Package 'data.sketches'

July 9, 2026

Title Probabilistic Streaming Data Sketches

Version 0.1.0

Description Provides an interface to the 'Apache DataSketches' (<https://datasketches.apache.org/>) library of streaming algorithms for approximate analytics on data too large to hold or process exactly. Sketches are compact, mergeable summaries built in a single pass over a stream that answer queries such as approximate distinct counts, quantiles and ranks, frequent items and point-frequency estimates, weighted sampling, and set membership with mathematically proven error bounds. Implements Karnin-Lang-Liberty (KLL), Relative Error Quantiles (REQ), t-Digest, HyperLogLog (HLL), Compressed Probabilistic Counting (CPC), Theta, Frequent Items, Count-Min, Array of Doubles, Variance Optimal (VarOpt), Exact and Bounded Probabilistic Proportional-to-Size (EBPPS), and Bloom filter sketches, with native serialization for interoperability with other 'Apache DataSketches' implementations.

License MIT + file LICENSE

Copyright file inst/COPYRIGHTS

URL https://github.com/pedrobtz/data.sketches, https://pedrobtz.github.io/data.sketches/

BugReports https://github.com/pedrobtz/data.sketches/issues

Depends R (>= 4.1.0)

Imports R6, rlang

Suggests testthat (>= 3.0.0)

LinkingTo cpp11

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

SystemRequirements C++17

NeedsCompilation yes

Author Pedro Baltazar [aut, cre, cph],
The Apache Software Foundation [ctb] (Author of bundled Apache DataSketches C++ code),
Stephan Brumme [ctb] (Author of bundled xxhash64.h code),
Austin Appleby [ctb] (Author of bundled public-domain MurmurHash3 code),
Sean Eron Anderson [ctb] (Author of bundled public-domain bit-hack code used in ceiling_power_of_2.hpp)

Maintainer Pedro Baltazar <pedrobtz@gmail.com>

Repository CRAN

Date/Publication 2026-07-09 09:00:02 UTC

Contents

`array_of_doubles` *Array of Doubles (Tuple) sketch for estimating sums alongside distinct counts*

Description

Creates an Array of Doubles sketch, a Tuple sketch that extends a `theta()` sketch by associating a fixed-length array of `num_values` doubles with each retained key. It estimates not only the number of distinct keys (`$estimate()`, as for `theta()`) but also the sum of each value column over the full input stream (`$column_sums()`), e.g. to estimate the total of a numeric measure across distinct users.

Usage

```
array_of_doubles(
  x = NULL,
  values = NULL,
  lg_k = NULL,
  num_values = NULL,
  seed = NULL,
  bytes = NULL
)
```

Arguments

x	Optional numeric or character vector of keys to update the new sketch with. Each element is hashed and contributes to the distinct-count estimate.
values	Optional value(s) associated with each element of x: a numeric vector (when num_values == 1) or a numeric matrix with num_values columns, recycled to length(x) rows if a single value/row is supplied. Defaults to 1s (so $column_sums() estimates the count of each key, like $estimate()). Cannot be set without x.
lg_k	log2 of the nominal number of entries, a single whole number in [5, 26]. Larger lg_k is more accurate and larger. Defaults to 12 (resolved when a fresh sketch is built). Must not be set when bytes is supplied.
num_values	Number of double values associated with each retained key, a single whole number in [1, 255]. Defaults to 1. Must not be set when bytes is supplied.
seed	Hash seed, a single non-negative whole number up to 2^53. Defaults to 9001 (the upstream default), resolved whether or not bytes is supplied.
bytes	Optional raw vector holding a native serialized sketch to reconstruct. The result is always a compact sketch.

Details

At most one of x or bytes may be supplied:

- Pass x to build a sketch and immediately update it with a numeric or character vector of keys (optionally with values).
- Pass bytes to reconstruct a sketch from a native serialized payload (as produced by sketch$serialize()). The result is always a *compact* sketch (see below); lg_k and num_values must not be supplied alongside bytes. Unlike lg_k, the hash seed is *not* stored in the payload and must be supplied if the original sketch did not use the default.
- Pass neither for an empty (mutable) sketch with the given lg_k, num_values, and seed.

update() silently ignores NA/NaN/NA_character_ in x (and the corresponding row of values), matching the missing-value policy used across families; there is no na_rm argument.

An Array of Doubles sketch is either an *update* sketch (mutable, $is_compact() is FALSE) or a *compact* sketch (immutable, $is_compact() is TRUE). Fresh sketches built from x/lg_k are update sketches and can be grown with $update(). Compact sketches arise from bytes = reconstruction, $merge(), or any of the array_of_doubles_*() set operations, and cannot be updated further. $lg_k() is only defined for update sketches.

Two sketches can only be merged with `$merge()`, or combined with `array_of_doubles_union()` / `array_of_doubles_intersection()`, if they share the same `seed` (a mismatch raises `datasketches_seed_mismatch`) and the same `num_values` (a mismatch raises `datasketches_incompatible_sketch`). Value arrays for matching keys are combined by element-wise sum. `$merge()` mutates the receiver into a compact sketch holding the union of both inputs (so it can no longer be `$update()`d afterward).

Value

An `array_of_doubles_sketch` object. Key methods:

`$update(x, values = NULL)` Add keys with associated values (mutates, returns the sketch). Errors if the sketch is compact.

`$merge(other)` Absorb another sketch with the same `seed` and `num_values`, becoming compact (mutates, returns the sketch).

`$estimate()` Approximate number of distinct keys seen.

`$lower_bound(num_std_dev = 1)` / `$upper_bound(num_std_dev = 1)` Approximate confidence bounds on `estimate()`, at 1, 2, or 3 standard deviations.

`$column_sums()` Estimated sum of each value column over the full input stream.

`$lg_k()`, `$num_values()`, `$seed()`, `$theta()`, `$num_retained()`, `$is_empty()`, `$is_estimation_mode()`, `$is_ordere` Metadata accessors.

`$summary()`, `$inspect()`, `$serialize()` Structured metadata, verbose debug output, and the native byte payload.

Examples

```
keys <- sample(1000, 5000, replace = TRUE)
values <- runif(length(keys))
sketch <- array_of_doubles(keys, values)
sketch$estimate()
sketch$column_sums()

# Round-trip through the native byte format (always compact).
restored <- array_of_doubles(bytes = sketch$serialize())
restored$is_compact()
identical(restored$column_sums(), sketch$column_sums())
```

`array_of_doubles_set_operations`

Array of Doubles sketch set operations

Description

Combine two `array_of_doubles()` sketches into a new compact `array_of_doubles_sketch` result, without mutating either input. `a` and `b` must be Array of Doubles sketches created with the same `seed` (a mismatch raises `datasketches_seed_mismatch`).

Usage

```
array_of_doubles_union(a, b, lg_k = NULL)

array_of_doubles_intersection(a, b)

array_of_doubles_difference(a, b)
```

Arguments

a, b	array_of_doubles_sketch objects created with the same seed.
lg_k	For array_of_doubles_union(), log2 of the nominal number of entries for the union's internal sketch, a single whole number in [5, 26]. Defaults to the larger of a and b's configured lg_k (or 12 for compact inputs).

Details

- array_of_doubles_union(a, b) estimates the size of the union union(A, B). a and b must also share the same num_values (a mismatch raises datasketches_incompatible_sketch); value arrays for matching keys are combined by element-wise sum.
- array_of_doubles_intersection(a, b) estimates the size of the intersection intersection(A, B), with the same num_values requirement and combining rule as array_of_doubles_union().
- array_of_doubles_difference(a, b) estimates the size of the set difference A \\ B (elements in A but not B), retaining a's value arrays unchanged for the retained keys.

Value

A compact array_of_doubles_sketch object.

Examples

```
a <- array_of_doubles(1:1000, runif(1000))
b <- array_of_doubles(501:1500, runif(1000))
array_of_doubles_union(a, b)$column_sums()
array_of_doubles_intersection(a, b)$estimate()
array_of_doubles_difference(a, b)$estimate()
```

---

bloom_filter *Bloom filter for approximate set membership*

Description

Creates a Bloom filter, a probabilistic data structure for approximate set membership. Querying an item that has been added always returns TRUE (no false negatives); querying an item that has never been added may return TRUE with probability up to the configured false-positive probability.

Usage

```
bloom_filter(
  x = NULL,
  max_items = NULL,
  fpp = NULL,
  num_bits = NULL,
  num_hashes = NULL,
  seed = NULL,
  bytes = NULL
)
```

Arguments

x	Optional numeric or character vector of items to update the new filter with.
max_items	Target maximum number of distinct items, a single positive whole number up to 2^53. Must be supplied together with fpp, and cannot be combined with num_bits/num_hashes. Must not be set when bytes is supplied.
fpp	Target false-positive probability, a single number in (0, 1]. Must be supplied together with max_items. Must not be set when bytes is supplied.
num_bits	Number of bits in the filter, a single positive whole number up to 2^53. Must be supplied together with num_hashes, and cannot be combined with max_items/fpp. Must not be set when bytes is supplied.
num_hashes	Number of hash functions applied per item, a single whole number in [1, 65535]. Must be supplied together with num_bits. Must not be set when bytes is supplied.
seed	Hash seed, a single non-negative whole number up to 2^53. Defaults to 9001. Two filters can only be combined if their seed (and num_hashes and capacity) match. Must not be set when bytes is supplied.
bytes	Optional raw vector holding a native serialized filter to reconstruct.

Details

Unlike the other sketch families, a Bloom filter is not sub-linear in size: it is sized up front and does not resize itself. There are two sizing strategies, which cannot be combined:

- max_items and fpp size the filter for a target number of distinct items and a target false-positive probability.
- num_bits and num_hashes size the filter explicitly.

If neither strategy is specified, the filter defaults to max_items = 10000 and fpp = 0.01.

At most one of x or bytes may be supplied:

- Pass x to build a filter and immediately update it with a numeric or character vector of items.
- Pass bytes to reconstruct a filter from a native serialized payload (as produced by filter$serialize()). max_items, fpp, num_bits, num_hashes, and seed must not be supplied alongside bytes; they are restored from the payload.

- Pass neither for an empty (mutable) filter with the given sizing.

`update()`, `query()`, and `query_and_update()` silently ignore (or return `NA` for) `NA`/`NaN`/`NA_character_` in `x`, matching the missing-value policy used across families.

Two filters can only be combined with `$merge()` (logical OR) or `$intersect()` (logical AND) if they are "compatible": they share the same `seed`, `num_hashes`, and `capacity` (a mismatch raises `datasketches_incompatible_sketch`).

Value

A `bloom_filter` object. Key methods:

`$update(x)` Add items (mutates, returns the filter).

`$query(x)` Logical vector: might each element have been seen?

`$query_and_update(x)` `$query()` against the prior state, then `$update()` (mutates, returns the query result).

`$merge(other)` In-place logical OR with a compatible filter (mutates, returns the filter).

`$intersect(other)` In-place logical AND with a compatible filter (mutates, returns the filter).

`$invert()` In-place logical NOT (mutates, returns the filter).

`$reset()` Clear all bits, keeping sizing and `seed` (mutates, returns the filter).

`$is_compatible(other)` Whether `other` may be combined with this filter.

`$capacity()`, `$num_hashes()`, `$seed()`, `$bits_used()`, `$is_empty()` Metadata accessors.

`$summary()`, `$inspect()`, `$serialize()` Structured metadata, verbose debug output, and the native byte payload.

Examples

```
bf <- bloom_filter(letters, max_items = 1000, fpp = 0.01)
bf$query(c("a", "z", "!"))

# Round-trip through the native byte format.
restored <- bloom_filter(bytes = bf$serialize())
restored$query("a")
```

---

bloom_filter_suggest *Suggest Bloom filter sizing parameters*

Description

Helpers that translate a target accuracy into Bloom filter constructor arguments for the `num_bits`/`num_hashes` sizing strategy. These compute the same values that `bloom_filter()` uses internally for the `max_items`/`fpp` sizing strategy, for callers who want to inspect or reuse them (for example, to create multiple compatible filters with an explicit `seed`).

Usage

```
bloom_filter_suggest_num_filter_bits(max_items, fpp)

bloom_filter_suggest_num_hashes(max_items, num_bits)
```

Arguments

max_items	Target maximum number of distinct items, a single positive whole number up to 2^53.
fpp	Target false-positive probability, a single number in (0, 1].
num_bits	Number of bits in the filter, a single positive whole number up to 2^53.

Value

A single number: bloom_filter_suggest_num_filter_bits() returns the suggested num_bits (a double, which may exceed .Machine$integer.max); bloom_filter_suggest_num_hashes() returns the suggested num_hashes (an integer).

Examples

```
num_bits <- bloom_filter_suggest_num_filter_bits(1000, 0.01)
num_hashes <- bloom_filter_suggest_num_hashes(1000, num_bits)
bf <- bloom_filter(num_bits = num_bits, num_hashes = num_hashes)
```

---

count_min *Count-Min sketch for approximate point-frequency estimation*

Description

Creates a Count-Min sketch, a mergeable summary that estimates the frequency (sum of weights) of individual items in a numeric or character stream far larger than memory, with one-sided error: $estimate() never under-estimates the true frequency.

Usage

```
count_min(
  x = NULL,
  weight = NULL,
  num_hashes = NULL,
  num_buckets = NULL,
  seed = NULL,
  bytes = NULL
)
```

Arguments

x	Optional numeric or character vector to update the new sketch with.
weight	Optional weight(s) for x: a single finite number (recycled, may be negative or fractional), or a vector of such values matching the length of x. Defaults to 1 (each occurrence counts once). Cannot be set without x.
num_hashes	Number of hash functions, a single whole number in [1, 255]. Larger values increase confidence but also memory use. Defaults to 3. Must not be set when bytes is supplied. See count_min_suggest_num_hashes().
num_buckets	Number of buckets per hash function, a single whole number of at least 3 (and such that num_buckets * num_hashes < 2^30). Larger values are more accurate and larger. Defaults to 55. Must not be set when bytes is supplied. See count_min_suggest_num_buckets().
seed	Hash seed, a single non-negative whole number up to 2^53. Defaults to 9001. Two sketches can only be merged if their seed (and num_hashes and num_buckets) match.
bytes	Optional raw vector holding a native serialized sketch to reconstruct.

Details

At most one of x or bytes may be supplied:

- Pass x to build a sketch and immediately update it with a numeric or character vector (optionally with weight).
- Pass bytes to reconstruct a sketch from a native serialized payload (as produced by sketch$serialize()). num_hashes and num_buckets are restored from the payload and must not be supplied alongside bytes.
- Pass neither for an empty sketch with the given num_hashes and num_buckets.

Numeric items are hashed via the raw bytes of their IEEE-754 double representation; this is internally consistent between update() and the estimate()/lower_bound()/upper_bound() queries, but is not guaranteed to match hashes produced by other DataSketches language implementations for the same numeric value.

NA/NaN/NA_character_ are silently ignored by update(), matching the missing-value policy used across families; there is no na_rm argument.

Two sketches can only be $merge()d if they share the same num_hashes, num_buckets, and seed; a mismatch raises datasketches_incompatible_sketch.

Value

A count_min_sketch object. Key methods:

$update(x, weight = NULL) Add numeric or character values with an optional weight (mutates, returns the sketch).

$merge(other) Absorb another sketch with matching num_hashes, num_buckets, and seed (mutates, returns the sketch).

`$estimate(item)`, `$lower_bound(item)`, `$upper_bound(item)` Estimated frequency and guaranteed bounds for one or more items.

`$total_weight()`, `$relative_error()`, `$num_hashes()`, `$num_buckets()`, `$seed()`, `$is_empty()` Metadata accessors.

`$summary()`, `$inspect()`, `$serialize()` Structured metadata, verbose debug output, and the native byte payload.

Examples

```
words <- sample(letters[1:5], 1000, replace = TRUE, prob = c(.5, .25, .1, .1, .05))
sketch <- count_min(words)
sketch$estimate("a")
sketch$relative_error()

# Round-trip through the native byte format.
restored <- count_min(bytes = sketch$serialize())
identical(restored$total_weight(), sketch$total_weight())
```

`count_min_suggest` *Suggest Count-Min sketch parameters*

Description

Helpers to translate a desired accuracy into the `num_buckets` and `num_hashes` arguments of `count_min()`.

Usage

```
count_min_suggest_num_buckets(relative_error)

count_min_suggest_num_hashes(confidence)
```

Arguments

`relative_error`	Desired relative error, a single positive number. `count_min_suggest_num_buckets()` returns the smallest `num_buckets` such that the sketch's `$relative_error()` does not exceed this value.
`confidence`	Desired confidence, a single number in `(0, 1]`. `count_min_suggest_num_hashes()` returns the smallest `num_hashes` such that, with this probability, `$estimate()` is within `$relative_error()` of the true frequency.

Value

A single integer.

Examples

```
num_buckets <- count_min_suggest_num_buckets(0.05)
num_hashes <- count_min_suggest_num_hashes(0.95)
sketch <- count_min(num_hashes = num_hashes, num_buckets = num_buckets)
```

---

cpc *CPC sketch for approximate distinct counting*

Description

Creates a CPC (Compressed Probabilistic Counting) sketch, a very compact, mergeable summary that estimates the number of distinct values seen in a stream far larger than memory. CPC sketches are similar in purpose to `hll()` but serialize to a smaller payload, at the cost of slightly higher CPU use.

Usage

```
cpc(x = NULL, lg_k = NULL, seed = NULL, bytes = NULL)
```

Arguments

`x`	Optional numeric or character vector to update the new sketch with. Each element is hashed and contributes to the distinct-count estimate.
`lg_k`	log2 of the number of bins, a single whole number in `[4, 26]`. Larger `lg_k` is more accurate and larger. Defaults to 11 (resolved when a fresh sketch is built). Must not be set when `bytes` is supplied.
`seed`	Hash seed, a single non-negative whole number up to `2^53`. Defaults to `9001` (the upstream default), resolved whether or not `bytes` is supplied.
`bytes`	Optional raw vector holding a native serialized sketch to reconstruct.

Details

At most one of `x` or `bytes` may be supplied:

- Pass `x` to build a sketch and immediately update it with a numeric or character vector.
- Pass `bytes` to reconstruct a sketch from a native serialized payload (as produced by `sketch$serialize()`). `lg_k` is restored from the payload and must not be supplied alongside `bytes`. Unlike `lg_k`, the hash `seed` is *not* stored in the payload and must be supplied if the original sketch did not use the default.
- Pass neither for an empty sketch with the given `lg_k` and `seed`.

`update()` silently ignores `NA`/`NaN`/`NA_character_`, matching the missing-value policy used across families; there is no `na_rm` argument.

Two sketches can only be merged with `$merge()` if they share the same `seed`; a mismatch raises `datasketches_seed_mismatch`.

Value

A cpc_sketch object. Key methods:

$update(x) Add numeric or character values (mutates, returns the sketch).

$merge(other) Absorb another sketch with the same seed (mutates, returns the sketch).

$estimate() Approximate number of distinct values seen.

$lower_bound(num_std_dev = 1) / $upper_bound(num_std_dev = 1) Approximate confidence bounds on estimate(), at 1, 2, or 3 standard deviations.

$lg_k(), $seed(), $is_empty() Metadata accessors.

$summary(), $inspect(), $serialize() Structured metadata, verbose debug output, and the native byte payload.

Examples

```
sketch <- cpc(sample(1000, 5000, replace = TRUE))
sketch$estimate()
sketch$lower_bound()
sketch$upper_bound()

# Round-trip through the native byte format.
restored <- cpc(bytes = sketch$serialize())
identical(restored$estimate(), sketch$estimate())
```

---

ebpps *EBPPS sketch for proportional-to-size sampling*

Description

Creates an EBPPS (Exact and Bounded Probabilistic Proportional-to-Size) sketch, which samples up to k items from a stream of weighted (item, weight) pairs. It is a modern alternative to classic reservoir sampling: each item's inclusion probability is proportional to its share of the total stream weight, with a tight bound on the resulting sample size.

Usage

```
ebpps(x = NULL, weight = NULL, k = NULL, type = NULL, bytes = NULL)
```

Arguments

x	Optional numeric or character vector of items to update the new sketch with.
weight	Optional weight(s) for each element of x: a single non-negative, finite number, or a vector of such values matching length(x). Defaults to 1. Cannot be set without x.
k	Maximum sample size, a single whole number in [1, 2^31 - 2]. Defaults to 256. Must not be set when bytes is supplied.

type	Item type for a fresh sketch, either "double" or "character". Defaults to the type of x (or "double" if x is not supplied). Must not be set when bytes is supplied.
bytes	Optional raw vector holding a native serialized sketch to reconstruct.

Details

Unlike the hash-based cardinality and frequency sketches, EBPPS retains items verbatim rather than hashing them, so the item type (numeric or character) is fixed when the sketch is created and cannot change.

At most one of x or bytes may be supplied:

- Pass x to build a sketch and immediately update it with a numeric or character vector of items (optionally with weight). The item type is inferred from x unless type is supplied.
- Pass bytes to reconstruct a sketch from a native serialized payload (as produced by sketch$serialize()). weight, k, and type must not be supplied alongside bytes; they are restored from the payload.
- Pass neither for an empty (mutable) sketch with the given k and type.

update() silently ignores NA/NaN/NA_character_ in x (and the corresponding weight), matching the missing-value policy used across families; there is no na_rm argument.

Two sketches can only be merged with $merge() if they hold the same item type (a mismatch raises datasketches_incompatible_sketch). The merged sketch is resized to the smaller of the two inputs' configured k, matching the native implementation.

Value

An ebpps_sketch object. Key methods:

$update(x, weight = NULL) Add weighted items (mutates, returns the sketch).

$merge(other) Absorb another sketch with the same item type (mutates, returns the sketch).

$result() The current sample as a numeric or character vector.

$k(), $n(), $cumulative_weight(), $c(), $is_empty(), $is_character() Metadata accessors.

$summary(), $inspect(), $serialize() Structured metadata, verbose debug output, and the native byte payload.

Examples

```
items <- 1:1000
weights <- runif(1000)
sketch <- ebpps(items, weights, k = 50)
sketch$result()
sketch$c()

# Round-trip through the native byte format.
restored <- ebpps(bytes = sketch$serialize())
restored$k()
```

`frequent_items` *Frequent Items sketch for approximate frequency estimation*

Description

Creates a Frequent Items sketch, a mergeable summary that estimates the frequencies of the most frequent items in a character stream far larger than memory, with guaranteed error bounds.

Usage

```
frequent_items(
  x = NULL,
  weight = NULL,
  lg_max_map_size = NULL,
  lg_start_map_size = NULL,
  bytes = NULL
)
```

Arguments

`x`	Optional character vector to update the new sketch with.
`weight`	Optional weight(s) for `x`: a single non-negative whole number (recycled), or a vector of such values matching the length of `x`. Defaults to `1` (each occurrence counts once). Cannot be set without `x`.
`lg_max_map_size`	log2 of the maximum size of the internal hash map, a single whole number in `[3, 30]`. Larger values are more accurate and larger. Defaults to `8`. Must not be set when `bytes` is supplied.
`lg_start_map_size`	log2 of the starting size of the internal hash map, a single whole number in `[3, lg_max_map_size]`. Defaults to `3`. Must not be set when `bytes` is supplied.
`bytes`	Optional raw vector holding a native serialized sketch to reconstruct.

Details

At most one of `x` or `bytes` may be supplied:

- Pass `x` to build a sketch and immediately update it with a character vector (optionally with `weight`).
- Pass `bytes` to reconstruct a sketch from a native serialized payload (as produced by `sketch$serialize()`). `lg_max_map_size` and `lg_start_map_size` are restored from the payload and must not be supplied alongside `bytes`.
- Pass neither for an empty sketch with the given `lg_max_map_size` and `lg_start_map_size`.

`NA_character_` is silently ignored by `update()`, matching the missing-value policy used across families; there is no `na_rm` argument.

Value

A `frequent_items_sketch` object. Key methods:

`$update(x, weight = NULL)` Add character values with an optional weight (mutates, returns the sketch).

`$merge(other)` Absorb another sketch (mutates, returns the sketch).

`$estimate(item)`, `$lower_bound(item)`, `$upper_bound(item)` Estimated frequency and guaranteed bounds for one or more items.

`$frequent_items(error_type = "no_false_positives", threshold = NULL)` A data frame of items whose estimated frequency exceeds `threshold` (defaults to `$maximum_error()`), with columns `item`, `estimate`, `lower_bound`, and `upper_bound`.

`$maximum_error()`, `$epsilon()`, `$total_weight()`, `$num_active_items()`, `$is_empty()` Metadata accessors.

`$summary()`, `$inspect()`, `$serialize()` Structured metadata, verbose debug output, and the native byte payload.

Examples

```
words <- sample(letters[1:5], 1000, replace = TRUE, prob = c(.5, .25, .1, .1, .05))
sketch <- frequent_items(words)
sketch$frequent_items()
sketch$estimate("a")

# Round-trip through the native byte format.
restored <- frequent_items(bytes = sketch$serialize())
identical(restored$total_weight(), sketch$total_weight())
```

`hll` *HLL sketch for approximate distinct counting*

Description

Creates an HLL (HyperLogLog) sketch, a compact, mergeable summary that estimates the number of distinct values seen in a stream far larger than memory.

Usage

```
hll(x = NULL, lg_k = NULL, type = NULL, bytes = NULL)
```

Arguments

`x`	Optional numeric or character vector to update the new sketch with. Each element is hashed and contributes to the distinct-count estimate.
`lg_k`	log2 of the number of buckets, a single whole number in `[4, 21]`. Larger `lg_k` is more accurate and larger. Defaults to `12` (resolved when a fresh sketch is built). Must not be set when `bytes` is supplied.

type	One of "HLL_4", "HLL_6", or "HLL_8", controlling the per-bucket encoding width (a size/speed trade-off that does not affect accuracy). Defaults to "HLL_4" (resolved when a fresh sketch is built). Must not be set when bytes is supplied.
bytes	Optional raw vector holding a native serialized sketch to reconstruct.

Details

At most one of x or bytes may be supplied:

- Pass x to build a sketch and immediately update it with a numeric or character vector.
- Pass bytes to reconstruct a sketch from a native serialized payload (as produced by sketch$serialize()). Configuration is restored from the payload, so lg_k and type must not be supplied alongside bytes.
- Pass neither for an empty sketch with the given lg_k and type.

update() silently ignores NA/NaN/NA_character_, matching the missing-value policy used across families; there is no na_rm argument.

Value

An hll_sketch object. Key methods:

$update(x) Add numeric or character values (mutates, returns the sketch).

$merge(other) Absorb another sketch (mutates, returns the sketch).

$estimate() Approximate number of distinct values seen.

$lower_bound(num_std_dev = 1) / $upper_bound(num_std_dev = 1) Approximate confidence bounds on estimate(), at 1, 2, or 3 standard deviations.

$lg_k(), $hll_type(), $is_empty(), $is_compact() Metadata accessors.

$summary(), $inspect(), $serialize() Structured metadata, verbose debug output, and the native byte payload.

Examples

```
sketch <- hll(sample(1000, 5000, replace = TRUE))
sketch$estimate()
sketch$lower_bound()
sketch$upper_bound()

# Round-trip through the native byte format.
restored <- hll(bytes = sketch$serialize())
identical(restored$estimate(), sketch$estimate())
```

kll_doubles *KLL sketch for approximate quantiles of a numeric stream*

Description

Creates a KLL quantile sketch over double values. A KLL sketch is a compact, mergeable summary that answers approximate quantile, rank, CDF, and PMF queries over a stream far larger than memory, with a configurable accuracy/size trade-off controlled by k.

Usage

```
kll_doubles(x = NULL, k = NULL, bytes = NULL)
```

Arguments

x	Optional numeric vector to update the new sketch with.
k	Sketch width controlling the accuracy/size trade-off, a whole number in [8, 65535]. Larger k is more accurate and larger. Defaults to 200 (resolved when a fresh sketch is built). Must not be set when bytes is supplied.
bytes	Optional raw vector holding a native serialized sketch to reconstruct.

Details

At most one of x or bytes may be supplied:

- Pass x to build a sketch and immediately update it with a numeric vector.
- Pass bytes to reconstruct a sketch from a native serialized payload (as produced by sketch$serialize()). The width is restored from the payload, so k must not be supplied alongside bytes.
- Pass neither for an empty sketch of width k.

update() silently ignores NA/NaN, matching the upstream/Python behaviour; there is no na_rm argument.

Value

A kll_doubles_sketch object. Key methods:

$update(x) Add numeric values (mutates, returns the sketch).

$merge(other) Absorb another sketch (mutates, returns the sketch).

$quantile(probs, inclusive = TRUE) Approximate quantiles for probabilities in [0, 1].

$rank(x, inclusive = TRUE) Approximate ranks of x; missing inputs return NA.

$cdf(split_points) / $pmf(split_points) Cumulative / mass estimates; return length(split_points) + 1 values.

$n(), $k(), $num_retained(), $is_empty(), $is_estimation_mode(), $min(), $max(), $rank_error(pmf = FALSE) Metadata and accuracy accessors.

$summary(), $inspect(), $serialize() Structured metadata, verbose debug output, and the native byte payload.

Examples

```
sketch <- kll_doubles(rnorm(10000))
sketch$quantile(c(0.25, 0.5, 0.75))
sketch$rank(c(-1, 0, 1))

# Round-trip through the native byte format.
restored <- kll_doubles(bytes = sketch$serialize())
identical(restored$quantile(0.5), sketch$quantile(0.5))
```

`kll_floats` *KLL sketch for approximate quantiles of a numeric stream stored as floats*

Description

Creates a KLL quantile sketch over `float` (32-bit) values. A KLL sketch is a compact, mergeable summary that answers approximate quantile, rank, CDF, and PMF queries over a stream far larger than memory, with a configurable accuracy/size trade-off controlled by `k`.

Usage

```
kll_floats(x = NULL, k = NULL, bytes = NULL)
```

Arguments

`x`	Optional numeric vector to update the new sketch with.
`k`	Sketch width controlling the accuracy/size trade-off, a whole number in `[8, 65535]`. Larger `k` is more accurate and larger. Defaults to `200` (resolved when a fresh sketch is built). Must not be set when `bytes` is supplied.
`bytes`	Optional raw vector holding a native serialized sketch to reconstruct.

Details

Retained items are stored as native 32-bit `float`, not R's 64-bit `double`. Updates and query results are rounded to float precision; use `kll_doubles()` when full double precision is required.

At most one of `x` or `bytes` may be supplied:

- Pass `x` to build a sketch and immediately update it with a numeric vector.
- Pass `bytes` to reconstruct a sketch from a native serialized payload (as produced by `sketch$serialize()`). The width is restored from the payload, so `k` must not be supplied alongside `bytes`.
- Pass neither for an empty sketch of width `k`.

`update()` silently ignores `NA`/`NaN`, matching the upstream/Python behaviour; there is no `na_rm` argument.

Value

A `kll_floats_sketch` object. Key methods:

`$update(x)` Add numeric values (mutates, returns the sketch).

`$merge(other)` Absorb another sketch (mutates, returns the sketch).

`$quantile(probs, inclusive = TRUE)` Approximate quantiles for probabilities in `[0, 1]`.

`$rank(x, inclusive = TRUE)` Approximate ranks of `x`; missing inputs return `NA`.

`$cdf(split_points)` / `$pmf(split_points)` Cumulative / mass estimates; return `length(split_points) + 1` values.

`$n()`, `$k()`, `$num_retained()`, `$is_empty()`, `$is_estimation_mode()`, `$min()`, `$max()`, `$rank_error(pmf = FALSE)` Metadata and accuracy accessors.

`$summary()`, `$inspect()`, `$serialize()` Structured metadata, verbose debug output, and the native byte payload.

Examples

```
sketch <- kll_floats(rnorm(10000))
sketch$quantile(c(0.25, 0.5, 0.75))
sketch$rank(c(-1, 0, 1))

# Round-trip through the native byte format.
restored <- kll_floats(bytes = sketch$serialize())
identical(restored$quantile(0.5), sketch$quantile(0.5))
```

`req` *REQ sketch for relative-error approximate quantiles of a numeric stream*

Description

Creates a REQ (Relative Error Quantiles) sketch over `double` values. Like `kll_doubles()`, a REQ sketch is a compact, mergeable summary that answers approximate quantile, rank, CDF, and PMF queries over a stream far larger than memory. Unlike KLL, REQ's accuracy is *relative* and rank-dependent: error is small near the prioritized end of the rank range (controlled by `hra`) and grows towards the other end.

Usage

```
req(x = NULL, k = NULL, hra = NULL, bytes = NULL)
```

Arguments

x	Optional numeric vector to update the new sketch with.
k	Sketch width controlling the accuracy/size trade-off, a single even whole number in [4, 1024]. Larger k is more accurate and larger; k = 12 corresponds to roughly 1% relative error at 95% confidence. Defaults to 12 (resolved when a fresh sketch is built). Must not be set when bytes is supplied.
hra	If TRUE, prioritize accuracy for high ranks (near 1.0); if FALSE, prioritize low ranks (near 0.0). Defaults to TRUE (resolved when a fresh sketch is built). Must not be set when bytes is supplied.
bytes	Optional raw vector holding a native serialized sketch to reconstruct.

Details

At most one of x or bytes may be supplied:

- Pass x to build a sketch and immediately update it with a numeric vector.
- Pass bytes to reconstruct a sketch from a native serialized payload (as produced by sketch$serialize()). Width and hra are restored from the payload, so k and hra must not be supplied alongside bytes.
- Pass neither for an empty sketch of width k.

update() silently ignores NA/NaN, matching the upstream/Python behaviour; there is no na_rm argument.

Value

A req_sketch object. Key methods:

$update(x) Add numeric values (mutates, returns the sketch).

$merge(other) Absorb another sketch (mutates, returns the sketch).

$quantile(probs, inclusive = TRUE) Approximate quantiles for probabilities in [0, 1].

$rank(x, inclusive = TRUE) Approximate ranks of x; missing inputs return NA.

$cdf(split_points) / $pmf(split_points) Cumulative / mass estimates; return length(split_points) + 1 values.

$rank_lower_bound(probs, num_std_dev = 1) / $rank_upper_bound(probs, num_std_dev = 1) Approximate confidence bounds on the rank(s) probs, at 1, 2, or 3 standard deviations.

$n(), $k(), $num_retained(), $is_empty(), $is_estimation_mode(), $min(), $max(), $is_hra() Metadata accessors.

$summary(), $inspect(), $serialize() Structured metadata, verbose debug output, and the native byte payload.

Examples

```
sketch <- req(rnorm(10000))
sketch$quantile(c(0.25, 0.5, 0.75))
sketch$rank(c(-1, 0, 1))
sketch$rank_upper_bound(0.99)

# Round-trip through the native byte format.
restored <- req(bytes = sketch$serialize())
identical(restored$quantile(0.5), sketch$quantile(0.5))
```

---

tdigest_double *t-Digest sketch for approximate quantiles of a numeric stream*

Description

Creates a t-Digest quantile sketch over `double` values. A t-Digest is a compact, mergeable summary that answers approximate quantile, rank, CDF, and PMF queries over a stream far larger than memory. Compared to `kll_doubles()` and `req()`, a t-Digest concentrates its accuracy near the tails of the distribution (extreme quantiles such as p99 or p99.9), at some cost to accuracy near the median.

Usage

```
tdigest_double(x = NULL, k = NULL, bytes = NULL)
```

Arguments

`x`	Optional numeric vector to update the new sketch with.
`k`	Compression parameter controlling the accuracy/size trade-off, a whole number in `[10, 65535]`. Larger `k` is more accurate and larger. Defaults to `200` (resolved when a fresh sketch is built). Must not be set when `bytes` is supplied.
`bytes`	Optional raw vector holding a native serialized sketch to reconstruct.

Details

At most one of `x` or `bytes` may be supplied:

- Pass `x` to build a sketch and immediately update it with a numeric vector.
- Pass `bytes` to reconstruct a sketch from a native serialized payload (as produced by `sketch$serialize()`). The width is restored from the payload, so `k` must not be supplied alongside `bytes`.
- Pass neither for an empty sketch of width `k`.

`update()` silently ignores `NA`/`NaN`, matching the upstream/Python behaviour; there is no `na_rm` argument.

Unlike `kll_doubles()` and `req()`, `$quantile()` and `$rank()` have no `inclusive` argument, and there is no `$rank_error()` accuracy accessor.

Value

A `tdigest_double_sketch` object. Key methods:

`$update(x)` Add numeric values (mutates, returns the sketch).

`$merge(other)` Absorb another sketch (mutates, returns the sketch).

`$quantile(probs)` Approximate quantiles for probabilities in `[0, 1]`.

`$rank(x)` Approximate ranks of `x`; missing inputs return `NA`.

`$cdf(split_points)` / `$pmf(split_points)` Cumulative / mass estimates; return `length(split_points) + 1` values.

`$n()`, `$k()`, `$is_empty()`, `$min()`, `$max()` Metadata accessors.

`$summary()`, `$inspect()`, `$serialize()` Structured metadata, verbose debug output, and the native byte payload.

Examples

```
sketch <- tdigest_double(rnorm(10000))
sketch$quantile(c(0.5, 0.99, 0.999))
sketch$rank(c(-1, 0, 1))

# Round-trip through the native byte format.
restored <- tdigest_double(bytes = sketch$serialize())
identical(restored$quantile(0.5), sketch$quantile(0.5))
```

---

theta *Theta sketch for approximate distinct counting and set operations*

Description

Creates a Theta sketch, a mergeable summary that estimates the number of distinct values seen in a stream far larger than memory. Unlike `hll()` and `cpc()`, Theta sketches natively support set operations: `theta_union()`, `theta_intersection()`, `theta_difference()`, and `theta_jaccard()` combine two sketches into a new result without mutating either input.

Usage

```
theta(x = NULL, lg_k = NULL, seed = NULL, bytes = NULL)
```

Arguments

`x`	Optional numeric or character vector to update the new sketch with. Each element is hashed and contributes to the distinct-count estimate.
`lg_k`	log2 of the nominal number of entries, a single whole number in `[5, 26]`. Larger `lg_k` is more accurate and larger. Defaults to `12` (resolved when a fresh sketch is built). Must not be set when `bytes` is supplied.

seed	Hash seed, a single non-negative whole number up to 2^53. Defaults to 9001 (the upstream default), resolved whether or not bytes is supplied.
bytes	Optional raw vector holding a native serialized sketch to reconstruct. The result is always a compact sketch.

Details

At most one of x or bytes may be supplied:

- Pass x to build a sketch and immediately update it with a numeric or character vector.
- Pass bytes to reconstruct a sketch from a native serialized payload (as produced by sketch$serialize()). The result is always a *compact* sketch (see below); lg_k must not be supplied alongside bytes. Unlike lg_k, the hash seed is *not* stored in the payload and must be supplied if the original sketch did not use the default.
- Pass neither for an empty (mutable) sketch with the given lg_k and seed.

update() silently ignores NA/NaN/NA_character_, matching the missing-value policy used across families; there is no na_rm argument.

A Theta sketch is either an *update* sketch (mutable, $is_compact() is FALSE) or a *compact* sketch (immutable, $is_compact() is TRUE). Fresh sketches built from x/lg_k are update sketches and can be grown with $update(). Compact sketches arise from bytes = reconstruction, $merge(), or any of the theta_*() set operations, and cannot be updated further. $lg_k() is only defined for update sketches.

Two sketches can only be merged with $merge(), or combined with a theta_*() set operation, if they share the same seed; a mismatch raises datasketches_seed_mismatch. $merge() mutates the receiver into a compact sketch holding the union of both inputs (so it can no longer be $update()d afterward).

Value

A theta_sketch object. Key methods:

$update(x) Add numeric or character values (mutates, returns the sketch). Errors if the sketch is compact.

$merge(other) Absorb another sketch with the same seed, becoming compact (mutates, returns the sketch).

$estimate() Approximate number of distinct values seen.

$lower_bound(num_std_dev = 1) / $upper_bound(num_std_dev = 1) Approximate confidence bounds on estimate(), at 1, 2, or 3 standard deviations.

$lg_k(), $seed(), $theta(), $num_retained(), $is_empty(), $is_estimation_mode(), $is_ordered(), $is_compac Metadata accessors.

$summary(), $inspect(), $serialize() Structured metadata, verbose debug output, and the native byte payload.

Examples

```
sketch <- theta(sample(1000, 5000, replace = TRUE))
sketch$estimate()
sketch$lower_bound()
sketch$upper_bound()

# Round-trip through the native byte format (always compact).
restored <- theta(bytes = sketch$serialize())
restored$is_compact()
identical(restored$estimate(), sketch$estimate())

# Set operations.
a <- theta(1:1000)
b <- theta(501:1500)
theta_union(a, b)$estimate()
theta_intersection(a, b)$estimate()
theta_difference(a, b)$estimate()
theta_jaccard(a, b)
```

---

theta_set_operations *Theta sketch set operations*

Description

Combine two theta() sketches into a new compact theta_sketch result, without mutating either input. a and b must be Theta sketches created with the same seed (a mismatch raises datasketches_seed_mismatch).

Usage

```
theta_union(a, b, lg_k = NULL)

theta_intersection(a, b)

theta_difference(a, b)

theta_jaccard(a, b)
```

Arguments

a, b	theta_sketch objects created with the same seed.
lg_k	For theta_union(), log2 of the nominal number of entries for the union's internal sketch, a single whole number in [5, 26]. Defaults to the larger of a and b's configured lg_k (or 12 for compact inputs).

Details

- `theta_union(a, b)` estimates the size of the union `union(A, B)`.
- `theta_intersection(a, b)` estimates the size of the intersection `intersection(A, B)`.
- `theta_difference(a, b)` estimates the size of the set difference `A \\ B` (elements in `A` but not `B`).
- `theta_jaccard(a, b)` estimates the Jaccard similarity index `J(A, B) = |intersection(A, B)| / |union(A, B)|`, returning a named numeric vector `c(lower_bound, estimate, upper_bound)` for a ~95% confidence interval.

Value

A compact `theta_sketch` object (`theta_union()`, `theta_intersection()`, `theta_difference()`), or a named numeric vector `c(lower_bound, estimate, upper_bound)` (`theta_jaccard()`).

Examples

```
a <- theta(1:1000)
b <- theta(501:1500)
theta_union(a, b)$estimate()
theta_intersection(a, b)$estimate()
theta_difference(a, b)$estimate()
theta_jaccard(a, b)
```

---

varopt *VarOpt sketch for variance-optimal sampling and subset-sum estimation*

Description

Creates a VarOpt sketch, which samples up to `k` items from a stream of weighted (item, weight) pairs. It is designed for minimum-variance estimation of subset sums: `$estimate_subset_sum()` estimates the total weight of all stream items matching a predicate, using only the retained sample.

Usage

```
varopt(x = NULL, weight = NULL, k = NULL, type = NULL, bytes = NULL)
```

Arguments

`x`	Optional numeric or character vector of items to update the new sketch with.
`weight`	Optional weight(s) for each element of `x`: a single non-negative, finite number, or a vector of such values matching `length(x)`. Defaults to `1`. Cannot be set without `x`.
`k`	Maximum sample size, a single whole number in `[1, 2^31 - 2]`. Defaults to `256`. Must not be set when `bytes` is supplied.

type	Item type for a fresh sketch, either "double" or "character". Defaults to the type of x (or "double" if x is not supplied). Must not be set when bytes is supplied.
bytes	Optional raw vector holding a native serialized sketch to reconstruct.

Details

Unlike the hash-based cardinality and frequency sketches, VarOpt retains items verbatim rather than hashing them, so the item type (numeric or character) is fixed when the sketch is created and cannot change.

At most one of x or bytes may be supplied:

- Pass x to build a sketch and immediately update it with a numeric or character vector of items (optionally with weight). The item type is inferred from x unless type is supplied.
- Pass bytes to reconstruct a sketch from a native serialized payload (as produced by sketch$serialize()). weight, k, and type must not be supplied alongside bytes; they are restored from the payload.
- Pass neither for an empty (mutable) sketch with the given k and type.

update() silently ignores NA/NaN/NA_character_ in x (and the corresponding weight), matching the missing-value policy used across families; there is no na_rm argument.

Two sketches can only be merged with $merge(), or combined with varopt_union(), if they hold the same item type (a mismatch raises datasketches_incompatible_sketch). Both operations resize the result for the larger of the two inputs' configured k.

Value

A varopt_sketch object. Key methods:

$update(x, weight = NULL) Add weighted items (mutates, returns the sketch).

$merge(other) Absorb another sketch with the same item type (mutates, returns the sketch).

$samples() A data frame of retained items and their estimated weights.

$estimate_subset_sum(predicate) Estimated total weight of stream items matching predicate, with lower_bound and upper_bound.

$k(), $n(), $num_samples(), $is_empty(), $is_character() Metadata accessors.

$summary(), $inspect(), $serialize() Structured metadata, verbose debug output, and the native byte payload.

Examples

```
items <- 1:1000
weights <- runif(1000)
sketch <- varopt(items, weights, k = 50)
sketch$samples()
sketch$estimate_subset_sum(\(x) x <= 500)

# Round-trip through the native byte format.
restored <- varopt(bytes = sketch$serialize())
identical(restored$samples(), sketch$samples())
```

`varopt_union` *Combine two VarOpt sketches*

Description

Combines two `varopt()` sketches into a new `varopt_sketch` result, without mutating either input. `a` and `b` must hold the same item type (a mismatch raises `datasketches_incompatible_sketch`). The result is sized for the larger of `a` and `b`'s configured `k`.

Usage

```
varopt_union(a, b)
```

Arguments

`a`, `b`	`varopt_sketch` objects holding the same item type.

Value

A `varopt_sketch` object.

Examples

```
a <- varopt(1:1000, runif(1000), k = 50)
b <- varopt(501:1500, runif(1000), k = 50)
u <- varopt_union(a, b)
u$k()
u$n()
```

Index