>>`
- Output Type: `MI`
- Input Metric: `M0`
- Output Measure: `T`

Value

Measurement

`make_pureDP_to_zCDP` *pureDP to zCDP constructor*

Description

Constructs a new output measurement where the output measure is casted from `MaxDivergence` to `ZeroConcentratedDivergence`.

Usage

```
make_pureDP_to_zCDP(measurement)
```

Arguments

`measurement` a measurement with a privacy measure to be casted

Details

Required features: `contrib`

[make_pureDP_to_zCDP in Rust documentation.](#)

Citations:

- [BS16 Concentrated Differential Privacy: Simplifications, Extensions, and Lower Bounds](#)

Value

Measurement

`make_quantile_score_candidates`
quantile score candidates constructor

Description

Makes a Transformation that scores how similar each candidate is to the given `alpha`-quantile on the input dataset.

Usage

```
make_quantile_score_candidates(input_domain, input_metric, candidates, alpha)
```

Arguments

<code>input_domain</code>	Uses a smaller sensitivity when the size of vectors in the input domain is known.
<code>input_metric</code>	Either <code>SymmetricDistance</code> or <code>InsertDeleteDistance</code> .
<code>candidates</code>	Potential quantiles to score
<code>alpha</code>	a value in $[0, 1]$. Choose 0.5 for median

Details

Required features: `contrib`

[make_quantile_score_candidates](#) in Rust documentation.

Citations:

- [Smith11 Privacy-Preserving Statistical Estimation with Optimal Convergence Rates](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<TIA>>`
- Output Domain: `MI`
- Input Metric: `VectorDomain<AtomDomain<u64>>`
- Output Metric: `LInfDistance<u64>`

Proof Definition:

[\(Proof Document\)](#)

Value

Transformation

`make_quantiles_from_counts`
quantiles from counts constructor

Description

Postprocess a noisy array of summary counts into quantiles.

Usage

```
make_quantiles_from_counts(
    bin_edges,
    alphas,
    interpolation = "linear",
    .TA = NULL,
    .F = "float"
)
```

Arguments

<code>bin_edges</code>	The edges that the input data was binned into before counting.
<code>alphas</code>	Return all specified alpha -quantiles.
<code>interpolation</code>	Must be one of linear or nearest
<code>.TA</code>	Atomic Type of the bin edges and data.
<code>.F</code>	Float type of the alpha argument. One of f32 or f64

Details

Required features: `contrib`

[make_quantiles_from_counts](#) in Rust documentation.

Supporting Elements:

- Input Type: `Vec<TA>`
- Output Type: `Vec<TA>`

Value

Function

`make_randomized_response`

randomized response constructor

Description

Make a Measurement that implements randomized response on a categorical value.

Usage

```
make_randomized_response(categories, prob, .T = NULL)
```

Arguments

<code>categories</code>	Set of valid outcomes
<code>prob</code>	Probability of returning the correct answer. Must be in <code>[1/num_categories, 1]</code>
<code>.T</code>	Data type of a category.

Details

Required features: `contrib`

[make_randomized_response](#) in Rust documentation.

Citations:

- [Warner65 Randomized Response: A Survey Technique for Eliminating Evasive Answer Bias](#)

Supporting Elements:

- Input Domain: `AtomDomain<T>`
- Output Type: `DiscreteDistance`
- Input Metric: `MaxDivergence`
- Output Measure: `T`

Proof Definition:

[\(Proof Document\)](#)

Value

Measurement

make_randomized_response_bitvec

randomized response bitvec constructor

Description

Make a Measurement that implements randomized response on a bit vector.

Usage

```
make_randomized_response_bitvec(
    input_domain,
    input_metric,
    f,
    constant_time = FALSE
)
```

Arguments

<code>input_domain</code>	BitVectorDomain with max_weight
<code>input_metric</code>	DiscreteDistance
<code>f</code>	Per-bit flipping probability. Must be in (0, 1].
<code>constant_time</code>	Whether to run the Bernoulli samplers in constant time, this is likely to be extremely slow.

Details

This primitive can be useful for implementing RAPPOR.

Required features: `contrib`

[make_randomized_response_bitvec](#) in Rust documentation.

Citations:

- [RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response](#)

Supporting Elements:

- Input Domain: `BitVectorDomain`
- Output Type: `DiscreteDistance`
- Input Metric: `MaxDivergence`
- Output Measure: `BitVector`

Proof Definition:

[\(Proof Document\)](#)

Value

Measurement

make_randomized_response_bool*randomized response bool constructor*

Description

Make a Measurement that implements randomized response on a boolean value.

Usage

```
make_randomized_response_bool(prob, constant_time = FALSE)
```

Arguments

prob Probability of returning the correct answer. Must be in [0.5, 1]
constant_time Set to true to enable constant time. Slower.

Details

Required features: `contrib`

[make_randomized_response_bool](#) in Rust documentation.

Citations:

- [Warner65 Randomized Response: A Survey Technique for Eliminating Evasive Answer Bias](#)

Supporting Elements:

- Input Domain: `AtomDomain<bool>`
- Output Type: `DiscreteDistance`
- Input Metric: `MaxDivergence`
- Output Measure: `bool`

Proof Definition:

[\(Proof Document\)](#)

Value

Measurement

```
make_report_noisy_max_gumbel
    report noisy max gumbel constructor
```

Description

Make a Measurement that takes a vector of scores and privately selects the index of the highest score.

Usage

```
make_report_noisy_max_gumbel(
    input_domain,
    input_metric,
    scale,
    optimize = "max"
)
```

Arguments

<code>input_domain</code>	Domain of the input vector. Must be a non-nullable <code>VectorDomain</code>
<code>input_metric</code>	Metric on the input domain. Must be <code>LInfDistance</code>
<code>scale</code>	Scale for the noise distribution
<code>optimize</code>	Set to "min" to report noisy min

Details

Required features: `contrib`

[make_report_noisy_max_gumbel in Rust documentation.](#)

Citations:

- [MS20 Permute-and-Flip: A New Mechanism for Differentially Private Selection](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<TIA>>`
- Output Type: `LInfDistance<TIA>`
- Input Metric: `MaxDivergence`
- Output Measure: `usize`

Value

Measurement

make_resize	resize constructor
-------------	--------------------

Description

Make a Transformation that either truncates or imputes records with `constant` to match a provided `size`.

Usage

```
make_resize(  
    input_domain,  
    input_metric,  
    size,  
    constant,  
    .M0 = "SymmetricDistance"  
)
```

Arguments

<code>input_domain</code>	Domain of input data.
<code>input_metric</code>	Metric of input data.
<code>size</code>	Number of records in output data.
<code>constant</code>	Value to impute with.
<code>.M0</code>	Output Metric. One of <code>InsertDeleteDistance</code> or <code>SymmetricDistance</code>

Details

Required features: `contrib`

[make_resize in Rust documentation.](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<TA>>`
- Output Domain: `MI`
- Input Metric: `VectorDomain<AtomDomain<TA>>`
- Output Metric: `M0`

Value

A vector of the same type `TA`, but with the provided `size`.

make_select_column	<i>select column constructor</i>
--------------------	----------------------------------

Description

Make a Transformation that retrieves the column **key** from a dataframe as `Vec<TOA>`.

Usage

```
make_select_column(key, .TOA, .K = NULL)
```

Arguments

<code>key</code>	categorical/hashable data type of the key/column name
<code>.TOA</code>	Atomic Output Type to downcast vector to
<code>.K</code>	data type of key

Details

Required features: `contrib`

[make_select_column in Rust documentation.](#)

Supporting Elements:

- Input Domain: `DataFrameDomain<K>`
- Output Domain: `SymmetricDistance`
- Input Metric: `VectorDomain<AtomDomain<TOA>>`
- Output Metric: `SymmetricDistance`

Value

Transformation

make_select_private_candidate	<i>select private candidate constructor</i>
-------------------------------	---

Description

Select a private candidate whose score is above a threshold.

Usage

```
make_select_private_candidate(measurement, stop_probability, threshold)
```

Arguments

measurement	A measurement that releases a 2-tuple of (score, candidate)
stop_probability	The probability of stopping early at any iteration.
threshold	The threshold score. Return immediately if the score is above this threshold.

Details

Given **measurement** that satisfies **-DP**, returns new measurement **M'** that satisfies **2-DP**. **M'** releases the first invocation of **measurement** whose score is above **threshold**.

Each time a score is below **threshold** the algorithm may terminate with probability **stop_probability** and return nothing.

measurement should make releases in the form of (score, candidate). If you are writing a custom scorer measurement in Python, specify the output type as `T0=(float, "ExtrinsicObject")`. This ensures that the float value is accessible to the algorithm. The candidate, left as arbitrary Python data, is held behind the `ExtrinsicObject`.

Algorithm 1 in [Private selection from private candidates](#) (Liu and Talwar, STOC 2019).

Required features: **contrib**

[make_select_private_candidate](#) in Rust documentation.

Supporting Elements:

- Input Domain: **DI**
- Output Type: **MI**
- Input Metric: **MaxDivergence**
- Output Measure: **Option<(f64, T0)>**

Proof Definition:

[\(Proof Document\)](#)

Value

A measurement that returns a release from **measurement** whose score is greater than **threshold**, or none.

make_sequential_composition

sequential composition constructor

Description

Construct a Measurement that when invoked, returns a queryable that interactively composes measurements.

Usage

```
make_sequential_composition(
    input_domain,
    input_metric,
    output_measure,
    d_in,
    d_mids
)
```

Arguments

<code>input_domain</code>	indicates the space of valid input datasets
<code>input_metric</code>	how distances are measured between members of the input domain
<code>output_measure</code>	how privacy is measured
<code>d_in</code>	maximum distance between adjacent input datasets
<code>d_mids</code>	maximum privacy expenditure of each query

Details

Required features: `contrib`

[make_sequential_composition in Rust documentation.](#)

Citations:

- [LW22 Composition Theorems for Interactive Differential Privacy](#)
- [VW21 Concurrent Composition of Differential Privacy](#)
- [HSTVVXZ23 Concurrent Composition for Interactive Differential Privacy with Adaptive Privacy-Loss Parameters](#)

Composition Properties

- sequential: all measurements are applied to the same dataset
- basic: the composition is the linear sum of the privacy usage of each query
- interactive: mechanisms can be specified based on answers to previous queries
- compositor: all privacy parameters specified up-front

If the privacy measure supports concurrency, this compositor allows you to spawn multiple interactive mechanisms and interleave your queries amongst them.

Supporting Elements:

- Input Domain: `DI`
- Output Type: `MI`
- Input Metric: `MO`
- Output Measure: `Queryable<Measurement<DI, MI, MO, TO>, TO>`

Value

Measurement

```
make_sized_bounded_float_checked_sum
```

sized bounded float checked sum constructor

Description

Make a Transformation that computes the sum of bounded floats with known dataset size.

Usage

```
make_sized_bounded_float_checked_sum(size, bounds, .S = "Pairwise<.T>")
```

Arguments

<code>size</code>	Number of records in input data.
<code>bounds</code>	Tuple of lower and upper bounds for data in the input domain.
<code>.S</code>	Summation algorithm to use over some data type T (T is shorthand for <code>S::Item</code>)

Details

This uses a restricted-sensitivity proof that takes advantage of known dataset size for better utility.

S (summation algorithm)	input type
<code>Sequential<S::Item></code>	<code>Vec<S::Item></code>
<code>Pairwise<S::Item></code>	<code>Vec<S::Item></code>

`S::Item` is the type of all of the following: each bound, each element in the input data, the output data, and the output sensitivity.

For example, to construct a transformation that pairwise-sums `f32` half-precision floats, set `S` to `Pairwise<f32>`.

Required features: `contrib`

[make_sized_bounded_float_checked_sum in Rust documentation.](#)

Citations:

- [CSVW22 Widespread Underestimation of Sensitivity...](#)
- [DMNS06 Calibrating Noise to Sensitivity in Private Data Analysis](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<S::Item>>`
- Output Domain: `SymmetricDistance`
- Input Metric: `AtomDomain<S::Item>`
- Output Metric: `AbsoluteDistance<S::Item>`

Value

Transformation

make_sized_bounded_float_ordered_sum
<i>sized bounded float ordered sum constructor</i>

Description

Make a Transformation that computes the sum of bounded floats with known ordering and dataset size.

Usage

```
make_sized_bounded_float_ordered_sum(size, bounds, .S = "Pairwise<T>")
```

Arguments

size	Number of records in input data.
bounds	Tuple of lower and upper bounds for data in the input domain.
.S	Summation algorithm to use over some data type T (T is shorthand for S::Item)

Details

Only useful when `make_bounded_float_checked_sum` returns an error due to potential for overflow. This uses a restricted-sensitivity proof that takes advantage of known dataset size for better utility. You may need to use `make_ordered_random` to impose an ordering on the data.

S (summation algorithm)	input type
Sequential<S::Item>	Vec<S::Item>
Pairwise<S::Item>	Vec<S::Item>

S::Item is the type of all of the following: each bound, each element in the input data, the output data, and the output sensitivity.

For example, to construct a transformation that pairwise-sums f32 half-precision floats, set S to Pairwise<f32>.

Required features: contrib

[make_sized_bounded_float_ordered_sum in Rust documentation.](#)

Citations:

- [CSVW22 Widespread Underestimation of Sensitivity...](#)
- [DMNS06 Calibrating Noise to Sensitivity in Private Data Analysis](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<S::Item>>`
- Output Domain: `InsertDeleteDistance`
- Input Metric: `AtomDomain<S::Item>`
- Output Metric: `AbsoluteDistance<S::Item>`

Value

Transformation

`make_sized_bounded_int_checked_sum`
*sized bounded int checked sum constructor***Description**

Make a Transformation that computes the sum of bounded ints. The effective range is reduced, as $(\text{bounds} * \text{size})$ must not overflow.

Usage

```
make_sized_bounded_int_checked_sum(size, bounds, .T = NULL)
```

Arguments

<code>size</code>	Number of records in input data.
<code>bounds</code>	Tuple of lower and upper bounds for data in the input domain.
<code>.T</code>	Atomic Input Type and Output Type

Details

Required features: `contrib`

[make_sized_bounded_int_checked_sum](#) in Rust documentation.

Citations:

- [CSVW22 Widespread Underestimation of Sensitivity...](#)
- [DMNS06 Calibrating Noise to Sensitivity in Private Data Analysis](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<T>>`
- Output Domain: `SymmetricDistance`
- Input Metric: `AtomDomain<T>`
- Output Metric: `AbsoluteDistance<T>`

Value

Transformation

```
make_sized_bounded_int_monotonic_sum
```

sized bounded int monotonic sum constructor

Description

Make a Transformation that computes the sum of bounded ints, where all values share the same sign.

Usage

```
make_sized_bounded_int_monotonic_sum(size, bounds, .T = NULL)
```

Arguments

<code>size</code>	Number of records in input data.
<code>bounds</code>	Tuple of lower and upper bounds for data in the input domain.
<code>.T</code>	Atomic Input Type and Output Type

Details

Required features: `contrib`

[make_sized_bounded_int_monotonic_sum](#) in Rust documentation.

Citations:

- [CSVW22 Widespread Underestimation of Sensitivity...](#)
- [DMNS06 Calibrating Noise to Sensitivity in Private Data Analysis](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<T>>`
- Output Domain: `SymmetricDistance`
- Input Metric: `AtomDomain<T>`
- Output Metric: `AbsoluteDistance<T>`

Value

Transformation

```
make_sized_bounded_int_ordered_sum
    sized bounded int ordered sum constructor
```

Description

Make a Transformation that computes the sum of bounded ints with known dataset size.

Usage

```
make_sized_bounded_int_ordered_sum(size, bounds, .T = NULL)
```

Arguments

<code>size</code>	Number of records in input data.
<code>bounds</code>	Tuple of lower and upper bounds for data in the input domain.
<code>.T</code>	Atomic Input Type and Output Type

Details

This uses a restricted-sensitivity proof that takes advantage of known dataset size for better utility. You may need to use `make_ordered_random` to impose an ordering on the data.

Required features: `contrib`

[make_sized_bounded_int_ordered_sum](#) in Rust documentation.

Citations:

- [CSVW22 Widespread Underestimation of Sensitivity...](#)
- [DMNS06 Calibrating Noise to Sensitivity in Private Data Analysis](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<T>>`
- Output Domain: `InsertDeleteDistance`
- Input Metric: `AtomDomain<T>`
- Output Metric: `AbsoluteDistance<T>`

Value

Transformation

```
make_sized_bounded_int_split_sum
    sized bounded int split sum constructor
```

Description

Make a Transformation that computes the sum of bounded ints with known dataset size.

Usage

```
make_sized_bounded_int_split_sum(size, bounds, .T = NULL)
```

Arguments

<code>size</code>	Number of records in input data.
<code>bounds</code>	Tuple of lower and upper bounds for data in the input domain.
<code>.T</code>	Atomic Input Type and Output Type

Details

This uses a restricted-sensitivity proof that takes advantage of known dataset size for better utility. Adds the saturating sum of the positives to the saturating sum of the negatives.

Required features: `contrib`

[make_sized_bounded_int_split_sum](#) in Rust documentation.

Citations:

- [CSVW22 Widespread Underestimation of Sensitivity...](#)
- [DMNS06 Calibrating Noise to Sensitivity in Private Data Analysis](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<T>>`
- Output Domain: `SymmetricDistance`
- Input Metric: `AtomDomain<T>`
- Output Metric: `AbsoluteDistance<T>`

Value

Transformation

`make_split_dataframe` *split dataframe constructor*

Description

Make a Transformation that splits each record in a String into a `Vec<Vec<String>>`, and loads the resulting table into a dataframe keyed by `col_names`.

Usage

```
make_split_dataframe(separator, col_names, .K = NULL)
```

Arguments

<code>separator</code>	The token(s) that separate entries in each record.
<code>col_names</code>	Column names for each record entry.
<code>.K</code>	categorical/hashable data type of column names

Details

Required features: `contrib`

[make_split_dataframe in Rust documentation.](#)

Supporting Elements:

- Input Domain: `AtomDomain<String>`
- Output Domain: `SymmetricDistance`
- Input Metric: `DataFrameDomain<K>`
- Output Metric: `SymmetricDistance`

Value

Transformation

`make_split_lines` *split lines constructor*

Description

Make a Transformation that takes a string and splits it into a `Vec<String>` of its lines.

Usage

```
make_split_lines()
```

Details

Required features: `contrib`

[make_split_lines](#) in Rust documentation.

Supporting Elements:

- Input Domain: `AtomDomain<String>`
- Output Domain: `SymmetricDistance`
- Input Metric: `VectorDomain<AtomDomain<String>>`
- Output Metric: `SymmetricDistance`

Value

Transformation

<code>make_split_records</code>	<i>split records constructor</i>
---------------------------------	----------------------------------

Description

Make a Transformation that splits each record in a `Vec<String>` into a `Vec<Vec<String>>`.

Usage

```
make_split_records(separator)
```

Arguments

`separator` The token(s) that separate entries in each record.

Details

Required features: `contrib`

[make_split_records](#) in Rust documentation.

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<String>>`
- Output Domain: `SymmetricDistance`
- Input Metric: `VectorDomain<VectorDomain<AtomDomain<String>>>`
- Output Metric: `SymmetricDistance`

Value

Transformation

make_subset_by	subset by constructor
----------------	-----------------------

Description

Make a Transformation that subsets a dataframe by a boolean column.

Usage

```
make_subset_by(indicator_column, keep_columns, .TK = NULL)
```

Arguments

indicator_column	name of the boolean column that indicates inclusion in the subset
keep_columns	list of column names to apply subset to
.TK	Type of the column name

Details

Required features: `contrib`

[make_subset_by](#) in [Rust documentation](#).

Supporting Elements:

- Input Domain: `DataFrameDomain<TK>`
- Output Domain: `SymmetricDistance`
- Input Metric: `DataFrameDomain<TK>`
- Output Metric: `SymmetricDistance`

Value

Transformation

make_sum	sum constructor
----------	-----------------

Description

Make a Transformation that computes the sum of bounded data. Use `make_clamp` to bound data.

Usage

```
make_sum(input_domain, input_metric)
```

Arguments

`input_domain` Domain of the input data.
`input_metric` One of `SymmetricDistance` or `InsertDeleteDistance`.

Details

If dataset size is known, uses a restricted-sensitivity proof that takes advantage of known dataset size for better utility.

Required features: `contrib`

[make_sum in Rust documentation.](#)

Citations:

- [CSVW22 Widespread Underestimation of Sensitivity...](#)
- [DMNS06 Calibrating Noise to Sensitivity in Private Data Analysis](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<T>>`
- Output Domain: `MI`
- Input Metric: `AtomDomain<T>`
- Output Metric: `AbsoluteDistance<T>`

Value

Transformation

`make_sum_of_squared_deviations`
sum of squared deviations constructor

Description

Make a Transformation that computes the sum of squared deviations of bounded data.

Usage

```
make_sum_of_squared_deviations(input_domain, input_metric, .S = "Pairwise<T>")
```

Arguments

`input_domain` Domain of input data
`input_metric` Metric on input domain
`.S` Summation algorithm to use on data type T. One of `Sequential<T>` or `Pairwise<T>`.

Details

This uses a restricted-sensitivity proof that takes advantage of known dataset size. Use `make_clamp` to bound data and `make_resize` to establish dataset size.

S (summation algorithm)	input type
Sequential<S::Item>	Vec<S::Item>
Pairwise<S::Item>	Vec<S::Item>

`S::Item` is the type of all of the following: each bound, each element in the input data, the output data, and the output sensitivity.

For example, to construct a transformation that computes the SSD of `f32` half-precision floats, set `S` to `Pairwise<f32>`.

Required features: `contrib`

[make_sum_of_squared_deviations in Rust documentation.](#)

Citations:

- [CSVW22 Widespread Underestimation of Sensitivity...](#)
- [DMNS06 Calibrating Noise to Sensitivity in Private Data Analysis](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<S::Item>>`
- Output Domain: `SymmetricDistance`
- Input Metric: `AtomDomain<S::Item>`
- Output Metric: `AbsoluteDistance<S::Item>`

Value

Transformation

<code>make_unordered</code>	<i>unordered constructor</i>
-----------------------------	------------------------------

Description

Make a Transformation that converts the ordered dataset metric `MI` to the respective ordered dataset metric with a no-op.

Usage

`make_unordered(input_domain, input_metric)`

Arguments

input_domain Domain of input data
input_metric Metric on input domain

Details

MI	MI::UnorderedMetric
InsertDeleteDistance	SymmetricDistance
HammingDistance	ChangeOneDistance

Required features: contrib
[make_unordered in Rust documentation.](#)

Supporting Elements:

- Input Domain: D
- Output Domain: MI
- Input Metric: D
- Output Metric: MI::UnorderedMetric

Value

Transformation

make_user_measurement
<i>user measurement constructor</i>

Description

Construct a Measurement from user-defined callbacks.

Usage

```
make_user_measurement(  
  input_domain,  
  input_metric,  
  output_measure,  
  function_,  
  privacy_map,  
  .TO = "ExtrinsicObject"  
)
```

Arguments

<code>input_domain</code>	A domain describing the set of valid inputs for the function.
<code>input_metric</code>	The metric from which distances between adjacent inputs are measured.
<code>output_measure</code>	The measure from which distances between adjacent output distributions are measured.
<code>function_</code>	A function mapping data from <code>input_domain</code> to a release of type <code>T0</code> .
<code>privacy_map</code>	A function mapping distances from <code>input_metric</code> to <code>output_measure</code> .
<code>.T0</code>	The data type of outputs from the function.

Details

Required features: `contrib`, `honest-but-curious`

Why honest-but-curious?:

This constructor only returns a valid measurement if for every pair of elements x, x' in `input_domain`, and for every pair `(d_in, d_out)`, where `d_in` has the associated type for `input_metric` and `d_out` has the associated type for `output_measure`, if x, x' are `d_in`-close under `input_metric`, `privacy_map(d_in)` does not raise an exception, and `privacy_map(d_in) <= d_out`, then `function(x)`, `function(x')` are `d_out`-close under `output_measure`.

In addition, `function` must not have side-effects, and `privacy_map` must be a pure function.

Supporting Elements:

- Input Domain: `AnyDomain`
- Output Type: `AnyMetric`
- Input Metric: `AnyMeasure`
- Output Measure: `AnyObject`

Value

Measurement

`make_user_transformation`
user transformation constructor

Description

Construct a Transformation from user-defined callbacks.

Usage

```
make_user_transformation(
    input_domain,
    input_metric,
    output_domain,
    output_metric,
    function_,
    stability_map
)
```

Arguments

input_domain A domain describing the set of valid inputs for the function.

input_metric The metric from which distances between adjacent inputs are measured.

output_domain A domain describing the set of valid outputs of the function.

output_metric The metric from which distances between outputs of adjacent inputs are measured.

function_ A function mapping data from **input_domain** to **output_domain**.

stability_map A function mapping distances from **input_metric** to **output_metric**.

Details

Required features: `contrib`, `honest-but-curious`

Why honest-but-curious?:

This constructor only returns a valid transformation if for every pair of elements x, x' in **input_domain**, and for every pair (d_{in}, d_{out}) , where d_{in} has the associated type for **input_metric** and d_{out} has the associated type for **output_metric**, if x, x' are d_{in} -close under **input_metric**, **stability_map**(d_{in}) does not raise an exception, and **stability_map**(d_{in}) $\leq d_{out}$, then **function**(x), **function**(x') are d_{out} -close under **output_metric**.

In addition, for every element x in **input_domain**, **function**(x) is a member of **output_domain** or raises a data-independent runtime exception.

In addition, **function** must not have side-effects, and **stability_map** must be a pure function.

Value

Transformation

make_variance	<i>variance constructor</i>
---------------	-----------------------------

Description

Make a Transformation that computes the variance of bounded data.

Usage

```
make_variance(input_domain, input_metric, ddof = 1L, .S = "Pairwise<T>")
```

Arguments

<code>input_domain</code>	Domain of input data
<code>input_metric</code>	Metric on input domain
<code>ddof</code>	Delta degrees of freedom. Set to 0 if not a sample, 1 for sample estimate.
<code>.S</code>	Summation algorithm to use on data type T. One of <code>Sequential<T></code> or <code>Pairwise<T></code> .

Details

This uses a restricted-sensitivity proof that takes advantage of known dataset size. Use `make_clamp` to bound data and `make_resize` to establish dataset size.

Required features: `contrib`

[make_variance in Rust documentation.](#)

Citations:

- [DHK15 Differential Privacy for Social Science Inference](#)

Supporting Elements:

- Input Domain: `VectorDomain<AtomDomain<S::Item>>`
- Output Domain: `SymmetricDistance`
- Input Metric: `AtomDomain<S::Item>`
- Output Metric: `AbsoluteDistance<S::Item>`

Value

Transformation

<code>make_zCDP_to_approxDP</code>	<i>zCDP to approxDP constructor</i>
------------------------------------	-------------------------------------

Description

Constructs a new output measurement where the output measure is casted from `ZeroConcentratedDivergence` to `SmoothedMaxDivergence`.

Usage

```
make_zCDP_to_approxDP(measurement)
```

Arguments

`measurement` a measurement with a privacy measure to be casted

Details

Required features: `contrib`

[make_zCDP_to_approxDP in Rust documentation.](#)

Value

Measurement

<code>map_domain</code>	<i>Construct an instance of MapDomain.</i>
-------------------------	--

Description

Construct an instance of `MapDomain`.

Usage

```
map_domain(key_domain, value_domain)
```

Arguments

`key_domain` domain of keys in the hashmap
`value_domain` domain of values in the hashmap

Value

Domain

max_divergence	<i>Privacy measure used to define ϵ-pure differential privacy.</i>
----------------	--

Description

In the following proof definition, d corresponds to ϵ when also quantified over all adjacent datasets. That is, ϵ is the greatest possible d over all pairs of adjacent datasets x, x' where $Y \sim M(x)$, $Y' \sim M(x')$. $M(\cdot)$ is a measurement (commonly known as a mechanism). The measurement's input metric defines the notion of adjacency, and the measurement's input domain defines the set of possible datasets.

Usage

```
max_divergence()
```

Details**Proof Definition:**

For any two distributions Y, Y' and any non-negative d , Y, Y' are d -close under the max divergence measure whenever

$$D_{\infty}(Y, Y') = \max_{S \subseteq \text{Supp}(Y)} \left[\ln \frac{\Pr[Y \in S]}{\Pr[Y' \in S]} \right] \leq d.$$

Value

Measure

measure_debug	<i>Debug a measure.</i>
---------------	-------------------------

Description

Debug a measure.

Usage

```
measure_debug(this)
```

Arguments

this	The measure to debug (stringify).
------	-----------------------------------

Value

str

`measure_distance_type`*Get the distance type of a `measure`.*

Description

Get the distance type of a `measure`.

Usage

```
measure_distance_type(this)
```

Arguments

`this` The measure to retrieve the distance type from.

Value

str

`measure_type`*Get the type of a `measure`.*

Description

Get the type of a `measure`.

Usage

```
measure_type(this)
```

Arguments

`this` The measure to retrieve the type from.

Value

str

measurement_check	<i>Check the privacy relation of the <code>measurement</code> at the given <code>d_in</code>, <code>d_out</code></i>
-------------------	--

Description

Check the privacy relation of the `measurement` at the given `d_in`, `d_out`

Usage

```
measurement_check(measurement, distance_in, distance_out)
```

Arguments

<code>measurement</code>	Measurement to check the privacy relation of.
<code>distance_in</code>	undocumented
<code>distance_out</code>	undocumented

Value

True indicates that the relation passed at the given distance.

measurement_function	<i>Get the function from a measurement.</i>
----------------------	---

Description

Get the function from a measurement.

Usage

```
measurement_function(this)
```

Arguments

<code>this</code>	The measurement to retrieve the value from.
-------------------	---

Value

Function

measurement_input_carrier_type*Get the input (carrier) data type of this.*

Description

Get the input (carrier) data type of **this**.

Usage

```
measurement_input_carrier_type(this)
```

Arguments

this The measurement to retrieve the type from.

Value

str

measurement_input_distance_type*Get the input distance type of measurement.*

Description

Get the input distance type of **measurement**.

Usage

```
measurement_input_distance_type(this)
```

Arguments

this The measurement to retrieve the type from.

Value

str

`measurement_input_domain`*Get the input domain from a measurement.*

Description

Get the input domain from a `measurement`.

Usage

```
measurement_input_domain(this)
```

Arguments

<code>this</code>	The measurement to retrieve the value from.
-------------------	---

Value

Domain

`measurement_input_metric`*Get the input domain from a measurement.*

Description

Get the input domain from a `measurement`.

Usage

```
measurement_input_metric(this)
```

Arguments

<code>this</code>	The measurement to retrieve the value from.
-------------------	---

Value

Metric

measurement_invoke	<i>Invoke the measurement with arg. Returns a differentially private release.</i>
--------------------	---

Description

Invoke the `measurement` with `arg`. Returns a differentially private release.

Usage

```
measurement_invoke(this, arg)
```

Arguments

<code>this</code>	Measurement to invoke.
<code>arg</code>	Input data to supply to the measurement. A member of the measurement's input domain.

measurement_map	<i>Use the measurement to map a given d_in to d_out.</i>
-----------------	--

Description

Use the measurement to map a given `d_in` to `d_out`.

Usage

```
measurement_map(measurement, distance_in)
```

Arguments

<code>measurement</code>	Measurement to check the map distances with.
<code>distance_in</code>	Distance in terms of the input metric.

`measurement_output_distance_type`*Get the output distance type of measurement.*

Description

Get the output distance type of `measurement`.

Usage

```
measurement_output_distance_type(this)
```

Arguments

<code>this</code>	The measurement to retrieve the type from.
-------------------	--

Value

str

`measurement_output_measure`*Get the output domain from a measurement.*

Description

Get the output domain from a `measurement`.

Usage

```
measurement_output_measure(this)
```

Arguments

<code>this</code>	The measurement to retrieve the value from.
-------------------	---

Value

Measure

<code>metric_debug</code>	<i>Debug a metric.</i>
---------------------------	------------------------

Description

Debug a `metric`.

Usage

```
metric_debug(this)
```

Arguments

<code>this</code>	The metric to debug (stringify).
-------------------	----------------------------------

Value

str

<code>metric_distance_type</code>	<i>Get the distance type of a metric.</i>
-----------------------------------	---

Description

Get the distance type of a `metric`.

Usage

```
metric_distance_type(this)
```

Arguments

<code>this</code>	The metric to retrieve the distance type from.
-------------------	--

Value

str

metric_type	<i>Get the type of a metric.</i>
-------------	----------------------------------

Description

Get the type of a `metric`.

Usage

```
metric_type(this)
```

Arguments

<code>this</code>	The metric to retrieve the type from.
-------------------	---------------------------------------

Value

str

new_domain	<i>new domain</i>
------------	-------------------

Description

new domain

Usage

```
new_domain(ptr, log)
```

Arguments

<code>ptr</code>	a pointer to a domain
<code>log</code>	call history

<code>new_function</code>	<i>Construct a Function from a user-defined callback. Can be used to build a postprocessor.</i>
---------------------------	---

Description

Required features: `contrib`, `honest-but-curious`

Usage

```
new_function(function_, .T0)
```

Arguments

<code>function_</code>	A function mapping data to a value of type <code>T0</code>
<code>.T0</code>	Output Type

Details

[new_function in Rust documentation.](#)

Why honest-but-curious?:

An OpenDP `function` must satisfy two criteria. These invariants about functions are necessary to show correctness of other algorithms.

First, `function` must not use global state. For instance, a postprocessor that accesses the system clock time can be used to build a measurement that reveals elapsed execution time, which escalates a side-channel vulnerability into a direct vulnerability.

Secondly, `function` must only raise data-independent exceptions. For instance, raising an exception with the value of a DP release will both reveal the DP output and cancel the computation, potentially avoiding privacy accounting.

Value

Function

<code>new_function_internal</code>	<i>new function</i>
------------------------------------	---------------------

Description

`new function`

Usage

```
new_function_internal(ptr, log)
```

Arguments

ptr	a pointer to a function
log	call history

new_hashtab	<i>create an instance of a hashtab from keys and values</i>
-------------	---

Description

create an instance of a hashtab from keys and values

Usage

```
new_hashtab(keys, vals)
```

Arguments

keys	a vector of keys
vals	a vector of values

new_measure	<i>new measure</i>
-------------	--------------------

Description

new measure

Usage

```
new_measure(ptr, log)
```

Arguments

ptr	a pointer to a measure
log	call history

<code>new_measurement</code>	<i>new measurement</i>
------------------------------	------------------------

Description

new measurement

Usage

`new_measurement(ptr, log)`

Arguments

<code>ptr</code>	pointer to the measurement struct
<code>log</code>	call history

<code>new_metric</code>	<i>new metric</i>
-------------------------	-------------------

Description

new metric

Usage

`new_metric(ptr, log)`

Arguments

<code>ptr</code>	a pointer to a metric
<code>log</code>	call history

<code>new_odometer</code>	<i>new odometer</i>
---------------------------	---------------------

Description

new odometer

Usage

`new_odometer(ptr, log)`

Arguments

<code>ptr</code>	pointer to the odometer struct
<code>log</code>	call history

```
new_odometer_queryable_internal
    new odometer queryable
```

Description

new odometer queryable

Usage

```
new_odometer_queryable_internal(ptr)
```

Arguments

ptr a pointer to an odometer queryable

```
new_privacy_profile      Construct a PrivacyProfile from a user-defined callback.
```

Description

Required features: `contrib`, `honest-but-curious`

Usage

```
new_privacy_profile(curve)
```

Arguments

curve A privacy curve mapping epsilon to delta

Details

Why honest-but-curious?:

The privacy profile should implement a well-defined $\delta(\epsilon)$ curve:

- monotonically decreasing
- rejects epsilon values that are less than zero or nan
- returns delta values only within $[0, 1]$

<code>new_privacy_profile_internal</code>
<i>new privacy profile</i>

Description

new privacy profile

Usage

`new_privacy_profile_internal(ptr)`

Arguments

`ptr` a pointer to a privacy profile

<code>new_queryable</code>	<i>Construct a queryable from a user-defined transition function.</i>
----------------------------	---

Description

Required features: `contrib`

Usage

`new_queryable(transition, .Q = "ExtrinsicObject", .A = "ExtrinsicObject")`

Arguments

<code>transition</code>	A transition function taking a reference to self, a query, and an internal/external indicator
<code>.Q</code>	Query Type
<code>.A</code>	Output Type

<code>new_queryable_internal</code>	<i>new queryable</i>
-------------------------------------	----------------------

Description

new queryable

Usage

`new_queryable_internal(ptr)`

Arguments

`ptr` a pointer to a queryable

<code>new_transformation</code>	<i>new transformation</i>
---------------------------------	---------------------------

Description

new transformation

Usage

`new_transformation(ptr, log)`

Arguments

`ptr` pointer to the transformation struct
`log` call history

<code>object_type</code>	<i>Internal function. Retrieve the type descriptor string of an Any-Object.</i>
--------------------------	---

Description

Internal function. Retrieve the type descriptor string of an AnyObject.

Usage

`object_type(this)`

Arguments

`this` A pointer to the AnyObject.

Value

str

<code>odometer_input_carrier_type</code>	<i>Get the input (carrier) data type of <code>this</code>.</i>
--	--

Description

Get the input (carrier) data type of `this`.

Usage

`odometer_input_carrier_type(this)`

Arguments

`this` The odometer to retrieve the type from.

Value

str

`odometer_input_domain`*Get the input domain from a `odometer`.*

Description

Get the input domain from a `odometer`.

Usage

```
odometer_input_domain(this)
```

Arguments

`this` The odometer to retrieve the value from.

Value

Domain

`odometer_input_metric`*Get the input domain from a `odometer`.*

Description

Get the input domain from a `odometer`.

Usage

```
odometer_input_metric(this)
```

Arguments

`this` The odometer to retrieve the value from.

Value

Metric

<code>odometer_invoke</code>	<i>Invoke the <code>odometer</code> with <code>arg</code>. Returns a differentially private release.</i>
------------------------------	--

Description

Invoke the `odometer` with `arg`. Returns a differentially private release.

Usage

```
odometer_invoke(this, arg)
```

Arguments

<code>this</code>	Odometer to invoke.
<code>arg</code>	Input data to supply to the odometer. A member of the odometer's input domain.

<code>odometer_output_measure</code>	<i>Get the output domain from a <code>odometer</code>.</i>
--------------------------------------	--

Description

Get the output domain from a `odometer`.

Usage

```
odometer_output_measure(this)
```

Arguments

<code>this</code>	The odometer to retrieve the value from.
-------------------	--

Value

Measure

`odometer_queryable_invoke`*Eval the odometer queryable with an invoke query.*

Description

Eval the odometer `queryable` with an `invoke query`.

Usage

```
odometer_queryable_invoke(queryable, query)
```

Arguments

<code>queryable</code>	Queryable to eval.
<code>query</code>	Invoke query to supply to the queryable.

`odometer_queryable_invoke_type`*Get the invoke query type of an odometer queryable.*

Description

Get the invoke query type of an odometer `queryable`.

Usage

```
odometer_queryable_invoke_type(this)
```

Arguments

<code>this</code>	The queryable to retrieve the query type from.
-------------------	--

Value

`str`

`odometer_queryable_privacy_loss`*Retrieve the privacy loss of an odometer queryable.*

Description

Retrieve the privacy loss of an odometer `queryable`.

Usage

```
odometer_queryable_privacy_loss(queryable, d_in)
```

Arguments

<code>queryable</code>	Queryable to eval.
<code>d_in</code>	Maximum distance between adjacent inputs in the input domain.

`odometer_queryable_privacy_loss_type`*Get the map query type of an odometer queryable.*

Description

Get the map query type of an odometer `queryable`.

Usage

```
odometer_queryable_privacy_loss_type(this)
```

Arguments

<code>this</code>	The queryable to retrieve the query type from.
-------------------	--

Value

str

opendp*OpenDP R Bindings*

Description

The OpenDP Library is a modular collection of statistical algorithms that adhere to the definition of differential privacy.

Details

More information can be found at the following locations:

- opendp.org
- docs.opendp.org
- [GitHub](https://github.com/opendp/opendp)

Author(s)

Maintainer: Stefano Iacus <siacus@iq.harvard.edu>

Authors:

- The OpenDP Project <contact@opendp.org> (<https://github.com/opendp/opendp/graphs/contributors>)

Other contributors:

- The authors of the dependency Rust crates (see `inst/AUTHORS` file for details) [contributor]

See Also

Useful links:

- <https://opendp.org/>
- <https://github.com/opendp/opendp>
- <https://docs.opendp.org/en/stable/index.html>
- Report bugs at <https://github.com/opendp/opendp/issues>

option_domain	Construct an instance of OptionDomain.
---------------	--

Description

[option_domain](#) in Rust documentation.

Usage

```
option_domain(element_domain, .D = NULL)
```

Arguments

element_domain	undocumented
.D	The type of the inner domain.

Value

OptionDomain

parse_or_infer	Parse a runtime type or infer it from an example
----------------	--

Description

Mirrors Python's public `RuntimeType.parse_or_infer(...)` entrypoint.

Usage

```
parse_or_infer(
    type_name,
    public_example,
    generics = list(),
    allow_extrinsic = FALSE
)
```

Arguments

type_name	A type descriptor to normalize, or NULL.
public_example	A value to infer from when <code>type_name</code> is NULL.
generics	Generic type names to preserve during parsing.
allow_extrinsic	Return <code>ExtrinsicObject</code> for unknown host objects when TRUE.

Value

A normalized `runtime_type`.

`privacy_profile_delta`

Internal function. Use a `PrivacyProfile` to find epsilon at a given epsilon.

Description

Internal function. Use a `PrivacyProfile` to find epsilon at a given `epsilon`.

Usage

```
privacy_profile_delta(curve, epsilon)
```

Arguments

<code>curve</code>	The <code>PrivacyProfile</code> .
<code>epsilon</code>	What to fix epsilon to compute delta.

Value

Delta at a given `epsilon`.

`privacy_profile_epsilon`

Internal function. Use an `PrivacyProfile` to find epsilon at a given delta.

Description

Internal function. Use an `PrivacyProfile` to find epsilon at a given `delta`.

Usage

```
privacy_profile_epsilon(profile, delta)
```

Arguments

<code>profile</code>	The <code>PrivacyProfile</code> .
<code>delta</code>	What to fix delta to compute epsilon.

Value

Epsilon at a given `delta`.

<code>queryable_eval</code>	<i>Eval the <code>queryable</code> with <code>query</code>. Returns a differentially private release.</i>
-----------------------------	---

Description

Eval the `queryable` with `query`. Returns a differentially private release.

Usage

```
queryable_eval(queryable, query)
```

Arguments

<code>queryable</code>	Queryable to eval.
<code>query</code>	The input to the queryable.

<code>queryable_query_type</code>	<i>Get the query type of <code>queryable</code>.</i>
-----------------------------------	--

Description

Get the query type of `queryable`.

Usage

```
queryable_query_type(this)
```

Arguments

<code>this</code>	The queryable to retrieve the query type from.
-------------------	--

Value

str

<code>renyi_divergence</code>	<i>Privacy measure used to define $\epsilon(\alpha)$-Rényi differential privacy.</i>
-------------------------------	---

Description

In the following proof definition, d corresponds to an RDP curve when also quantified over all adjacent datasets. That is, an RDP curve $\epsilon(\alpha)$ is no smaller than $d(\alpha)$ for any possible choices of α , and over all pairs of adjacent datasets x, x' where $Y \sim M(x)$, $Y' \sim M(x')$. $M(\cdot)$ is a measurement (commonly known as a mechanism). The measurement's input metric defines the notion of adjacency, and the measurement's input domain defines the set of possible datasets.

Usage

```
renyi_divergence()
```

Details

Proof Definition:

For any two distributions Y, Y' and any curve d , Y, Y' are d -close under the Rényi divergence measure if, for any given $\alpha \in (1, \infty)$,

$$D_\alpha(Y, Y') = \frac{1}{1-\alpha} \mathbb{E}_{x \sim Y'} \left[\ln \left(\frac{\Pr[Y = x]}{\Pr[Y' = x]} \right)^\alpha \right] \leq d(\alpha)$$

Note that this ϵ and α are not privacy parameters ϵ and α until quantified over all adjacent datasets, as is done in the definition of a measurement.

Value

Measure

<code>rt_infer</code>	<i>Infer a runtime type from a public example</i>
-----------------------	---

Description

Mirrors Python's public `RuntimeType.infer(...)` entrypoint.

Usage

```
rt_infer(public_example, allow_extrinsic = FALSE)
```

Arguments

`public_example`

A value whose runtime type should be inferred.

`allow_extrinsic`

Return `ExtrinsicObject` for unknown host objects when TRUE.

Value

A normalized `runtime_type`.

<code>rt_parse</code>	<i>Parse a runtime type descriptor into a runtime_type object</i>
-----------------------	---

Description

Mirrors Python's public `RuntimeType.parse(...)` entrypoint.

Usage

```
rt_parse(type_name, generics = list())
```

Arguments

<code>type_name</code>	A type descriptor to normalize.
<code>generics</code>	Generic type names to preserve during parsing.

Value

A normalized `runtime_type`.

<code>smoothed_max_divergence</code>	<i>Privacy measure used to define $\epsilon(\delta)$-approximate differential privacy.</i>
--------------------------------------	---

Description

In the following proof definition, d corresponds to a privacy profile when also quantified over all adjacent datasets. That is, a privacy profile $\epsilon(\delta)$ is no smaller than $d(\delta)$ for all possible choices of δ , and over all pairs of adjacent datasets x, x' where $Y \sim M(x)$, $Y' \sim M(x')$. $M(\cdot)$ is a measurement (commonly known as a mechanism). The measurement's input metric defines the notion of adjacency, and the measurement's input domain defines the set of possible datasets.

Usage

```
smoothed_max_divergence()
```

Details

The distance d is of type `PrivacyProfile`, so it can be invoked with an ϵ to retrieve the corresponding δ .

Proof Definition:

For any two distributions Y, Y' and any curve $d(\cdot)$, Y, Y' are d -close under the smoothed max divergence measure whenever, for any choice of non-negative ϵ , and $\delta = d(\epsilon)$,

$$D_{\infty}^{\delta}(Y, Y') = \max_{S \subseteq \text{Supp}(Y)} \left[\ln \frac{\Pr[Y \in S] + \delta}{\Pr[Y' \in S]} \right] \leq \epsilon.$$

Note that ϵ and δ are not privacy parameters ϵ and δ until quantified over all adjacent datasets, as is done in the definition of a measurement.

Value

Measure

String	<i>type signature for a string</i>
--------	------------------------------------

Description

type signature for a string

Usage

String

<code>symmetric_distance</code>	<i>Construct an instance of the <code>SymmetricDistance</code> metric.</i>
---------------------------------	--

Description

Construct an instance of the `SymmetricDistance` metric.

Usage

`symmetric_distance()`

Value

Metric

```
then_adaptive_composition
```

partial adaptive composition constructor

Description

See documentation for [make_adaptive_composition\(\)](#) for details.

Usage

```
then_adaptive_composition(lhs, output_measure, d_in, d_mids)
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>output_measure</code>	how privacy is measured
<code>d_in</code>	maximum distance between adjacent input datasets
<code>d_mids</code>	maximum privacy expenditure of each query

Value

Measurement

```
then_alp_queryable
```

partial alp queryable constructor

Description

See documentation for [make_alp_queryable\(\)](#) for details.

Usage

```
then_alp_queryable(
  lhs,
  scale,
  total_limit,
  value_limit = NULL,
  size_factor = 50L,
  alpha = 4L
)
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>scale</code>	Privacy loss parameter. This is equal to epsilon/sensitivity.
<code>total_limit</code>	Either the true value or an upper bound estimate of the sum of all values in the input.
<code>value_limit</code>	Upper bound on individual values (referred to as <code>max</code>). Entries above <code>max</code> are clamped.
<code>size_factor</code>	Optional multiplier (default of 50) for setting the size of the projection.
<code>alpha</code>	Optional parameter (default of 4) for scaling and determining p in randomized response step.

Value

Measurement

<code>then_approximate</code>	<i>partial approximate constructor</i>
-------------------------------	--

Description

See documentation for [make_approximate\(\)](#) for details.

Usage

```
then_approximate(lhs, measurement)
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>measurement</code>	a measurement with a privacy measure to be casted

Value

Measurement

<code>then_b_ary_tree</code>	<i>partial b ary tree constructor</i>
------------------------------	---------------------------------------

Description

See documentation for [make_b_ary_tree\(\)](#) for details.

Usage

```
then_b_ary_tree(lhs, leaf_count, branching_factor)
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>leaf_count</code>	The number of leaf nodes in the b-ary tree.
<code>branching_factor</code>	The number of children on each branch of the resulting tree. Larger branching factors result in shallower trees.

Value

Transformation

<code>then_basic_composition</code>	<i>partial basic composition constructor</i>
-------------------------------------	--

Description

See documentation for [make_basic_composition\(\)](#) for details.

Usage

```
then_basic_composition(lhs, measurements)
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>measurements</code>	A vector of Measurements to compose.

Value

Measurement

```
then_bounded_float_checked_sum
```

partial bounded float checked sum constructor

Description

See documentation for [make_bounded_float_checked_sum\(\)](#) for details.

Usage

```
then_bounded_float_checked_sum(lhs, size_limit, bounds, .S = "Pairwise<.T>")
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>size_limit</code>	Upper bound on number of records to keep in the input data.
<code>bounds</code>	Tuple of lower and upper bounds for data in the input domain.
<code>.S</code>	Summation algorithm to use over some data type T (T is shorthand for <code>S::Item</code>)

Value

Transformation

```
then_bounded_float_ordered_sum
```

partial bounded float ordered sum constructor

Description

See documentation for [make_bounded_float_ordered_sum\(\)](#) for details.

Usage

```
then_bounded_float_ordered_sum(lhs, size_limit, bounds, .S = "Pairwise<.T>")
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>size_limit</code>	Upper bound on the number of records in input data. Used to bound sensitivity.
<code>bounds</code>	Tuple of lower and upper bounds for data in the input domain.
<code>.S</code>	Summation algorithm to use over some data type T (T is shorthand for <code>S::Item</code>)

Value

Transformation

then_bounded_int_monotonic_sum

partial bounded int monotonic sum constructor

Description

See documentation for [make_bounded_int_monotonic_sum\(\)](#) for details.

Usage

```
then_bounded_int_monotonic_sum(lhs, bounds, .T = NULL)
```

Arguments

lhs	The prior transformation or metric space.
bounds	Tuple of lower and upper bounds for data in the input domain.
.T	Atomic Input Type and Output Type

Value

Transformation

then_bounded_int_ordered_sum

partial bounded int ordered sum constructor

Description

See documentation for [make_bounded_int_ordered_sum\(\)](#) for details.

Usage

```
then_bounded_int_ordered_sum(lhs, bounds, .T = NULL)
```

Arguments

lhs	The prior transformation or metric space.
bounds	Tuple of lower and upper bounds for data in the input domain.
.T	Atomic Input Type and Output Type

Value

Transformation

```
then_bounded_int_split_sum
    partial bounded int split sum constructor
```

Description

See documentation for [make_bounded_int_split_sum\(\)](#) for details.

Usage

```
then_bounded_int_split_sum(lhs, bounds, .T = NULL)
```

Arguments

lhs	The prior transformation or metric space.
bounds	Tuple of lower and upper bounds for data in the input domain.
.T	Atomic Input Type and Output Type

Value

Transformation

```
then_canonical_noise partial canonical noise constructor
```

Description

See documentation for [make_canonical_noise\(\)](#) for details.

Usage

```
then_canonical_noise(lhs, d_in, d_out)
```

Arguments

lhs	The prior transformation or metric space.
d_in	Sensitivity
d_out	Privacy parameters (,)

Value

Measurement

<code>then_cast</code>	<i>partial cast constructor</i>
------------------------	---------------------------------

Description

See documentation for `make_cast()` for details.

Usage

```
then_cast(lhs, .TOA)
```

Arguments

- `lhs` The prior transformation or metric space.
- `.TOA` Atomic Output Type to cast into

Value

Transformation

<code>then_cast_default</code>	<i>partial cast default constructor</i>
--------------------------------	---

Description

See documentation for `make_cast_default()` for details.

Usage

```
then_cast_default(lhs, .TOA)
```

Arguments

- `lhs` The prior transformation or metric space.
- `.TOA` Atomic Output Type to cast into

Value

Transformation

then_cast_inherent	<i>partial cast inherent constructor</i>
--------------------	--

Description

See documentation for [make_cast_inherent\(\)](#) for details.

Usage

```
then_cast_inherent(lhs, .TOA)
```

Arguments

- lhs The prior transformation or metric space.
- .TOA Atomic Output Type to cast into

Value

Transformation

then_cdf	<i>partial cdf constructor</i>
----------	--------------------------------

Description

See documentation for [make_cdf\(\)](#) for details.

Usage

```
then_cdf(lhs, .TA = "float")
```

Arguments

- lhs The prior transformation or metric space.
- .TA Atomic Type. One of f32 or f64

Value

Function

<code>then_chain_mt</code>	<i>partial chain mt constructor</i>
----------------------------	-------------------------------------

Description

See documentation for `make_chain_mt()` for details.

Usage

```
then_chain_mt(lhs, measurement1, transformation0)
```

Arguments

- `lhs` The prior transformation or metric space.
- `measurement1` outer mechanism
- `transformation0` inner transformation

Value

Measurement

<code>then_chain_pm</code>	<i>partial chain pm constructor</i>
----------------------------	-------------------------------------

Description

See documentation for `make_chain_pm()` for details.

Usage

```
then_chain_pm(lhs, postprocess1, measurement0)
```

Arguments

- `lhs` The prior transformation or metric space.
- `postprocess1` outer postprocessor
- `measurement0` inner measurement/mechanism

Value

Measurement

then_chain_tt	<i>partial chain tt constructor</i>
---------------	-------------------------------------

Description

See documentation for [make_chain_tt\(\)](#) for details.

Usage

```
then_chain_tt(lhs, transformation1, transformation0)
```

Arguments

lhs	The prior transformation or metric space.
transformation1	outer transformation
transformation0	inner transformation

Value

Transformation

then_clamp	<i>partial clamp constructor</i>
------------	----------------------------------

Description

See documentation for [make_clamp\(\)](#) for details.

Usage

```
then_clamp(lhs, bounds)
```

Arguments

lhs	The prior transformation or metric space.
bounds	Tuple of inclusive lower and upper bounds.

Value

Transformation

then_composition	<i>partial composition constructor</i>
------------------	--

Description

See documentation for [make_composition\(\)](#) for details.

Usage

```
then_composition(lhs, measurements)
```

Arguments

lhs	The prior transformation or metric space.
measurements	A vector of Measurements to compose.

Value

Measurement

then_consistent_b_ary_tree	<i>partial consistent b ary tree constructor</i>
----------------------------	--

Description

See documentation for [make_consistent_b_ary_tree\(\)](#) for details.

Usage

```
then_consistent_b_ary_tree(lhs, branching_factor, .TIA = "int", .TOA = "float")
```

Arguments

lhs	The prior transformation or metric space.
branching_factor	the maximum number of children
.TIA	Atomic type of the input data. Should be an integer type.
.TOA	Atomic type of the output data. Should be a float type.

Value

Function

then_count	<i>partial count constructor</i>
------------	----------------------------------

Description

See documentation for [make_count\(\)](#) for details.

Usage

```
then_count(lhs, .TO = "int")
```

Arguments

lhs	The prior transformation or metric space.
.TO	Output Type. Must be numeric.

Value

Transformation

then_count_by	<i>partial count by constructor</i>
---------------	-------------------------------------

Description

See documentation for [make_count_by\(\)](#) for details.

Usage

```
then_count_by(lhs, .TV = "int")
```

Arguments

lhs	The prior transformation or metric space.
.TV	Type of Value. Express counts in terms of this integral type.

Value

The carrier type is `HashMap<TK, TV>`, a hashmap of the count (TV) for each unique data input (TK).

```
then_count_by_categories
```

partial count by categories constructor

Description

See documentation for [make_count_by_categories\(\)](#) for details.

Usage

```
then_count_by_categories(
  lhs,
  categories,
  null_category = TRUE,
  .MO = "L1Distance<int>",
  .TOA = "int"
)
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>categories</code>	The set of categories to compute counts for.
<code>null_category</code>	Include a count of the number of elements that were not in the category set at the end of the vector.
<code>.MO</code>	Output Metric.
<code>.TOA</code>	Atomic Output Type that is numeric.

Value

The carrier type is `Vec<TOA>`, a vector of the counts (`TOA`).

```
then_count_distinct
```

partial count distinct constructor

Description

See documentation for [make_count_distinct\(\)](#) for details.

Usage

```
then_count_distinct(lhs, .TO = "int")
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>.TO</code>	Output Type. Must be numeric.

Value

Transformation

<code>then_create_dataframe</code>	<i>partial create dataframe constructor</i>
------------------------------------	---

Description

See documentation for [make_create_dataframe\(\)](#) for details.

Usage

```
then_create_dataframe(lhs, col_names, .K = NULL)
```

Arguments

- | | |
|------------------------|--|
| <code>lhs</code> | The prior transformation or metric space. |
| <code>col_names</code> | Column names for each record entry. |
| <code>.K</code> | categorical/hashable data type of column names |

Value

Transformation

<code>then_df_cast_default</code>	<i>partial df cast default constructor</i>
-----------------------------------	--

Description

See documentation for [make_df_cast_default\(\)](#) for details.

Usage

```
then_df_cast_default(lhs, column_name, .TIA, .TOA)
```

Arguments

- | | |
|--------------------------|---|
| <code>lhs</code> | The prior transformation or metric space. |
| <code>column_name</code> | column name to be transformed |
| <code>.TIA</code> | Atomic Input Type to cast from |
| <code>.TOA</code> | Atomic Output Type to cast into |

Value

Transformation

<code>then_df_is_equal</code>	<i>partial df is equal constructor</i>
-------------------------------	--

Description

See documentation for [make_df_is_equal\(\)](#) for details.

Usage

```
then_df_is_equal(lhs, column_name, value, .TIA = NULL)
```

Arguments

- `lhs` The prior transformation or metric space.
- `column_name` Column name to be transformed
- `value` Value to check for equality
- `.TIA` Atomic Input Type to cast from

Value

Transformation

<code>then_drop_null</code>	<i>partial drop null constructor</i>
-----------------------------	--------------------------------------

Description

See documentation for [make_drop_null\(\)](#) for details.

Usage

```
then_drop_null(lhs)
```

Arguments

- `lhs` The prior transformation or metric space.

Value

Transformation

then_find	<i>partial find constructor</i>
-----------	---------------------------------

Description

See documentation for [make_find\(\)](#) for details.

Usage

```
then_find(lhs, categories)
```

Arguments

lhs	The prior transformation or metric space.
categories	The set of categories to find indexes from.

Value

Transformation

then_find_bin	<i>partial find bin constructor</i>
---------------	-------------------------------------

Description

See documentation for [make_find_bin\(\)](#) for details.

Usage

```
then_find_bin(lhs, edges)
```

Arguments

lhs	The prior transformation or metric space.
edges	The set of edges to split bins by.

Value

Transformation

then_fix_delta	<i>partial fix delta constructor</i>
----------------	--------------------------------------

Description

See documentation for [make_fix_delta\(\)](#) for details.

Usage

```
then_fix_delta(lhs, measurement, delta)
```

Arguments

lhs	The prior transformation or metric space.
measurement	a measurement with a privacy curve to be fixed
delta	parameter to fix the privacy curve with

Value

Measurement

then_fixed_approxDP_to_approxDP	<i>partial fixed approxDP to approxDP constructor</i>
---------------------------------	---

Description

See documentation for [make_fixed_approxDP_to_approxDP\(\)](#) for details.

Usage

```
then_fixed_approxDP_to_approxDP(lhs, measurement)
```

Arguments

lhs	The prior transformation or metric space.
measurement	a measurement with a privacy measure to be casted

Value

Measurement

```
then_fully_adaptive_composition
    partial fully adaptive composition constructor
```

Description

See documentation for [make_fully_adaptive_composition\(\)](#) for details.

Usage

```
then_fully_adaptive_composition(lhs, output_measure)
```

Arguments

lhs The prior transformation or metric space.
output_measure how privacy is measured

Value

Odometer

```
then_gaussian            partial gaussian constructor
```

Description

See documentation for [make_gaussian\(\)](#) for details.

Usage

```
then_gaussian(lhs, scale, k = NULL, .MO = "ZeroConcentratedDivergence")
```

Arguments

lhs The prior transformation or metric space.
scale Noise scale parameter for the gaussian distribution. `scale == standard_deviation`.
k The noise granularity in terms of 2^k .
.MO Output Measure. The only valid measure is `ZeroConcentratedDivergence`.

Value

Measurement

Examples

```
library(opendp)
enable_features("contrib")
gaussian <- make_gaussian(
  atom_domain(.T = f64, nan = FALSE),
  absolute_distance(.T = f64),
  scale = 1.0)
gaussian(arg = 100.0)

# Or, more readably, define the space and then chain:
space <- c(atom_domain(.T = f64, nan = FALSE), absolute_distance(.T = f64))
gaussian <- space |> then_gaussian(scale = 1.0)
gaussian(arg = 100.0)

# Sensitivity of this measurement:
gaussian(d_in = 1)
gaussian(d_in = 2)
gaussian(d_in = 4)

# Typically will be used with vectors rather than individual numbers:
space <- c(vector_domain(atom_domain(.T = i32)), l2_distance(.T = i32))
gaussian <- space |> then_gaussian(scale = 1.0)
gaussian(arg = c(10L, 20L, 30L))
```

```
then_gaussian_threshold
```

partial gaussian threshold constructor

Description

See documentation for [make_gaussian_threshold\(\)](#) for details.

Usage

```
then_gaussian_threshold(
  lhs,
  scale,
  threshold,
  k = NULL,
  .M0 = "Approximate<ZeroConcentratedDivergence>"
)
```

Arguments

lhs	The prior transformation or metric space.
scale	Noise scale parameter for the laplace distribution. <code>scale == standard_deviation / sqrt(2)</code> .
threshold	Exclude pairs with values whose distance from zero exceeds this value.

<code>k</code>	The noise granularity in terms of 2^k .
<code>.MO</code>	Output Measure.

Value

Measurement

<code>then_geometric</code>	<i>partial geometric constructor</i>
-----------------------------	--------------------------------------

Description

See documentation for [make_geometric\(\)](#) for details.

Usage

```
then_geometric(lhs, scale, bounds = NULL, .MO = "MaxDivergence")
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>scale</code>	Noise scale parameter for the distribution. <code>scale == standard_deviation / sqrt(2)</code> .
<code>bounds</code>	Set bounds on the count to make the algorithm run in constant-time.
<code>.MO</code>	Measure used to quantify privacy loss. Valid values are just <code>MaxDivergence</code>

Value

Measurement

<code>then_identity</code>	<i>partial identity constructor</i>
----------------------------	-------------------------------------

Description

See documentation for [make_identity\(\)](#) for details.

Usage

```
then_identity(lhs)
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
------------------	---

Value

Transformation

```
then_impute_constant
```

partial impute constant constructor

Description

See documentation for [make_impute_constant\(\)](#) for details.

Usage

```
then_impute_constant(lhs, constant)
```

Arguments

lhs	The prior transformation or metric space.
constant	Value to replace nulls with.

Value

Transformation

```
then_impute_uniform_float
```

partial impute uniform float constructor

Description

See documentation for [make_impute_uniform_float\(\)](#) for details.

Usage

```
then_impute_uniform_float(lhs, bounds)
```

Arguments

lhs	The prior transformation or metric space.
bounds	Tuple of inclusive lower and upper bounds.

Value

Transformation

then_index	<i>partial index constructor</i>
------------	----------------------------------

Description

See documentation for [make_index\(\)](#) for details.

Usage

```
then_index(lhs, categories, null, .TOA = NULL)
```

Arguments

lhs	The prior transformation or metric space.
categories	The set of categories to index into.
null	Category to return if the index is out-of-range of the category set.
.TOA	Atomic Output Type. Output data will be <code>Vec<TOA></code> .

Value

Transformation

then_is_equal	<i>partial is equal constructor</i>
---------------	-------------------------------------

Description

See documentation for [make_is_equal\(\)](#) for details.

Usage

```
then_is_equal(lhs, value)
```

Arguments

lhs	The prior transformation or metric space.
value	value to check against

Value

Transformation

<code>then_is_null</code>	<i>partial is null constructor</i>
---------------------------	------------------------------------

Description

See documentation for `make_is_null()` for details.

Usage

```
then_is_null(lhs)
```

Arguments

`lhs` The prior transformation or metric space.

Value

Transformation

<code>then_laplace</code>	<i>partial laplace constructor</i>
---------------------------	------------------------------------

Description

See documentation for `make_laplace()` for details.

Usage

```
then_laplace(lhs, scale, k = NULL, .MO = "MaxDivergence")
```

Arguments

`lhs` The prior transformation or metric space.

`scale` Noise scale parameter for the Laplace distribution. `scale == standard_deviation / sqrt(2)`.

`k` The noise granularity in terms of 2^k , only valid for domains over floats.

`.MO` Measure used to quantify privacy loss. Valid values are just `MaxDivergence`

Value

Measurement

```
then_laplace_threshold
```

partial laplace threshold constructor

Description

See documentation for [make_laplace_threshold\(\)](#) for details.

Usage

```
then_laplace_threshold(
    lhs,
    scale,
    threshold,
    k = NULL,
    .MO = "Approximate<MaxDivergence>"
)
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>scale</code>	Noise scale parameter for the laplace distribution. <code>scale == standard_deviation / sqrt(2)</code> .
<code>threshold</code>	Exclude counts that are less than this minimum value.
<code>k</code>	The noise granularity in terms of 2^k .
<code>.MO</code>	Output Measure.

Value

Measurement

```
then_lipschitz_float_mul
```

partial lipschitz float mul constructor

Description

See documentation for [make_lipschitz_float_mul\(\)](#) for details.

Usage

```
then_lipschitz_float_mul(lhs, constant, bounds)
```

Arguments

- `lhs` The prior transformation or metric space.
- `constant` The constant to multiply aggregates by.
- `bounds` Tuple of inclusive lower and upper bounds.

Value

Transformation

<code>then_mean</code>	<i>partial mean constructor</i>
------------------------	---------------------------------

Description

See documentation for [make_mean\(\)](#) for details.

Usage

```
then_mean(lhs)
```

Arguments

- `lhs` The prior transformation or metric space.

Value

Transformation

<code>then_metric_bounded</code>	<i>partial metric bounded constructor</i>
----------------------------------	---

Description

See documentation for [make_metric_bounded\(\)](#) for details.

Usage

```
then_metric_bounded(lhs)
```

Arguments

- `lhs` The prior transformation or metric space.

Value

Transformation

then_metric_unbounded	<i>partial metric unbounded constructor</i>
-----------------------	---

Description

See documentation for [make_metric_unbounded\(\)](#) for details.

Usage

```
then_metric_unbounded(lhs)
```

Arguments

lhs The prior transformation or metric space.

Value

Transformation

then_noise	<i>partial noise constructor</i>
------------	----------------------------------

Description

See documentation for [make_noise\(\)](#) for details.

Usage

```
then_noise(lhs, output_measure, scale, k = NULL)
```

Arguments

lhs The prior transformation or metric space.

output_measure Privacy measure. Either `MaxDivergence` or `ZeroConcentratedDivergence`.

scale Noise scale parameter.

k The noise granularity in terms of 2^k .

Value

Measurement

<code>then_noise_threshold</code>	<i>partial noise threshold constructor</i>
-----------------------------------	--

Description

See documentation for `make_noise_threshold()` for details.

Usage

```
then_noise_threshold(lhs, output_measure, scale, threshold, k = NULL)
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>output_measure</code>	Privacy measure. Either <code>MaxDivergence</code> or <code>ZeroConcentratedDivergence</code> .
<code>scale</code>	Noise scale parameter for the laplace distribution. <code>scale == standard_deviation / sqrt(2)</code> .
<code>threshold</code>	Exclude counts that are less than this minimum value.
<code>k</code>	The noise granularity in terms of 2^k .

Value

Measurement

<code>then_noisy_max</code>	<i>partial noisy max constructor</i>
-----------------------------	--------------------------------------

Description

See documentation for `make_noisy_max()` for details.

Usage

```
then_noisy_max(lhs, output_measure, scale, negate = FALSE)
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>output_measure</code>	One of <code>MaxDivergence</code> , <code>ZeroConcentratedDivergence</code>
<code>scale</code>	Scale for the noise distribution
<code>negate</code>	Set to true to return min

Value

Measurement

then_noisy_top_k	<i>partial noisy top k constructor</i>
------------------	--

Description

See documentation for [make_noisy_top_k\(\)](#) for details.

Usage

```
then_noisy_top_k(lhs, output_measure, k, scale, negate = FALSE)
```

Arguments

lhs	The prior transformation or metric space.
output_measure	One of <code>MaxDivergence</code> or <code>ZeroConcentratedDivergence</code>
k	Number of indices to select.
scale	Scale for the noise distribution.
negate	Set to true to return bottom k

Value

Measurement

then_ordered_random	<i>partial ordered random constructor</i>
---------------------	---

Description

See documentation for [make_ordered_random\(\)](#) for details.

Usage

```
then_ordered_random(lhs)
```

Arguments

lhs	The prior transformation or metric space.
-----	---

Value

Transformation

<code>then_population_amplification</code>	<i>partial population amplification constructor</i>
--	---

Description

See documentation for [make_population_amplification\(\)](#) for details.

Usage

```
then_population_amplification(lhs, measurement, population_size)
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>measurement</code>	the computation to amplify
<code>population_size</code>	the size of the population from which the input dataset is a simple sample

Value

Measurement

<code>then_postprocess</code>	<i>Compose a measurement with a postprocessing function.</i>
-------------------------------	--

Description

This provides Python-like ergonomics for chaining a pure function after a measurement release.

Usage

```
then_postprocess(lhs, f, .TO = "ExtrinsicObject")
```

Arguments

<code>lhs</code>	A measurement to postprocess.
<code>f</code>	A pure postprocessing function applied to the release.
<code>.TO</code>	The runtime type of the postprocessed output.

Value

A composed measurement.

`then_privacy_filter` *partial privacy filter constructor*

Description

See documentation for [make_privacy_filter\(\)](#) for details.

Usage

```
then_privacy_filter(lhs, odometer, d_in, d_out)
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>odometer</code>	A privacy odometer
<code>d_in</code>	Upper bound on the distance between adjacent datasets
<code>d_out</code>	Upper bound on the privacy loss

Value

Measurement

`then_private_quantile`
partial private quantile constructor

Description

See documentation for [make_private_quantile\(\)](#) for details.

Usage

```
then_private_quantile(lhs, output_measure, candidates, alpha, scale)
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>output_measure</code>	Either MaxDivergence or ZeroConcentratedDivergence.
<code>candidates</code>	Potential quantiles to score
<code>alpha</code>	a value in $[0, 1]$. Choose 0.5 for median
<code>scale</code>	the scale of the noise added

Value

Measurement

`then_pureDP_to_zCDP` *partial pureDP to zCDP constructor*

Description

See documentation for [make_pureDP_to_zCDP\(\)](#) for details.

Usage

```
then_pureDP_to_zCDP(lhs, measurement)
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>measurement</code>	a measurement with a privacy measure to be casted

Value

Measurement

`then_quantile_score_candidates`
partial quantile score candidates constructor

Description

See documentation for [make_quantile_score_candidates\(\)](#) for details.

Usage

```
then_quantile_score_candidates(lhs, candidates, alpha)
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>candidates</code>	Potential quantiles to score
<code>alpha</code>	a value in $[0, 1]$. Choose 0.5 for median

Value

Transformation

```
then_quantiles_from_counts
```

partial quantiles from counts constructor

Description

See documentation for [make_quantiles_from_counts\(\)](#) for details.

Usage

```
then_quantiles_from_counts(  
  lhs,  
  bin_edges,  
  alphas,  
  interpolation = "linear",  
  .TA = NULL,  
  .F = "float"  
)
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>bin_edges</code>	The edges that the input data was binned into before counting.
<code>alphas</code>	Return all specified alpha -quantiles.
<code>interpolation</code>	Must be one of linear or nearest
<code>.TA</code>	Atomic Type of the bin edges and data.
<code>.F</code>	Float type of the alpha argument. One of f32 or f64

Value

Function

```
then_randomized_response
```

partial randomized response constructor

Description

See documentation for [make_randomized_response\(\)](#) for details.

Usage

```
then_randomized_response(lhs, categories, prob, .T = NULL)
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>categories</code>	Set of valid outcomes
<code>prob</code>	Probability of returning the correct answer. Must be in $[1/\text{num_categories}, 1]$
<code>.T</code>	Data type of a category.

Value

Measurement

`then_randomized_response_bitvec`
partial randomized response bitvec constructor

Description

See documentation for [make_randomized_response_bitvec\(\)](#) for details.

Usage

```
then_randomized_response_bitvec(lhs, f, constant_time = FALSE)
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>f</code>	Per-bit flipping probability. Must be in $(0, 1]$.
<code>constant_time</code>	Whether to run the Bernoulli samplers in constant time, this is likely to be extremely slow.

Value

Measurement

```
then_randomized_response_bool
```

partial randomized response bool constructor

Description

See documentation for [make_randomized_response_bool\(\)](#) for details.

Usage

```
then_randomized_response_bool(lhs, prob, constant_time = FALSE)
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>prob</code>	Probability of returning the correct answer. Must be in [0.5, 1]
<code>constant_time</code>	Set to true to enable constant time. Slower.

Value

Measurement

```
then_report_noisy_max_gumbel
```

partial report noisy max gumbel constructor

Description

See documentation for [make_report_noisy_max_gumbel\(\)](#) for details.

Usage

```
then_report_noisy_max_gumbel(lhs, scale, optimize = "max")
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>scale</code>	Scale for the noise distribution
<code>optimize</code>	Set to "min" to report noisy min

Value

Measurement

<code>then_resize</code>	<i>partial resize constructor</i>
--------------------------	-----------------------------------

Description

See documentation for `make_resize()` for details.

Usage

```
then_resize(lhs, size, constant, .MO = "SymmetricDistance")
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>size</code>	Number of records in output data.
<code>constant</code>	Value to impute with.
<code>.MO</code>	Output Metric. One of <code>InsertDeleteDistance</code> or <code>SymmetricDistance</code>

Value

A vector of the same type `TA`, but with the provided `size`.

<code>then_select_column</code>	<i>partial select column constructor</i>
---------------------------------	--

Description

See documentation for `make_select_column()` for details.

Usage

```
then_select_column(lhs, key, .TOA, .K = NULL)
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>key</code>	categorical/hashable data type of the key/column name
<code>.TOA</code>	Atomic Output Type to downcast vector to
<code>.K</code>	data type of key

Value

Transformation

```
then_select_private_candidate
    partial select private candidate constructor
```

Description

See documentation for [make_select_private_candidate\(\)](#) for details.

Usage

```
then_select_private_candidate(lhs, measurement, stop_probability, threshold)
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>measurement</code>	A measurement that releases a 2-tuple of (score, candidate)
<code>stop_probability</code>	The probability of stopping early at any iteration.
<code>threshold</code>	The threshold score. Return immediately if the score is above this threshold.

Value

A measurement that returns a release from `measurement` whose score is greater than `threshold`, or none.

```
then_sequential_composition
    partial sequential composition constructor
```

Description

See documentation for [make_sequential_composition\(\)](#) for details.

Usage

```
then_sequential_composition(lhs, output_measure, d_in, d_mids)
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>output_measure</code>	how privacy is measured
<code>d_in</code>	maximum distance between adjacent input datasets
<code>d_mids</code>	maximum privacy expenditure of each query

Value

Measurement

```
then_sized_bounded_float_checked_sum
```

partial sized bounded float checked sum constructor

Description

See documentation for [make_sized_bounded_float_checked_sum\(\)](#) for details.

Usage

```
then_sized_bounded_float_checked_sum(lhs, size, bounds, .S = "Pairwise<T>")
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>size</code>	Number of records in input data.
<code>bounds</code>	Tuple of lower and upper bounds for data in the input domain.
<code>.S</code>	Summation algorithm to use over some data type T (T is shorthand for <code>S::Item</code>)

Value

Transformation

```
then_sized_bounded_float_ordered_sum
```

partial sized bounded float ordered sum constructor

Description

See documentation for [make_sized_bounded_float_ordered_sum\(\)](#) for details.

Usage

```
then_sized_bounded_float_ordered_sum(lhs, size, bounds, .S = "Pairwise<T>")
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>size</code>	Number of records in input data.
<code>bounds</code>	Tuple of lower and upper bounds for data in the input domain.
<code>.S</code>	Summation algorithm to use over some data type T (T is shorthand for <code>S::Item</code>)

Value

Transformation

then_sized_bounded_int_checked_sum

partial sized bounded int checked sum constructor

Description

See documentation for [make_sized_bounded_int_checked_sum\(\)](#) for details.

Usage

```
then_sized_bounded_int_checked_sum(lhs, size, bounds, .T = NULL)
```

Arguments

lhs	The prior transformation or metric space.
size	Number of records in input data.
bounds	Tuple of lower and upper bounds for data in the input domain.
.T	Atomic Input Type and Output Type

Value

Transformation

then_sized_bounded_int_monotonic_sum

partial sized bounded int monotonic sum constructor

Description

See documentation for [make_sized_bounded_int_monotonic_sum\(\)](#) for details.

Usage

```
then_sized_bounded_int_monotonic_sum(lhs, size, bounds, .T = NULL)
```

Arguments

lhs	The prior transformation or metric space.
size	Number of records in input data.
bounds	Tuple of lower and upper bounds for data in the input domain.
.T	Atomic Input Type and Output Type

Value

Transformation

```
then_sized_bounded_int_ordered_sum
```

partial sized bounded int ordered sum constructor

Description

See documentation for [make_sized_bounded_int_ordered_sum\(\)](#) for details.

Usage

```
then_sized_bounded_int_ordered_sum(lhs, size, bounds, .T = NULL)
```

Arguments

lhs	The prior transformation or metric space.
size	Number of records in input data.
bounds	Tuple of lower and upper bounds for data in the input domain.
.T	Atomic Input Type and Output Type

Value

Transformation

```
then_sized_bounded_int_split_sum
```

partial sized bounded int split sum constructor

Description

See documentation for [make_sized_bounded_int_split_sum\(\)](#) for details.

Usage

```
then_sized_bounded_int_split_sum(lhs, size, bounds, .T = NULL)
```

Arguments

lhs	The prior transformation or metric space.
size	Number of records in input data.
bounds	Tuple of lower and upper bounds for data in the input domain.
.T	Atomic Input Type and Output Type

Value

Transformation

`then_split_dataframe` *partial split dataframe constructor*

Description

See documentation for [make_split_dataframe\(\)](#) for details.

Usage

```
then_split_dataframe(lhs, separator, col_names, .K = NULL)
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>separator</code>	The token(s) that separate entries in each record.
<code>col_names</code>	Column names for each record entry.
<code>.K</code>	categorical/hashable data type of column names

Value

Transformation

`then_split_lines` *partial split lines constructor*

Description

See documentation for [make_split_lines\(\)](#) for details.

Usage

```
then_split_lines(lhs)
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
------------------	---

Value

Transformation

<code>then_split_records</code>	<i>partial split records constructor</i>
---------------------------------	--

Description

See documentation for `make_split_records()` for details.

Usage

```
then_split_records(lhs, separator)
```

Arguments

- `lhs` The prior transformation or metric space.
- `separator` The token(s) that separate entries in each record.

Value

Transformation

<code>then_subset_by</code>	<i>partial subset by constructor</i>
-----------------------------	--------------------------------------

Description

See documentation for `make_subset_by()` for details.

Usage

```
then_subset_by(lhs, indicator_column, keep_columns, .TK = NULL)
```

Arguments

- `lhs` The prior transformation or metric space.
- `indicator_column` name of the boolean column that indicates inclusion in the subset
- `keep_columns` list of column names to apply subset to
- `.TK` Type of the column name

Value

Transformation

then_sum	<i>partial sum constructor</i>
----------	--------------------------------

Description

See documentation for [make_sum\(\)](#) for details.

Usage

```
then_sum(lhs)
```

Arguments

lhs	The prior transformation or metric space.
-----	---

Value

Transformation

then_sum_of_squared_deviations	<i>partial sum of squared deviations constructor</i>
--------------------------------	--

Description

See documentation for [make_sum_of_squared_deviations\(\)](#) for details.

Usage

```
then_sum_of_squared_deviations(lhs, .S = "Pairwise<T>")
```

Arguments

lhs	The prior transformation or metric space.
.S	Summation algorithm to use on data type T. One of <code>Sequential<T></code> or <code>Pairwise<T></code> .

Value

Transformation

<code>then_unordered</code>	<i>partial unordered constructor</i>
-----------------------------	--------------------------------------

Description

See documentation for [make_unordered\(\)](#) for details.

Usage

```
then_unordered(lhs)
```

Arguments

`lhs` The prior transformation or metric space.

Value

Transformation

<code>then_user_measurement</code>	<i>partial user measurement constructor</i>
------------------------------------	---

Description

See documentation for [make_user_measurement\(\)](#) for details.

Usage

```
then_user_measurement(  
  lhs,  
  output_measure,  
  function_,  
  privacy_map,  
  .T0 = "ExtrinsicObject"  
)
```

Arguments

`lhs` The prior transformation or metric space.

`output_measure` The measure from which distances between adjacent output distributions are measured.

`function_` A function mapping data from `input_domain` to a release of type `T0`.

`privacy_map` A function mapping distances from `input_metric` to `output_measure`.

`.T0` The data type of outputs from the function.

Value

Measurement

```
then_user_transformation
```

partial user transformation constructor

Description

See documentation for [make_user_transformation\(\)](#) for details.

Usage

```
then_user_transformation(
    lhs,
    input_domain,
    input_metric,
    output_domain,
    output_metric,
    function_,
    stability_map
)
```

Arguments

<code>lhs</code>	The prior transformation or metric space.
<code>input_domain</code>	A domain describing the set of valid inputs for the function.
<code>input_metric</code>	The metric from which distances between adjacent inputs are measured.
<code>output_domain</code>	A domain describing the set of valid outputs of the function.
<code>output_metric</code>	The metric from which distances between outputs of adjacent inputs are measured.
<code>function_</code>	A function mapping data from <code>input_domain</code> to <code>output_domain</code> .
<code>stability_map</code>	A function mapping distances from <code>input_metric</code> to <code>output_metric</code> .

Value

Transformation

<code>then_variance</code>	<i>partial variance constructor</i>
----------------------------	-------------------------------------

Description

See documentation for [make_variance\(\)](#) for details.

Usage

```
then_variance(lhs, ddof = 1L, .S = "Pairwise<T>")
```

Arguments

- | | |
|-------------------|--|
| <code>lhs</code> | The prior transformation or metric space. |
| <code>ddof</code> | Delta degrees of freedom. Set to 0 if not a sample, 1 for sample estimate. |
| <code>.S</code> | Summation algorithm to use on data type T. One of <code>Sequential<T></code> or <code>Pairwise<T></code> . |

Value

Transformation

<code>then_zCDP_to_approxDP</code>	<i>partial zCDP to approxDP constructor</i>
------------------------------------	---

Description

See documentation for [make_zCDP_to_approxDP\(\)](#) for details.

Usage

```
then_zCDP_to_approxDP(lhs, measurement)
```

Arguments

- | | |
|--------------------------|---|
| <code>lhs</code> | The prior transformation or metric space. |
| <code>measurement</code> | a measurement with a privacy measure to be casted |

Value

Measurement

to_str.default	Convert a format-able value to a string representation
----------------	--

Description

Convert a format-able value to a string representation

Usage

```
## Default S3 method:  
to_str(x, depth)
```

Arguments

- x value to convert to string
- depth offset from start of string

to_str.hashtab	Convert hashtab to a string representation
----------------	--

Description

Convert hashtab to a string representation

Usage

```
## S3 method for class 'hashtab'  
to_str(x, depth = 0L)
```

Arguments

- x value to convert to string
- depth offset from start of string

<code>to_then</code>	<i>Convert a <code>make_</code> constructor into a <code>then_</code> constructor.</i>
----------------------	--

Description

This mirrors the Python extras utility for constructors whose first two arguments are `input_domain` and `input_metric`.

Usage

```
to_then(constructor)
```

Arguments

`constructor` A constructor whose first two arguments are `input_domain` and `input_metric`.

Value

A partial constructor suitable for `|>` chaining.

<code>transformation_check</code>	<i>Check the privacy relation of the <code>measurement</code> at the given <code>d_in</code>, <code>d_out</code></i>
-----------------------------------	--

Description

Check the privacy relation of the `measurement` at the given `d_in`, `d_out`

Usage

```
transformation_check(transformation, distance_in, distance_out)
```

Arguments

<code>transformation</code>	undocumented
<code>distance_in</code>	undocumented
<code>distance_out</code>	undocumented

Value

True indicates that the relation passed at the given distance.

`transformation_function`*Get the function from a transformation.*

Description

Get the function from a transformation.

Usage

```
transformation_function(this)
```

Arguments

`this` The transformation to retrieve the value from.

Value

Function

`transformation_input_carrier_type`*Get the input (carrier) data type of this.*

Description

Get the input (carrier) data type of `this`.

Usage

```
transformation_input_carrier_type(this)
```

Arguments

`this` The transformation to retrieve the type from.

Value

str

`transformation_input_distance_type`*Get the input distance type of transformation.*

Description

Get the input distance type of `transformation`.

Usage

```
transformation_input_distance_type(this)
```

Arguments

`this` The transformation to retrieve the type from.

Value

str

`transformation_input_domain`*Get the input domain from a transformation.*

Description

Get the input domain from a `transformation`.

Usage

```
transformation_input_domain(this)
```

Arguments

`this` The transformation to retrieve the value from.

Value

Domain

`transformation_input_metric`*Get the input domain from a transformation.*

Description

Get the input domain from a `transformation`.

Usage

```
transformation_input_metric(this)
```

Arguments

<code>this</code>	The transformation to retrieve the value from.
-------------------	--

Value

Metric

`transformation_invoke`*Invoke the transformation with `arg`. Returns a differentially private release.*

Description

Invoke the `transformation` with `arg`. Returns a differentially private release.

Usage

```
transformation_invoke(this, arg)
```

Arguments

<code>this</code>	Transformation to invoke.
<code>arg</code>	Input data to supply to the transformation. A member of the transformation's input domain.

<code>transformation_map</code>	<i>Use the transformation to map a given <code>d_in</code> to <code>d_out</code>.</i>
---------------------------------	---

Description

Use the `transformation` to map a given `d_in` to `d_out`.

Usage

```
transformation_map(transformation, distance_in)
```

Arguments

<code>transformation</code>	Transformation to check the map distances with.
<code>distance_in</code>	Distance in terms of the input metric.

<code>transformation_output_distance_type</code>	<i>Get the output distance type of transformation.</i>
--	--

Description

Get the output distance type of `transformation`.

Usage

```
transformation_output_distance_type(this)
```

Arguments

<code>this</code>	The transformation to retrieve the type from.
-------------------	---

Value

`str`

`transformation_output_domain`*Get the output domain from a transformation.*

Description

Get the output domain from a `transformation`.

Usage

```
transformation_output_domain(this)
```

Arguments

`this` The transformation to retrieve the value from.

Value

Domain

`transformation_output_metric`*Get the output domain from a transformation.*

Description

Get the output domain from a `transformation`.

Usage

```
transformation_output_metric(this)
```

Arguments

`this` The transformation to retrieve the value from.

Value

Metric

u128	<i>type signature for a 128-bit unsigned integer</i>
------	--

Description

type signature for a 128-bit unsigned integer

Usage

u128

u16	<i>type signature for a 16-bit unsigned integer</i>
-----	---

Description

type signature for a 16-bit unsigned integer

Usage

u16

u32	<i>type signature for a 32-bit unsigned integer</i>
-----	---

Description

type signature for a 32-bit unsigned integer

Usage

u32

u64	<i>type signature for a 64-bit unsigned integer</i>
-----	---

Description

type signature for a 64-bit unsigned integer

Usage

u64

u8	<i>type signature for an 8-bit unsigned integer</i>
----	---

Description

type signature for an 8-bit unsigned integer

Usage

u8

user_distance	<i>Construct a new UserDistance. Any two instances of an UserDistance are equal if their string descriptors are equal.</i>
---------------	--

Description

Required features: `honest-but-curious`

Usage

```
user_distance(identifier, descriptor = NULL)
```

Arguments

identifier A string description of the metric.
descriptor Additional constraints on the domain.

Details

Why `honest-but-curious`?:

Your definition of `d` must satisfy the requirements of a pseudo-metric:

1. for any x , $d(x, x) = 0$
2. for any x, y , $d(x, y) \geq 0$ (non-negativity)
3. for any x, y , $d(x, y) = d(y, x)$ (symmetry)
4. for any x, y, z , $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality)

Value

Metric

<code>user_divergence</code>	<i>Privacy measure with meaning defined by an OpenDP Library user (you).</i>
------------------------------	--

Description

Any two instances of `UserDivergence` are equal if their descriptors compare equal.

Usage

```
user_divergence(identifier, descriptor = NULL)
```

Arguments

<code>identifier</code>	A string description of the privacy measure.
<code>descriptor</code>	Additional constraints on the privacy measure.

Details

Required features: `honest-but-curious`

Why `honest-but-curious`?:

The essential requirement of a privacy measure is that it is closed under postprocessing. Your privacy measure `D` must satisfy that, for any pure function `f` and any two distributions `Y`, `Y'`, then $D(Y, Y') \geq D(f(Y), f(Y'))$.

Beyond this, you should also consider whether your privacy measure can be used to provide meaningful privacy guarantees to your privacy units.

Value

Measure

<code>user_domain</code>	<i>Construct a new <code>UserDomain</code>. Any two instances of an <code>UserDomain</code> are equal if their string descriptors are equal. Contains a function used to check if any value is a member of the domain.</i>
--------------------------	--

Description

Required features: `honest-but-curious`

Usage

```
user_domain(identifier, member, descriptor = NULL)
```

Arguments

<code>identifier</code>	A string description of the data domain.
<code>member</code>	A function used to test if a value is a member of the data domain.
<code>descriptor</code>	Additional constraints on the domain.

Details

Why honest-but-curious?:

The identifier must uniquely identify this domain. If the identifier is not uniquely identifying, then two different domains with the same identifier will chain, which can violate transformation stability.

In addition, the member function must:

1. be a pure function
2. be sound (only return true if its input is a member of the domain).

Value

Domain

<code>usize</code>	<i>type signature for a pointer-sized unsigned integer</i>
--------------------	--

Description

type signature for a pointer-sized unsigned integer

Usage

`usize`

<code>vector_domain</code>	<i>Construct an instance of <code>VectorDomain</code>.</i>
----------------------------	--

Description

Construct an instance of `VectorDomain`.

Usage

`vector_domain(atom_domain, size = NULL)`

Arguments

<code>atom_domain</code>	The inner domain.
<code>size</code>	undocumented

Value

VectorDomain

zero_concentrated_divergence*Privacy measure used to define ρ -zero concentrated differential privacy.*

Description

In the following proof definition, d corresponds to ρ when also quantified over all adjacent datasets. That is, ρ is the greatest possible d over all pairs of adjacent datasets x, x' where $Y \sim M(x)$, $Y' \sim M(x')$. $M(\cdot)$ is a measurement (commonly known as a mechanism). The measurement's input metric defines the notion of adjacency, and the measurement's input domain defines the set of possible datasets.

Usage`zero_concentrated_divergence()`**Details****Proof Definition:**

For any two distributions Y, Y' and any non-negative d , Y, Y' are d -close under the zero-concentrated divergence measure if, for every possible choice of $\alpha \in (1, \infty)$,

$$D_\alpha(Y, Y') = \frac{1}{1-\alpha} \mathbb{E}_{x \sim Y'} \left[\ln \left(\frac{\Pr[Y = x]}{\Pr[Y' = x]} \right)^\alpha \right] \leq d \cdot \alpha.$$

Value

Measure

Index

- * **accuracy**
 - accuracy_to_discrete_gaussian_scale, [8](#)
 - accuracy_to_discrete_laplacian_scale, [9](#)
 - accuracy_to_gaussian_scale, [10](#)
 - accuracy_to_laplacian_scale, [10](#)
 - discrete_gaussian_scale_to_accuracy, [17](#)
 - discrete_laplacian_scale_to_accuracy, [18](#)
 - gaussian_scale_to_accuracy, [22](#)
 - laplacian_scale_to_accuracy, [27](#)
- * **combinators**
 - make_adaptive_composition, [28](#)
 - make_approximate, [31](#)
 - make_basic_composition, [32](#)
 - make_chain_mt, [42](#)
 - make_chain_pm, [43](#)
 - make_chain_tt, [44](#)
 - make_composition, [45](#)
 - make_fix_delta, [56](#)
 - make_fixed_approxDP_to_approxDP, [56](#)
 - make_fully_adaptive_composition, [57](#)
 - make_population_amplification, [78](#)
 - make_privacy_filter, [78](#)
 - make_pureDP_to_zCDP, [81](#)
 - make_select_private_candidate, [88](#)
 - make_sequential_composition, [89](#)
 - make_zCDP_to_approxDP, [106](#)
 - then_adaptive_composition, [134](#)
 - then_approximate, [135](#)
 - then_basic_composition, [136](#)
 - then_chain_mt, [142](#)
 - then_chain_pm, [142](#)
 - then_chain_tt, [143](#)
 - then_composition, [144](#)
 - then_fix_delta, [150](#)
 - then_fixed_approxDP_to_approxDP, [150](#)
 - then_fully_adaptive_composition, [151](#)
 - then_population_amplification, [162](#)
 - then_privacy_filter, [163](#)
 - then_pureDP_to_zCDP, [164](#)
 - then_select_private_candidate, [169](#)
 - then_sequential_composition, [169](#)
 - then_zCDP_to_approxDP, [178](#)
- * **core**
 - function_eval, [22](#)
 - measurement_check, [109](#)
 - measurement_function, [109](#)
 - measurement_input_carrier_type, [110](#)
 - measurement_input_distance_type, [110](#)
 - measurement_input_domain, [111](#)
 - measurement_input_metric, [111](#)
 - measurement_invoke, [112](#)
 - measurement_map, [112](#)
 - measurement_output_distance_type, [113](#)
 - measurement_output_measure, [113](#)
 - new_function, [116](#)
 - new_queryable, [120](#)
 - odometer_input_carrier_type, [122](#)
 - odometer_input_domain, [123](#)
 - odometer_input_metric, [123](#)
 - odometer_invoke, [124](#)
 - odometer_output_measure, [124](#)
 - odometer_queryable_invoke, [125](#)
 - odometer_queryable_invoke_type, [125](#)
 - odometer_queryable_privacy_loss,

- 126
- odometer_queryable_privacy_loss_type, 126
- queryable_eval, 130
- queryable_query_type, 130
- transformation_check, 180
- transformation_function, 181
- transformation_input_carrier_type, 181
- transformation_input_distance_type, 182
- transformation_input_domain, 182
- transformation_input_metric, 183
- transformation_invoke, 183
- transformation_map, 184
- transformation_output_distance_type, 184
- transformation_output_domain, 185
- transformation_output_metric, 185
- * **data**
 - object_type, 122
 - privacy_profile_delta, 129
 - privacy_profile_epsilon, 129
- * **domains**
 - atom_domain, 12
 - bitvector_domain, 15
 - domain_carrier_type, 19
 - domain_debug, 19
 - domain_type, 20
 - map_domain, 106
 - option_domain, 128
 - user_domain, 188
 - vector_domain, 189
- * **extras**
 - then_postprocess, 162
 - to_then, 180
- * **measurements**
 - make_alp_queryable, 29
 - make_canonical_noise, 38
 - make_gaussian, 58
 - make_gaussian_threshold, 59
 - make_geometric, 61
 - make_laplace, 67
 - make_laplace_threshold, 68
 - make_noise, 73
 - make_noise_threshold, 74
 - make_noisy_max, 75
 - make_noisy_top_k, 76
 - make_private_quantile, 79
 - make_randomized_response, 83
 - make_randomized_response_bitvec, 84
 - make_randomized_response_bool, 85
 - make_report_noisy_max_gumbel, 86
 - make_user_measurement, 102
 - then_alp_queryable, 134
 - then_canonical_noise, 139
 - then_gaussian, 151
 - then_gaussian_threshold, 152
 - then_geometric, 153
 - then_laplace, 156
 - then_laplace_threshold, 157
 - then_noise, 159
 - then_noise_threshold, 160
 - then_noisy_max, 160
 - then_noisy_top_k, 161
 - then_private_quantile, 163
 - then_randomized_response, 165
 - then_randomized_response_bitvec, 166
 - then_randomized_response_bool, 167
 - then_report_noisy_max_gumbel, 167
 - then_user_measurement, 176
- * **measures**
 - approximate, 11
 - fixed_smoothed_max_divergence, 21
 - max_divergence, 107
 - measure_debug, 107
 - measure_distance_type, 108
 - measure_type, 108
 - new_privacy_profile, 119
 - renyi_divergence, 131
 - smoothed_max_divergence, 132
 - user_divergence, 188
 - zero_concentrated_divergence, 190
- * **metrics**
 - absolute_distance, 8
 - change_one_distance, 16
 - discrete_distance, 17
 - hamming_distance, 23
 - insert_delete_distance, 25
 - l01inf_distance, 25
 - l02inf_distance, 26
 - l1_distance, 26
 - l2_distance, 27

- linf_distance, 28
- metric_debug, 114
- metric_distance_type, 114
- metric_type, 115
- symmetric_distance, 133
- user_distance, 187
- * **mod**
 - binary_search, 12
 - binary_search_chain, 13
 - binary_search_param, 14
 - disable_features, 17
 - enable_features, 20
 - hashitems, 23
 - new_domain, 115
 - new_function_internal, 116
 - new_hashtab, 117
 - new_measure, 117
 - new_measurement, 118
 - new_metric, 118
 - new_odometer, 118
 - new_odometer_queryable_internal, 119
 - new_privacy_profile_internal, 120
 - new_queryable_internal, 121
 - new_transformation, 121
 - to_str.default, 179
 - to_str.hashtab, 179
- * **transformations**
 - choose_branching_factor, 16
 - make_b_ary_tree, 31
 - make_bounded_float_checked_sum, 33
 - make_bounded_float_ordered_sum, 34
 - make_bounded_int_monotonic_sum, 35
 - make_bounded_int_ordered_sum, 36
 - make_bounded_int_split_sum, 37
 - make_cast, 39
 - make_cast_default, 40
 - make_cast_inherent, 41
 - make_cdf, 42
 - make_clamp, 44
 - make_consistent_b_ary_tree, 46
 - make_count, 47
 - make_count_by, 48
 - make_count_by_categories, 49
 - make_count_distinct, 50
 - make_create_dataframe, 51
 - make_df_cast_default, 51
 - make_df_is_equal, 52
 - make_drop_null, 53
 - make_find, 54
 - make_find_bin, 55
 - make_identity, 62
 - make_impute_constant, 63
 - make_impute_uniform_float, 64
 - make_index, 64
 - make_is_equal, 65
 - make_is_null, 66
 - make_lipschitz_float_mul, 69
 - make_mean, 70
 - make_metric_bounded, 71
 - make_metric_unbounded, 72
 - make_ordered_random, 77
 - make_quantile_score_candidates, 81
 - make_quantiles_from_counts, 82
 - make_resize, 87
 - make_select_column, 88
 - make_sized_bounded_float_checked_sum, 91
 - make_sized_bounded_float_ordered_sum, 92
 - make_sized_bounded_int_checked_sum, 93
 - make_sized_bounded_int_monotonic_sum, 94
 - make_sized_bounded_int_ordered_sum, 95
 - make_sized_bounded_int_split_sum, 96
 - make_split_dataframe, 97
 - make_split_lines, 97
 - make_split_records, 98
 - make_subset_by, 99
 - make_sum, 99
 - make_sum_of_squared_deviations, 100
 - make_unordered, 101
 - make_user_transformation, 103
 - make_variance, 105
 - then_b_ary_tree, 136
 - then_bounded_float_checked_sum, 137
 - then_bounded_float_ordered_sum,

- 137
- then_bounded_int_monotonic_sum, 138
- then_bounded_int_ordered_sum, 138
- then_bounded_int_split_sum, 139
- then_cast, 140
- then_cast_default, 140
- then_cast_inherent, 141
- then_cdf, 141
- then_clamp, 143
- then_consistent_b_ary_tree, 144
- then_count, 145
- then_count_by, 145
- then_count_by_categories, 146
- then_count_distinct, 146
- then_create_dataframe, 147
- then_df_cast_default, 147
- then_df_is_equal, 148
- then_drop_null, 148
- then_find, 149
- then_find_bin, 149
- then_identity, 153
- then_impute_constant, 154
- then_impute_uniform_float, 154
- then_index, 155
- then_is_equal, 155
- then_is_null, 156
- then_lipschitz_float_mul, 157
- then_mean, 158
- then_metric_bounded, 158
- then_metric_unbounded, 159
- then_ordered_random, 161
- then_quantile_score_candidates, 164
- then_quantiles_from_counts, 165
- then_resize, 168
- then_select_column, 168
- then_sized_bounded_float_checked_sum, 170
- then_sized_bounded_float_ordered_sum, 170
- then_sized_bounded_int_checked_sum, 171
- then_sized_bounded_int_monotonic_sum, 171
- then_sized_bounded_int_ordered_sum, 172
- then_sized_bounded_int_split_sum, 172
- then_split_dataframe, 173
- then_split_lines, 173
- then_split_records, 174
- then_subset_by, 174
- then_sum, 175
- then_sum_of_squared_deviations, 175
- then_unordered, 176
- then_user_transformation, 177
- then_variance, 178
- * **typing**
 - BitVector, 15
 - bool, 15
 - ExtrinsicObject, 20
 - f32, 21
 - f64, 21
 - i128, 24
 - i16, 24
 - i32, 24
 - i64, 24
 - i8, 25
 - parse_or_infer, 128
 - rt_infer, 131
 - rt_parse, 132
 - String, 133
 - u128, 186
 - u16, 186
 - u32, 186
 - u64, 186
 - u8, 187
 - usize, 189
- absolute_distance, 8
- accuracy_to_discrete_gaussian_scale, 8
- accuracy_to_discrete_laplacian_scale, 9
- accuracy_to_gaussian_scale, 10
- accuracy_to_laplacian_scale, 10
- approximate, 11
- atom_domain, 12
- binary_search, 12
- binary_search_chain, 13
- binary_search_param, 14
- BitVector, 15
- bitvector_domain, 15
- bool, 15

change_one_distance, 16
 choose_branching_factor, 16

 disable_features, 17
 discrete_distance, 17
 discrete_gaussian_scale_to_accuracy, 17
 discrete_laplacian_scale_to_accuracy, 18
 domain_carrier_type, 19
 domain_debug, 19
 domain_type, 20

 enable_features, 20
 ExtrinsicObject, 20

 f32, 21
 f64, 21
 fixed_smoothed_max_divergence, 21
 function_eval, 22

 gaussian_scale_to_accuracy, 22

 hamming_distance, 23
 hashitems, 23

 i128, 24
 i16, 24
 i32, 24
 i64, 24
 i8, 25
 insert_delete_distance, 25

 l01inf_distance, 25
 l02inf_distance, 26
 l1_distance, 26
 l2_distance, 27
 laplacian_scale_to_accuracy, 27
 linf_distance, 28

 make_adaptive_composition, 28
 make_adaptive_composition(), 134
 make_alp_queryable, 29
 make_alp_queryable(), 134
 make_approximate, 31
 make_approximate(), 135
 make_b_ary_tree, 31
 make_b_ary_tree(), 136
 make_basic_composition, 32
 make_basic_composition(), 136

 make_bounded_float_checked_sum, 33
 make_bounded_float_checked_sum(), 137
 make_bounded_float_ordered_sum, 34
 make_bounded_float_ordered_sum(), 137
 make_bounded_int_monotonic_sum, 35
 make_bounded_int_monotonic_sum(), 138
 make_bounded_int_ordered_sum, 36
 make_bounded_int_ordered_sum(), 138
 make_bounded_int_split_sum, 37
 make_bounded_int_split_sum(), 139
 make_canonical_noise, 38
 make_canonical_noise(), 139
 make_cast, 39
 make_cast(), 140
 make_cast_default, 40
 make_cast_default(), 140
 make_cast_inherent, 41
 make_cast_inherent(), 141
 make_cdf, 42
 make_cdf(), 141
 make_chain_mt, 42
 make_chain_mt(), 142
 make_chain_pm, 43
 make_chain_pm(), 142
 make_chain_tt, 44
 make_chain_tt(), 143
 make_clamp, 44
 make_clamp(), 143
 make_composition, 45
 make_composition(), 144
 make_consistent_b_ary_tree, 46
 make_consistent_b_ary_tree(), 144
 make_count, 47
 make_count(), 145
 make_count_by, 48
 make_count_by(), 145
 make_count_by_categories, 49
 make_count_by_categories(), 146
 make_count_distinct, 50
 make_count_distinct(), 146
 make_create_dataframe, 51
 make_create_dataframe(), 147
 make_df_cast_default, 51
 make_df_cast_default(), 147
 make_df_is_equal, 52
 make_df_is_equal(), 148
 make_drop_null, 53
 make_drop_null(), 148

make_find, 54
 make_find(), 149
 make_find_bin, 55
 make_find_bin(), 149
 make_fix_delta, 56
 make_fix_delta(), 150
 make_fixed_approxDP_to_approxDP, 56
 make_fixed_approxDP_to_approxDP(), 150
 make_fully_adaptive_composition, 57
 make_fully_adaptive_composition(), 151
 make_gaussian, 58
 make_gaussian(), 151
 make_gaussian_threshold, 59
 make_gaussian_threshold(), 152
 make_geometric, 61
 make_geometric(), 153
 make_identity, 62
 make_identity(), 153
 make_impute_constant, 63
 make_impute_constant(), 154
 make_impute_uniform_float, 64
 make_impute_uniform_float(), 154
 make_index, 64
 make_index(), 155
 make_is_equal, 65
 make_is_equal(), 155
 make_is_null, 66
 make_is_null(), 156
 make_laplace, 67
 make_laplace(), 156
 make_laplace_threshold, 68
 make_laplace_threshold(), 157
 make_lipschitz_float_mul, 69
 make_lipschitz_float_mul(), 157
 make_mean, 70
 make_mean(), 158
 make_metric_bounded, 71
 make_metric_bounded(), 158
 make_metric_unbounded, 72
 make_metric_unbounded(), 159
 make_noise, 73
 make_noise(), 159
 make_noise_threshold, 74
 make_noise_threshold(), 160
 make_noisy_max, 75
 make_noisy_max(), 160
 make_noisy_top_k, 76
 make_noisy_top_k(), 161
 make_ordered_random, 77
 make_ordered_random(), 161
 make_population_amplification, 78
 make_population_amplification(), 162
 make_privacy_filter, 78
 make_privacy_filter(), 163
 make_private_quantile, 79
 make_private_quantile(), 163
 make_pureDP_to_zCDP, 81
 make_pureDP_to_zCDP(), 164
 make_quantile_score_candidates, 81
 make_quantile_score_candidates(), 164
 make_quantiles_from_counts, 82
 make_quantiles_from_counts(), 165
 make_randomized_response, 83
 make_randomized_response(), 165
 make_randomized_response_bitvec, 84
 make_randomized_response_bitvec(), 166
 make_randomized_response_bool, 85
 make_randomized_response_bool(), 167
 make_report_noisy_max_gumbel, 86
 make_report_noisy_max_gumbel(), 167
 make_resize, 87
 make_resize(), 168
 make_select_column, 88
 make_select_column(), 168
 make_select_private_candidate, 88
 make_select_private_candidate(), 169
 make_sequential_composition, 89
 make_sequential_composition(), 169
 make_sized_bounded_float_checked_sum, 91
 make_sized_bounded_float_checked_sum(), 170
 make_sized_bounded_float_ordered_sum, 92
 make_sized_bounded_float_ordered_sum(), 170
 make_sized_bounded_int_checked_sum, 93
 make_sized_bounded_int_checked_sum(), 171
 make_sized_bounded_int_monotonic_sum, 94
 make_sized_bounded_int_monotonic_sum(),

- [171](#)
- [make_sized_bounded_int_ordered_sum,](#)
- [95](#)
- [make_sized_bounded_int_ordered_sum\(\),](#)
- [172](#)
- [make_sized_bounded_int_split_sum,](#) [96](#)
- [make_sized_bounded_int_split_sum\(\),](#)
- [172](#)
- [make_split_dataframe,](#) [97](#)
- [make_split_dataframe\(\),](#) [173](#)
- [make_split_lines,](#) [97](#)
- [make_split_lines\(\),](#) [173](#)
- [make_split_records,](#) [98](#)
- [make_split_records\(\),](#) [174](#)
- [make_subset_by,](#) [99](#)
- [make_subset_by\(\),](#) [174](#)
- [make_sum,](#) [99](#)
- [make_sum\(\),](#) [175](#)
- [make_sum_of_squared_deviations,](#) [100](#)
- [make_sum_of_squared_deviations\(\),](#) [175](#)
- [make_unordered,](#) [101](#)
- [make_unordered\(\),](#) [176](#)
- [make_user_measurement,](#) [102](#)
- [make_user_measurement\(\),](#) [176](#)
- [make_user_transformation,](#) [103](#)
- [make_user_transformation\(\),](#) [177](#)
- [make_variance,](#) [105](#)
- [make_variance\(\),](#) [178](#)
- [make_zCDP_to_approxDP,](#) [106](#)
- [make_zCDP_to_approxDP\(\),](#) [178](#)
- [map_domain,](#) [106](#)
- [max_divergence,](#) [107](#)
- [measure_debug,](#) [107](#)
- [measure_distance_type,](#) [108](#)
- [measure_type,](#) [108](#)
- [measurement_check,](#) [109](#)
- [measurement_function,](#) [109](#)
- [measurement_input_carrier_type,](#) [110](#)
- [measurement_input_distance_type,](#) [110](#)
- [measurement_input_domain,](#) [111](#)
- [measurement_input_metric,](#) [111](#)
- [measurement_invoke,](#) [112](#)
- [measurement_map,](#) [112](#)
- [measurement_output_distance_type,](#) [113](#)
- [measurement_output_measure,](#) [113](#)
- [metric_debug,](#) [114](#)
- [metric_distance_type,](#) [114](#)
- [metric_type,](#) [115](#)
- [new_domain,](#) [115](#)
- [new_function,](#) [116](#)
- [new_function_internal,](#) [116](#)
- [new_hashtab,](#) [117](#)
- [new_measure,](#) [117](#)
- [new_measurement,](#) [118](#)
- [new_metric,](#) [118](#)
- [new_odometer,](#) [118](#)
- [new_odometer_queryable_internal,](#) [119](#)
- [new_privacy_profile,](#) [119](#)
- [new_privacy_profile_internal,](#) [120](#)
- [new_queryable,](#) [120](#)
- [new_queryable_internal,](#) [121](#)
- [new_transformation,](#) [121](#)
- [object_type,](#) [122](#)
- [odometer_input_carrier_type,](#) [122](#)
- [odometer_input_domain,](#) [123](#)
- [odometer_input_metric,](#) [123](#)
- [odometer_invoke,](#) [124](#)
- [odometer_output_measure,](#) [124](#)
- [odometer_queryable_invoke,](#) [125](#)
- [odometer_queryable_invoke_type,](#) [125](#)
- [odometer_queryable_privacy_loss,](#) [126](#)
- [odometer_queryable_privacy_loss_type,](#)
- [126](#)
- [opendp,](#) [127](#)
- [opendp-package \(*opendp*\),](#) [127](#)
- [option_domain,](#) [128](#)
- [parse_or_infer,](#) [128](#)
- [privacy_profile_delta,](#) [129](#)
- [privacy_profile_epsilon,](#) [129](#)
- [queryable_eval,](#) [130](#)
- [queryable_query_type,](#) [130](#)
- [renyi_divergence,](#) [131](#)
- [rt_infer,](#) [131](#)
- [rt_parse,](#) [132](#)
- [smoothed_max_divergence,](#) [132](#)
- [String,](#) [133](#)
- [symmetric_distance,](#) [133](#)
- [then_adaptive_composition,](#) [134](#)
- [then_alp_queryable,](#) [134](#)
- [then_approximate,](#) [135](#)
- [then_b_ary_tree,](#) [136](#)
- [then_basic_composition,](#) [136](#)

then_bounded_float_checked_sum, 137
 then_bounded_float_ordered_sum, 137
 then_bounded_int_monotonic_sum, 138
 then_bounded_int_ordered_sum, 138
 then_bounded_int_split_sum, 139
 then_canonical_noise, 139
 then_cast, 140
 then_cast_default, 140
 then_cast_inherent, 141
 then_cdf, 141
 then_chain_mt, 142
 then_chain_pm, 142
 then_chain_tt, 143
 then_clamp, 143
 then_composition, 144
 then_consistent_b_ary_tree, 144
 then_count, 145
 then_count_by, 145
 then_count_by_categories, 146
 then_count_distinct, 146
 then_create_dataframe, 147
 then_df_cast_default, 147
 then_df_is_equal, 148
 then_drop_null, 148
 then_find, 149
 then_find_bin, 149
 then_fix_delta, 150
 then_fixed_approxDP_to_approxDP, 150
 then_fully_adaptive_composition, 151
 then_gaussian, 151
 then_gaussian_threshold, 152
 then_geometric, 153
 then_identity, 153
 then_impute_constant, 154
 then_impute_uniform_float, 154
 then_index, 155
 then_is_equal, 155
 then_is_null, 156
 then_laplace, 156
 then_laplace_threshold, 157
 then_lipschitz_float_mul, 157
 then_mean, 158
 then_metric_bounded, 158
 then_metric_unbounded, 159
 then_noise, 159
 then_noise_threshold, 160
 then_noisy_max, 160
 then_noisy_top_k, 161
 then_ordered_random, 161
 then_population_amplification, 162
 then_postprocess, 162
 then_privacy_filter, 163
 then_private_quantile, 163
 then_pureDP_to_zCDP, 164
 then_quantile_score_candidates, 164
 then_quantiles_from_counts, 165
 then_randomized_response, 165
 then_randomized_response_bitvec, 166
 then_randomized_response_bool, 167
 then_report_noisy_max_gumbel, 167
 then_resize, 168
 then_select_column, 168
 then_select_private_candidate, 169
 then_sequential_composition, 169
 then_sized_bounded_float_checked_sum, 170
 then_sized_bounded_float_ordered_sum, 170
 then_sized_bounded_int_checked_sum, 171
 then_sized_bounded_int_monotonic_sum, 171
 then_sized_bounded_int_ordered_sum, 172
 then_sized_bounded_int_split_sum, 172
 then_split_dataframe, 173
 then_split_lines, 173
 then_split_records, 174
 then_subset_by, 174
 then_sum, 175
 then_sum_of_squared_deviations, 175
 then_unordered, 176
 then_user_measurement, 176
 then_user_transformation, 177
 then_variance, 178
 then_zCDP_to_approxDP, 178
 to_str.default, 179
 to_str.hashtab, 179
 to_then, 180
 transformation_check, 180
 transformation_function, 181
 transformation_input_carrier_type, 181
 transformation_input_distance_type, 182
 transformation_input_domain, 182

- transformation_input_metric, [183](#)
- transformation_invoke, [183](#)
- transformation_map, [184](#)
- transformation_output_distance_type,
 [184](#)
- transformation_output_domain, [185](#)
- transformation_output_metric, [185](#)

- u128, [186](#)
- u16, [186](#)
- u32, [186](#)
- u64, [186](#)
- u8, [187](#)
- user_distance, [187](#)
- user_divergence, [188](#)
- user_domain, [188](#)
- usize, [189](#)

- vector_domain, [189](#)

- zero_concentrated_divergence, [190](#)