Quantitative Scales

dwtkns edited this page on Oct 30, 2014 · 74 revisions

Scales are functions that map from an input domain to an output range. **Quantitative** scales have a continuous domain, such as the set of real numbers, or dates. There are also ordinal scales, which have a discrete domain, such as a set of names or categories. Scales are an optional feature in D3; you don't have to use them, if you prefer to do the math yourself. However, using scales can greatly simplify the code needed to map a dimension of data to a visual representation.

A scale object, such as that returned by d3.scale.linear, is both an object and a function. That is: you can call the scale like any other function, and the scale has additional methods that change its behavior. Like other classes in D3, scales follow the method chaining pattern where setter methods return the scale itself, allowing multiple setters to be invoked in a concise statement.

Linear Scales

Linear scales are the most common scale, and a good default choice to map a continuous input domain to a continuous output range. The mapping is *linear* in that the output range value *y* can be expressed as a linear function of the input domain value *x*: *y* = *mx* + *b*. The input domain is typically a dimension of the data that you want to visualize, such as the height of students (measured in meters) in a sample population. The output range is typically a dimension of the desired output visualization, such as the height of bars (measured in pixels) in a histogram.

# d3.scale.**linear**()

Constructs a new linear scale with the default domain [0,1] and the default range [0,1]. Thus, the default linear scale is equivalent to the identity function for numbers; for example linear(0.5) returns 0.5.

# **linear**(*x*)

Given a value *x* in the input domain, returns the corresponding value in the output range.

Note: some interpolators **reuse return values**. For example, if the domain values are arbitrary objects, then d3.interpolateObject is automatically applied and the scale reuses the returned object. Often, the return value of a scale is immediately used to set an attribute or style, and you don't have to worry about this; however, if you need to store the scale's return value, use string coercion or create a copy as appropriate.

# linear.**invert**(*y*)

Returns the value in the input domain *x* for the corresponding value in the output range *y*. This represents the inverse mapping from range to domain. For a valid value *y* in the output range, linear(linear.invert(*y*)) equals *y*; similarly, for a valid value *x* in the input domain, linear.invert(linear(*x*)) equals *x*. Equivalently, you can construct the invert operator

Pages 70

Find a Page…

3.0
3.1
Api 参考
API Reference
API Reference (русскоязычная версия)
Arrays
Behaviors
Bundle Layout
Chord Layout
Cluster Layout
CN Home
Colors
Core
CSV
Drag Behavior
Show 55 more pages…

by building a new scale while swapping the domain and range. The invert operator is particularly useful for interaction, say to determine the value in the input domain that corresponds to the pixel location under the mouse.

Note: the invert operator is only supported if the output range is numeric! D3 allows the output range to be any type; under the hood, d3.interpolate or a custom interpolator of your choice is used to map the normalized parameter *t* to a value in the output range. Thus, the output range may be colors, strings, or even arbitrary objects. As there is no facility to "uninterpolate" arbitrary types, the invert operator is currently supported only on numeric ranges.

# linear.**domain**([*numbers*])

If *numbers* is specified, sets the scale's input domain to the specified array of numbers. The array must contain two or more numbers. If the elements in the given array are not numbers, they will be coerced to numbers; this coercion happens similarly when the scale is called. Thus, a linear scale can be used to encode types such as date objects that can be converted to numbers; however, it is often more convenient to use d3.time.scale for dates. (You can implement your own convertible number objects using valueOf.) If *numbers* is not specified, returns the scale's current input domain.

Although linear scales typically have just two numeric values in their domain, you can specify more than two values for a *polylinear* scale. In this case, there must be an equivalent number of values in the output range. A polylinear scale represents multiple piecewise linear scales that divide a continuous domain and range. This is particularly useful for defining diverging quantitative scales. For example, to interpolate between white and red for negative values, and white and green for positive values, say:

```
var color = d3.scale.linear()
    .domain([-1, 0, 1])
    .range(["red", "white", "green"]);
```

The resulting value of color(-.5) is rgb(255, 128, 128), and the value of color(.5) is rgb(128, 192, 128). Internally, polylinear scales perform a binary search for the output interpolator corresponding to the given domain value. By repeating values in both the domain and range, you can also force a chunk of the input domain to map to a constant in the output range.

# linear.**range**([*values*])

If *values* is specified, sets the scale's output range to the specified array of values. The array must contain two or more values, to match the cardinality of the input domain, otherwise the longer of the two is truncated to match the other. The elements in the given array need not be numbers; any value that is supported by the underlying interpolator will work. However, numeric ranges are required for the invert operator. If *values* is not specified, returns the scale's current output range.

# linear.**rangeRound**(*values*)

Sets the scale's output range to the specified array of values, while also setting the scale's interpolator to d3.interpolateRound. This is a convenience routine for when the values output by the scale should be exact integers, such as to avoid antialiasing artifacts. It is also possible to round the output values manually after the scale is applied.

# linear.**interpolate**([*factory*])

If *factory* is specified, sets the scale's output interpolator using the specified *factory*. The interpolator factory defaults to d3.interpolate, and is used to map the normalized domain parameter *t* in [0,1] to the corresponding value in the output range. The interpolator factory will be used to construct interpolators for each adjacent pair of values from the output range. If *factory* is not specified, returns the scale's interpolator factory.

# linear.**clamp**([*boolean*])

If *boolean* is specified, enables or disables clamping accordingly. By default, clamping is disabled, such that if a value outside the input domain is passed to the scale, the scale may return a value outside the output range through linear extrapolation. For example, with the default domain and range of [0,1], an input value of 2 will return an output value of 2. If clamping is enabled, the normalized domain parameter *t* is clamped to the range [0,1], such that the return value of the scale is always within the scale's output range. If *boolean* is not specified, returns whether or not the scale currently clamps values to within the output range.

# linear.**nice**([*count*])

Extends the domain so that it starts and ends on nice round values. This method typically modifies the scale's domain, and may only extend the bounds to the nearest round value. The precision of the round value is dependent on the extent of the domain *dx* according to the following formula: exp(round(log(*dx*)) - 1). Nicing is useful if the domain is computed from data and may be irregular. For example, for a domain of [0.20147987687960267, 0.996679553296417], the nice domain is [0.2, 1]. If the domain has more than two values, nicing the domain only affects the first and last value.

The optional tick *count* argument allows greater control over the step size used to extend the bounds, guaranteeing that the returned ticks will exactly cover the domain.

# linear.**ticks**([*count*])

Returns approximately *count* representative values from the scale's input domain. If *count* is not specified, it defaults to 10. The returned tick values are uniformly spaced, have human-readable values (such as multiples of powers of 10), and are guaranteed to be within the extent of the input domain. Ticks are often used to display reference lines, or tick marks, in conjunction with the visualized data. The specified *count* is only a hint; the scale may return more or fewer values depending on the input domain.

# linear.**tickFormat**(*count*, [*format*])

Returns a number format function suitable for displaying a tick value. The specified *count* should have the same value as the count that is used to generate the tick values. You don't have to use the scale's built-in tick format, but it automatically computes the appropriate precision based on the fixed interval between tick values.

The optional *format* argument allows a format specifier to be specified, where the precision of the format is automatically substituted by the scale to be appropriate for the tick interval. For example, to format percentage change, you might say:

```
var x = d3.scale.linear().domain([-1, 1]);
console.log(x.ticks(5).map(x.tickFormat(5, "+%"))); // ["-100%", "-50%", "+0%", "+5(
```

Likewise, if *format* uses the format type `s`, the scale will compute a SI-prefix based on the largest value in the domain, and use that SI-prefix for all tick values. If the *format* already specifies a precision, this method is equivalent to d3.format.

Note that when using a log scale in conjunction with an axis, you typically want to use axis.ticks rather than axis.tickFormat to take advantage of the log scale's custom tick format, as in bl.ocks.org/5537697.

# linear.**copy**()

Returns an exact copy of this linear scale. Changes to this scale will not affect the returned scale, and vice versa.

Identity Scales

Identity scales are a special case of linear scales where the domain and range are identical; the scale and its invert method are both the identity function. These scales are occasionally useful when working with pixel coordinates, say in conjunction with the axis and brush components.

# d3.scale.**identity**()

Constructs a new identity scale with the default domain [0, 1] and the default range [0, 1]. An identity scale is always equivalent to the identity function.

# **identity**(*x*)
# identity.**invert**(*x*)

Returns the given value *x*.

# identity.**domain**([*numbers*])
# identity.**range**([*numbers*])

If *numbers* is specified, sets the scale's input domain and output range to the specified array of numbers. The array must contain two or more numbers. If the elements in the given array are not numbers, they will be coerced to numbers; this coercion happens similarly when the scale is called. If numbers is not specified, returns the scale's current input domain (or equivalently, output range).

# identity.**ticks**([*count*])

Returns approximately *count* representative values from the scale's input domain (or equivalently, output range). If *count* is not specified, it defaults to 10. The returned tick values are uniformly spaced, have human-readable values (such as multiples of powers of 10), and are guaranteed to be within the extent of the input domain. Ticks are often used to display reference lines, or tick marks, in conjunction with the visualized data. The specified *count* is only a hint; the scale may return more or fewer values depending on the input domain.

# identity.**tickFormat**(*count*, [*format*])

Returns a number format function suitable for displaying a tick value. The specified *count* should have the same value as the count that is used to generate the tick values. You don't have to use the scale's built-in tick format, but it automatically computes the appropriate precision based on the fixed interval between tick values.

The optional *format* argument allows a format specifier to be specified. If the format specifier doesn't have a defined precision, the precision will be set automatically by the scale, returning the appropriate format. This provides a convenient, declarative way of specifying a format whose precision will be automatically set by the scale.

# identity.**copy**()

Returns an exact copy of this scale. Changes to this scale will not affect the returned scale, and vice versa.

Power Scales

Power scales are similar to linear scales, except there's an exponential transform that is applied to the input domain value before the output range value is computed. The mapping to the output range value *y* can be expressed as a function of the input domain value *x*: *y* = *mx^k* + *b*, where *k* is the exponent value. Power scales also support negative values, in which case the input value is multiplied by -1, and the resulting output value is also multiplied by -1.

# d3.scale.**sqrt**()

Constructs a new power scale with the default domain [0,1], the default range [0,1], and the exponent .5. This method is shorthand for:

```
d3.scale.pow().exponent(.5)
```

The returned scale is a function that takes a single argument *x* representing a value in the input domain; the return value is the corresponding value in the output range. Thus, the returned scale is equivalent to the sqrt function for numbers; for example sqrt(0.25) returns 0.5.

# d3.scale.**pow**()

Constructs a new power scale with the default domain [0,1], the default range [0,1], and the default exponent 1. Thus, the default power scale is equivalent to the identity function for numbers; for example pow(0.5) returns 0.5.

# **pow**(*x*)

Given a value *x* in the input domain, returns the corresponding value in the output range.

Note: some interpolators **reuse return values**. For example, if the domain values are arbitrary objects, then d3.interpolateObject is automatically applied and the scale reuses the returned object. Often, the return value of a scale is immediately used to set an attribute or style, and you don't have to worry about this; however, if you need to store the scale's return value, use string coercion or create a copy as appropriate.

# pow.**invert**(*y*)

Returns the value in the input domain *x* for the corresponding value in the output range *y*. This represents the inverse mapping from range to domain. For a valid value *y* in the output range, pow(pow.invert(*y*)) equals *y*; similarly, for a valid value *x* in the input domain, pow.invert(pow(*x*)) equals *x*. Equivalently, you can construct the invert operator by building a new scale while swapping the domain and range. The invert operator is particularly useful for interaction, say to determine the value in the input domain that corresponds to the pixel location under the mouse.

Note: the invert operator is only supported if the output range is numeric! D3 allows the output range to be any type; under the hood, d3.interpolate or a custom interpolator of your choice is used to map the normalized parameter *t* to a value in the output range. Thus, the output range may be colors, strings, or even arbitrary objects. As there is no facility to "uninterpolate" arbitrary types, the invert operator is currently supported only on numeric ranges.

# pow.**domain**([*numbers*])

If *numbers* is specified, sets the scale's input domain to the specified array of numbers. The array must contain two or more numbers. If the elements in the given array are not numbers, they will be coerced to numbers; this coercion happens similarly when the scale is called. Thus, a power scale can be used to encode any type that can be converted to numbers. If *numbers* is not specified, returns the scale's current input domain.

As with linear scales (see linear.domain), power scales can also accept more than two values for the domain and range, thus resulting in polypower scale.

# pow.**range**([*values*])

If *values* is specified, sets the scale's output range to the specified array of values. The array must contain two or more values, to match the cardinality of the input domain, otherwise the longer of the two is truncated to match the other. The elements in the given array need not be numbers; any value that is supported by the underlying interpolator will work. However, numeric ranges are required for the invert operator. If *values* is not specified, returns the scale's current output range.

# pow.**rangeRound**(*values*)

Sets the scale's output range to the specified array of values, while also setting the scale's interpolator to d3.interpolateRound. This is a convenience routine for when the values output by the scale should be exact integers, such as to avoid antialiasing artifacts. It is also possible to round the output values manually after the scale is applied.

# pow.**exponent**([*k*])

If *k* is specified, sets the current exponent to the given numeric value. If *k* is not specified, returns the current exponent. The default value is 1.

# pow.**interpolate**([*factory*])

If *factory* is specified, sets the scale's output interpolator using the specified *factory*. The interpolator factory defaults to d3.interpolate, and is used to map the normalized domain parameter *t* in [0,1] to the corresponding value in the output range. The interpolator factory will be used to construct interpolators for each adjacent pair of values from the output range. If *factory* is not specified, returns the scale's interpolator factory.

# pow.**clamp**([*boolean*])

If *boolean* is specified, enables or disables clamping accordingly. By default, clamping is disabled, such that if a value outside the input domain is passed to the scale, the scale may return a value outside the output range through linear extrapolation. For example, with the default domain and range of [0,1], an input value of 2 will return an output value of 2. If clamping is enabled, the normalized domain parameter *t* is clamped to the range [0,1], such that the return value of the scale is always within the scale's output range. If *boolean* is not specified, returns whether or not the scale currently clamps values to within the output range.

# pow.**nice**([*m*])

Extends the domain so that it starts and ends on nice round values. This method typically modifies the scale's domain, and may only extend the bounds to the nearest round value. The precision of the round value is dependent on the extent of the domain *dx* according to the following formula: exp(round(log(*dx*)) - 1). Nicing is useful if the domain is computed from data and may be irregular. For example, for a domain of [0.20147987687960267, 0.996679553296417], the nice domain is [0.2, 1]. If the domain has more than two values, nicing the domain only affects the first and last value.

The optional *m* argument allows a tick count to be specified to control the step size used prior to extending the bounds.

# pow.**ticks**([*count*])

Returns approximately *count* representative values from the scale's input domain. If *count* is not specified, it defaults to 10. The returned tick values are uniformly spaced, have human-readable values (such as multiples of powers of 10), and are guaranteed to be within the extent of the input domain. Ticks are often used to display reference lines, or tick marks, in conjunction with the visualized data. The specified *count* is only a hint; the scale may return more or fewer values depending on the input domain.

# pow.**tickFormat**([*count*, [*format*]])

Returns a number format function suitable for displaying a tick value. The specified *count* should have the same value as the count that is used to generate the tick values. You don't have to use the scale's built-in tick format, but it automatically computes the appropriate precision based on the fixed interval between tick values.

The optional *format* argument allows a format specifier to be specified. If the format specifier doesn't have a defined precision, the precision will be set automatically by the scale, returning the appropriate format. This provides a convenient, declarative way of specifying a format whose precision will be automatically set by the scale.

# pow.**copy**()

Returns an exact copy of this scale. Changes to this scale will not affect the returned scale, and vice versa.

Log Scales

Log scales are similar to linear scales, except there's a logarithmic transform that is applied to the input domain value before the output range value is computed. The mapping to the output range value *y* can be expressed as a function of the input domain value *x*: $y = m \log(x) + b$.

As log(0) is negative infinity, a log scale must have either an exclusively-positive or exclusively-negative domain; the domain must not include or cross zero. A log scale with a positive domain has a well-defined behavior for positive values, and a log scale with a negative domain has a well-defined behavior for negative values (the input value is multiplied by -1, and the resulting output value is also multiplied by -1). The behavior of the scale is undefined if you pass a negative value to a log scale with a positive domain or vice versa.

# d3.scale.**log**()

Constructs a new log scale with the default domain [1,10], the default range [0,1], and the base 10.

# **log**(*x*)

Given a value *x* in the input domain, returns the corresponding value in the output range.

Note: some interpolators **reuse return values**. For example, if the domain values are arbitrary objects, then d3.interpolateObject is automatically applied and the scale reuses the returned object. Often, the return value of a scale is immediately used to set an attribute or style, and you don’t have to worry about this; however, if you need to store the scale’s return value, use string coercion or create a copy as appropriate.

# log.**invert**(*y*)

Returns the value in the input domain *x* for the corresponding value in the output range *y*. This represents the inverse mapping from range to domain. For a valid value *y* in the output range, log(log.invert(*y*)) equals *y*; similarly, for a valid value *x* in the input domain, log.invert(log(*x*)) equals *x*. Equivalently, you can construct the invert operator by building a new scale while swapping the domain and range. The invert operator is particularly useful for interaction, say to determine the value in the input domain that corresponds to the pixel location under the mouse.

Note: the invert operator is only supported if the output range is numeric! D3 allows the output range to be any type; under the hood, d3.interpolate or a custom interpolator of your choice is used to map the normalized parameter *t* to a value in the output range. Thus, the output range may be colors, strings, or even arbitrary objects. As there is no facility to "uninterpolate" arbitrary types, the invert operator is currently supported only on numeric ranges.

# log.**domain**([*numbers*])

If *numbers* is specified, sets the scale's input domain to the specified array of numbers. The array must contain two or more numbers. If the elements in the given array are not numbers, they will be coerced to numbers; this coercion happens similarly when the scale is called. Thus, a log scale can be used to encode any type that can be converted to numbers. If *numbers* is not specified, returns the scale's current input domain.

As with linear scales (see linear.domain), log scales can also accept more than two values for the domain and range, thus resulting in polylog scale.

# log.**range**([*values*])

If *values* is specified, sets the scale's output range to the specified array of values. The array must contain two or more values, to match the cardinality of the input domain, otherwise the longer of the two is truncated to match the other. The elements in the given array need not be numbers; any value that is supported by the underlying interpolator will work. However, numeric ranges are required for the invert operator. If *values* is not specified, returns the scale's current output range.

# log.**rangeRound**(*values*)

Sets the scale's output range to the specified array of values, while also setting the scale's interpolator to d3.interpolateRound. This is a convenience routine for when the values output by the scale should be exact integers, such as to avoid antialiasing artifacts. It is also possible to round the output values manually after the scale is applied.

# log.**base**([*base*])

If *base* is specified, sets the base for this logarithmic scale. If *base* is not specified, returns the current base, which defaults to 10.

# log.**interpolate**([*factory*])

If *factory* is specified, sets the scale's output interpolator using the specified *factory*. The interpolator factory defaults to d3.interpolate, and is used to map the normalized domain parameter *t* in [0,1] to the corresponding value in the output range. The interpolator factory will be used to construct interpolators for each adjacent pair of values from the output range. If *factory* is not specified, returns the scale's interpolator factory.

# log.**clamp**([*boolean*])

If *boolean* is specified, enables or disables clamping accordingly. By default, clamping is disabled, such that if a value outside the input domain is passed to the scale, the scale may return a value outside the output range through linear extrapolation. For example, with the default domain and range of [0,1], an input value of 2 will return an output value of 2. If clamping is enabled, the normalized domain parameter *t* is clamped to the range [0,1], such that the return value of the scale is always within the scale's output range. If *boolean* is not specified, returns whether or not the scale currently clamps values to within the output range.

# log.**nice**()

Extends the domain so that it starts and ends on nice round values. This method typically modifies the scale's domain, and may only extend the bounds to the nearest round value. The nearest round value is based on integer powers of the scale's base, which defaults to 10. Nicing is useful if the domain is computed from data and may be irregular. For example, for a domain of [0.20147987687960267, 0.996679553296417], the nice domain is [0.1, 1]. If the domain has more than two values, nicing the domain only affects the first and last value.

# log.**ticks**()

Returns representative values from the scale's input domain. The returned tick values are uniformly spaced within each power of ten, and are guaranteed to be within the extent of the input domain. Ticks are often used to display reference lines, or tick marks, in conjunction with the visualized data. Note that the number of ticks cannot be customized (due to the nature of log scales); however, you can filter the returned array of values if you want to reduce the number of ticks.

# log.**tickFormat**([*count*, [*format*]])

Returns a number format function suitable for displaying a tick value. The returned tick format is implemented as `d.toPrecision(1)`. If a *count* is specified, then some of the tick labels may not be displayed; this is useful if there is not enough room to fit all of the tick labels. However, note that the tick marks will still be displayed (so that the log scale distortion remains visible). When specifying a count, you may also override the *format* function; you can also specify a format specifier as a string, and it will automatically be wrapped with d3.format. For example, to get a tick formatter that will display 20 ticks of a currency:

```
scale.tickFormat(20, "$,.2f");
```

If the format specifier doesn't have a defined precision, the precision will be set automatically by the scale, returning the appropriate format. This provides a convenient, declarative way of specifying a format whose precision will be automatically set by the scale.

# log.**copy**()

Returns an exact copy of this scale. Changes to this scale will not affect the returned scale, and vice versa.

Quantize Scales

Quantize scales are a variant of linear scales with a discrete rather than continuous range. The input domain is still continuous, and divided into uniform segments based on the number of values in (the cardinality of) the output range. The mapping is *linear* in that the output range value *y* can be expressed as a linear function of the input domain value *x*: $y = mx + b$. The input domain is typically a dimension of the data that you want to visualize, such as the height of students (measured in meters) in a sample population. The output range is typically a dimension of the desired output visualization, such as the height of bars (measured in pixels) in a histogram.

# d3.scale.**quantize**()

Constructs a new quantize scale with the default domain [0,1] and the default range [0,1]. Thus, the default quantize scale is equivalent to the round function for numbers; for example quantize(0.49) returns 0, and quantize(0.51) returns 1.

```
var q = d3.scale.quantize().domain([0, 1]).range(['a', 'b', 'c']);
//q(0.3) === 'a', q(0.4) === 'b', q(0.6) === 'b', q(0.7) ==='c';
//q.invertExtent('a') returns [0, 0.3333333333333333]
```

# **quantize**(*x*)

Given a value *x* in the input domain, returns the corresponding value in the output range.

# quantize.**invertExtent**(*y*)

Returns the extent of values in the input domain [*x0*, *x1*] for the corresponding value in the output range *y*, representing the inverse mapping from range to domain. This method is useful for interaction, say to determine the value in the input domain that corresponds to the pixel location under the mouse.

# quantize.**domain**([*numbers*])

If *numbers* is specified, sets the scale's input domain to the specified two-element array of numbers. If the array contains more than two numbers, only the first and last number are used. If the elements in the given array are not numbers, they will be coerced to numbers; this coercion happens similarly when the scale is called. Thus, a quantize scale can be used to encode any type that can be converted to numbers. If *numbers* is not specified, returns the scale's current input domain.

# quantize.**range**([*values*])

If *values* is specified, sets the scale's output range to the specified array of values. The array may contain any number of discrete values. The elements in the given array need not be numbers; any value or type will work. If *values* is not specified, returns the scale's current output range.

# quantize.**copy**()

Returns an exact copy of this scale. Changes to this scale will not affect the returned scale, and vice versa.

Quantile Scales

Quantile scales map an input domain to a discrete range. Although the input domain is continuous and the scale will accept any reasonable input value, the input domain is specified as a discrete set of values. The number of values in (the cardinality of) the output range determines the number of quantiles that will be computed from the input domain. To compute the quantiles, the input domain is sorted, and treated as a population of discrete values. The input domain is typically a dimension of the data that you want to visualize, such as the daily change of the stock market. The output range is typically a dimension of the desired output visualization, such as a diverging color scale.

# d3.scale.**quantile**()

Constructs a new quantile scale with an empty domain and an empty range. The quantile scale is invalid until both a domain and range are specified.

# **quantile**(*x*)

Given a value *x* in the input domain, returns the corresponding value in the output range.

# quantile.**invertExtent**(*y*)

Returns the extent of values in the input domain [*x0*, *x1*] for the corresponding value in the output range *y*, representing the inverse mapping from range to domain. This method is useful for interaction, say to determine the value in the input domain that corresponds to the pixel location under the mouse.

# quantile.**domain**([*numbers*])

If *numbers* is specified, sets the input domain of the quantile scale to the specified set of discrete numeric values. The array must not be empty, and must contain at least one numeric value; NaN, null and undefined values are ignored and not considered part of the sample population. If the elements in the given array are not numbers, they will be coerced to numbers; this coercion happens similarly when the scale is called. A copy of the input array is sorted and stored internally. Thus, a quantile scale can be used to encode any type that can be converted to numbers. If *numbers* is not specified, returns the scale's current input domain.

# quantile.**range**([*values*])

If *values* is specified, sets the discrete values in the output range. The array must not be empty, and may contain any type of value. The number of values in (the cardinality, or length, of) the *values* array determines the number of quantiles that are computed. For example, to compute quartiles, *values* must be an array of four elements such as [0, 1, 2, 3]. If *values* is not specified, returns the current output range.

# quantile.**quantiles**()

Returns the quantile thresholds. If the output range contains *n* discrete values, the returned threshold array will contain *n* - 1 values. Values less than the first element in the thresholds array, quantiles()[0], are considered in the first quantile; greater values less than the second threshold are in the second quantile, and so on. Internally, the thresholds array is used with d3.bisect to find the output quantile associated with the given input value.

# quantile.**copy**()

Returns an exact copy of this scale. Changes to this scale will not affect the returned scale, and vice versa.

Threshold Scales

Threshold scales are similar to quantize scales, except they allow you to map arbitrary subsets of the domain to discrete values in the range. The input domain is still continuous, and divided into slices based on a set of threshold values. The input domain is typically a dimension of the data that you want to visualize, such as the height of students (measured in meters) in a sample population. The output range is typically a dimension of the desired output visualization, such as a set of colors (represented as strings).

# d3.scale.**threshold**()

Constructs a new threshold scale with the default domain [.5] and the default range [0,1]. Thus, the default threshold scale is equivalent to the round function for numbers; for example threshold(0.49) returns 0, and threshold(0.51) returns 1.

```
var t = d3.scale.threshold().domain([0, 1]).range(['a', 'b', 'c']);
t(-1) === 'a';
t(0) === 'b';
t(0.5) === 'b';
t(1) === 'c';
t(1000) === 'c';
t.invertExtent('a'); //returns [undefined, 0]
t.invertExtent('b'); //returns [0, 1]
t.invertExtent('c'); //returns [1, undefined]
```

# **threshold**(*x*)

Given a value *x* in the input domain, returns the corresponding value in the output range.

# threshold.**invertExtent**(*y*)

Returns the extent of values in the input domain [*x0*, *x1*] for the corresponding value in the output range *y*, representing the inverse mapping from range to domain. This method is useful for interaction, say to determine the value in the input domain that corresponds to the pixel location under the mouse.

# threshold.**domain**([*domain*])

If *domain* is specified, sets the scale's input domain to the specified array of values. The values must be in sorted ascending order, or the behavior of the scale is undefined. The values are typically numbers, but any naturally ordered values (such as strings) will work. Thus, a threshold scale can be used to encode any type that is ordered. If the number of values in the scale's range is N + 1, the number of values in the scale's domain must be N. If there are fewer than N elements in the domain, the additional values in the range are ignored. If there are more than N elements in the domain, the scale may return undefined for some inputs. If *domain* is not specified, returns the scale's current input domain.

# threshold.**range**([*values*])

If *values* is specified, sets the scale's output range to the specified array of values. If the number of values in the scale's domain is N, the number of values in the scale's range must be N + 1. If there are fewer than N+1 elements in the range, the scale may return undefined for some inputs. If there are more than N + 1 elements in the range, the additional values are ignored. The elements in the given array need not be numbers; any value or type will work. If *values* is not specified, returns the scale's current output range.

# threshold.**copy**()

Returns an exact copy of this scale. Changes to this scale will not affect the returned scale, and vice versa.