Python Figure Reference: layout.xaxis

xaxis

Code: `fig.update_xaxes(...)`

Type: dict containing one or more of the keys listed below.

anchor

Code: `fig.update_xaxes(anchor=<VALUE>)`

Type: enumerated , one of (`"free"` | `"/^x([2-9]|[1-9][0-9]+)?( domain)?$/"` | `"/^y([2-9]|[1-9][0-9]+)?( domain)?$/"`)

If set to an opposite-letter axis id (e.g. `x2`, `y`), this axis is bound to the corresponding opposite-letter axis. If set to "free", this axis' position is determined by `position`.

automargin

Code: `fig.update_xaxes(automargin=<VALUE>)`

Type: flaglist string. Any combination of `"height"` , `"width"` , `"left"` , `"right"` , `"top"` , `"bottom"` joined with a `"+"` OR `True` or `False` .

Examples: `"height"` , `"width"` , `"height+width"` , `"height+width+left"` , `"True"`

Determines whether long tick labels automatically grow the figure margins.

autorange

Code: `fig.update_xaxes(autorange=<VALUE>)`

Type: enumerated , one of (`True` | `False` | `"reversed"` | `"min reversed"` | `"max reversed"` | `"min"` | `"max"`)

Default: `True`

Determines whether or not the range of this axis is computed in relation to the input data. See `rangemode` for more info. If `range` is provided and it has a value for both the lower and upper bound, `autorange` is set to "False". Using "min" applies autorange only to set the minimum. Using "max" applies autorange only to set the maximum. Using "min reversed" applies autorange only to set the minimum on a reversed axis. Using "max reversed" applies autorange only to set the maximum on a reversed axis. Using "reversed" applies autorange on both ends and reverses the axis direction.

autorangeoptions

Code: `fig.update_xaxes(autorangeoptions=dict(...))`

Type: dict containing one or more of the keys listed below.

clipmax

Code: `fig.update_xaxes(autorangeoptions_clipmax=<VALUE>)`

Type: number or categorical coordinate string

Clip autorange maximum if it goes beyond this value. Has no effect when `autorangeoptions.maxallowed` is provided.

clipmin

Code: `fig.update_xaxes(autorangeoptions_clipmin=<VALUE>)`

Type: number or categorical coordinate string

Clip autorange minimum if it goes beyond this value. Has no effect when `autorangeoptions.minallowed` is provided.

include

Code: `fig.update_xaxes(autorangeoptions_include=<VALUE>)`

Type: number or categorical coordinate string

Ensure this value is included in autorange.

maxallowed

Code: `fig.update_xaxes(autorangeoptions_maxallowed=<VALUE>)`

Type: number or categorical coordinate string

Use this value exactly as autorange maximum.

minallowed

Code: `fig.update_xaxes(autorangeoptions_minallowed=<VALUE>)`

Type: number or categorical coordinate string

Use this value exactly as autorange minimum.

autotickangles

Code: `fig.update_xaxes(autotickangles=list(...))`

Type: list

Default: `[0, 30, 90]`

When `tickangle` is set to "auto", it will be set to the first angle in this array that is large enough to prevent label overlap.

autotypenumbers

Code: `fig.update_xaxes(autotypenumbers=<VALUE>)`

Type: enumerated , one of (`"convert types"` | `"strict"`)

Default: `"convert types"`

Using "strict" a numeric string in trace data is not converted to a number. Using "convert types" a numeric string in trace data may be treated as a number during automatic axis `type` detection. Defaults to layout.autotypenumbers.

calendar

Code: `fig.update_xaxes(calendar=<VALUE>)`

Type: enumerated , one of (`"chinese"` | `"coptic"` | `"discworld"` | `"ethiopian"` | `"gregorian"` | `"hebrew"` | `"islamic"` | `"jalali"` | `"julian"` | `"mayan"` | `"nanakshahi"` | `"nepali"` | `"persian"` | `"taiwan"` | `"thai"` | `"ummalqura"`)

Default: `"gregorian"`

Sets the calendar system to use for `range` and `tick0` if this is a date axis. This does not set the calendar for interpreting data on this axis, that's specified in the trace or via the global `layout.calendar`

categoryarray

Code: `fig.update_xaxes(categoryarray=<VALUE>)`

Type: list, numpy array, or Pandas series of numbers, strings, or datetimes.

Sets the order in which categories on this axis appear. Only has an effect if `categoryorder` is set to "array". Used with `categoryorder`.

categoryorder

Code: `fig.update_xaxes(categoryorder=<VALUE>)`

Type: enumerated , one of (`"trace"` | `"category ascending"` | `"category descending"` | `"array"` | `"total ascending"` | `"total descending"` | `"min ascending"` | `"min descending"` | `"max ascending"` | `"max descending"` | `"sum ascending"` | `"sum descending"` | `"mean ascending"` | `"mean descending"` | `"geometric mean ascending"` | `"geometric mean descending"` | `"median ascending"` | `"median descending"`)

Default: `"trace"`

Specifies the ordering logic for the case of categorical variables. By default, plotly uses "trace", which specifies the order that is present in the data supplied. Set `categoryorder` to "category ascending" or "category descending" if order should be determined by the alphanumerical order of the category names. Set `categoryorder` to "array" to derive the ordering from the attribute `categoryarray`. If a category is not found in the `categoryarray` array, the sorting behavior for that attribute will be identical to the "trace" mode. The unspecified categories will follow the categories in `categoryarray`. Set `categoryorder` to "total ascending" or "total descending" if order should be determined by the numerical order of the values. Similarly, the order can be determined by the min, max, sum, mean, geometric mean or median of all the values.

`axes(color=<VALUE>)`

Sets default for all colors associated with this axis all at once: line, font, tick, and grid colors. Grid color is lightened by blending this with the plot background Individual pieces can override this.

constrain

Code: `fig.update_xaxes(constrain=<VALUE>)`

Type: enumerated , one of (`"range"` | `"domain"`)

If this axis needs to be compressed (either due to its own `scaleanchor` and `scaleratio` or those of the other axis), determines how that happens: by increasing the "range", or by decreasing the "domain". Default is "domain" for axes containing image traces, "range" otherwise.

constraintoward

Code: `fig.update_xaxes(constraintoward=<VALUE>)`

Type: enumerated , one of (`"left"` | `"center"` | `"right"` | `"top"` | `"middle"` | `"bottom"`)

If this axis needs to be compressed (either due to its own `scaleanchor` and `scaleratio` or those of the other axis), determines which direction we push the originally specified plot area. Options are "left", "center" (default), and "right" for x axes, and "top", "middle" (default), and "bottom" for y axes.

dividercolor

Code: `fig.update_xaxes(dividercolor=<VALUE>)`

Type: color

Default: `"#444"`

Sets the color of the dividers Only has an effect on "multicategory" axes.

dividerwidth

Code: `fig.update_xaxes(dividerwidth=<VALUE>)`

Type: number

Default: `1`

Sets the width (in px) of the dividers Only has an effect on "multicategory" axes.

domain

Code: `fig.update_xaxes(domain=list(...))`

Type: list

Default: `[0, 1]`

Sets the domain of this axis (in plot fraction).

dtick

Code: `fig.update_xaxes(dtick=<VALUE>)`

Type: number or categorical coordinate string

Sets the step in-between ticks on this axis. Use with `tick0`. Must be a positive number, or special strings available to "log" and "date" axes. If the axis `type` is "log", then ticks are set every 10^(n"dtick) where n is the tick number. For example, to set a tick mark at 1, 10, 100, 1000, ... set dtick to 1. To set tick marks at 1, 100, 10000, ... set dtick to 2. To set tick marks at 1, 5, 25, 125, 625, 3125, ... set dtick to log_10(5), or 0.69897000433. "log" has several special values; "L<f>", where `f` is a positive number, gives ticks linearly spaced in value (but not position). For example `tick0` = 0.1, `dtick` = "L0.5" will put ticks at 0.1, 0.6, 1.1, 1.6 etc. To show powers of 10 plus small digits between, use "D1" (all digits) or "D2" (only 2 and 5). `tick0` is ignored for "D1" and "D2". If the axis `type` is "date", then you must convert the time to milliseconds. For example, to set the interval between ticks to one day, set `dtick` to 86400000.0. "date" also has special values "M<n>" gives ticks spaced by a number of months. `n` must be a positive integer. To set ticks on the 15th of every third month, set `tick0` to "2000-01-15" and `dtick` to "M3". To set ticks every 4 years, set `dtick` to "M48"

exponentformat

Code: `fig.update_xaxes(exponentformat=<VALUE>)`

, one of (`"none"` | `"e"` | `"E"` | `"power"` | `"SI"` | `"B"`)

matting rule for the tick exponents. For example, consider the number 1,000,000,000. If "none", it appears as
e", 1e+9. If "E", 1E+9. If "power", 1x10^9 (with 9 in a super script). If "SI", 1G. If "B", 1B.

fixedrange

Code: fig.update_xaxes(fixedrange=<VALUE>)

Type: boolean

Determines whether or not this axis is zoom-able. If True, then zoom is disabled.

gridcolor

Code: fig.update_xaxes(gridcolor=<VALUE>)

Type: color

Default: "#eee"

Sets the color of the grid lines.

griddash

Code: fig.update_xaxes(griddash=<VALUE>)

Type: string

Default: "solid"

Sets the dash style of lines. Set to a dash type string ("solid", "dot", "dash", "longdash", "dashdot", or "longdashdot") or a dash length list in px (eg "5px,10px,2px,2px").

gridwidth

Code: fig.update_xaxes(gridwidth=<VALUE>)

Type: number greater than or equal to 0

Default: 1

Sets the width (in px) of the grid lines.

hoverformat

Code: fig.update_xaxes(hoverformat=<VALUE>)

Type: string

Default: ""

Sets the hover text formatting rule using d3 formatting mini-languages which are very similar to those in Python. For numbers, see: https://github.com/d3/d3-format/tree/v1.4.5#d3-format. And for dates see: https://github.com/d3/d3-time-format/tree/v2.2.3#locale_format. We add two items to d3's date formatter: "%h" for half of the year as a decimal number as well as "%{n}f" for fractional seconds with n digits. For example, "2016-10-13 09:15:23.456" with tickformat "%H~%M~%S.%2f" would display "09~15~23.46"

insiderange

Code: fig.update_xaxes(insiderange=list(...))

Type: list

Could be used to set the desired inside range of this axis (excluding the labels) when `ticklabelposition` of the anchored axis has "inside". Not implemented for axes with `type` "log". This would be ignored when `range` is provided.

labelalias

Code: fig.update_xaxes(labelalias=<VALUE>)

Type: number or categorical coordinate string

Replacement text for specific tick or hover labels. For example using {US: 'USA', CA: 'Canada'} changes US to USA and CA to Canada. The labels we would have shown must match the keys exactly, after adding any tickprefix or ticksuffix. For negative numbers the minus sign symbol used (U+2212) is wider than the regular ascii dash. That means you need to use −1 instead of -1. labelalias can be used with any axis type, and both keys (if needed) and values (if desired) can include html-like tags or MathJax.

axes(layer=<VALUE>)

, one of ("above traces" | "below traces")

aces"

Sets the layer on which this axis is displayed. If "above traces", this axis is displayed above all the subplot's traces If "below traces", this axis is displayed below all the subplot's traces, but above the grid lines. Useful when used together with scatter-like traces with `cliponaxis` set to "False" to show markers and/or text nodes above this axis.

linecolor

Code: fig.update_xaxes(linecolor=<VALUE>)

Type: color

Default: "#444"

Sets the axis line color.

linewidth

Code: fig.update_xaxes(linewidth=<VALUE>)

Type: number greater than or equal to 0

Default: 1

Sets the width (in px) of the axis line.

matches

Code: fig.update_xaxes(matches=<VALUE>)

Type: enumerated , one of ("/^x([2-9]|[1-9][0-9]+)?(domain)?$/" | "/^y([2-9]|[1-9][0-9]+)?(domain)?$/")

If set to another axis id (e.g. `x2`, `y`), the range of this axis will match the range of the corresponding axis in data-coordinates space. Moreover, matching axes share auto-range values, category lists and histogram auto-bins. Note that setting axes simultaneously in both a `scaleanchor` and a `matches` constraint is currently forbidden. Moreover, note that matching axes must have the same `type`.

maxallowed

Code: fig.update_xaxes(maxallowed=<VALUE>)

Type: number or categorical coordinate string

Determines the maximum range of this axis.

minallowed

Code: fig.update_xaxes(minallowed=<VALUE>)

Type: number or categorical coordinate string

Determines the minimum range of this axis.

minexponent

Code: fig.update_xaxes(minexponent=<VALUE>)

Type: number greater than or equal to 0

Default: 3

Hide SI prefix for 10^n if |n| is below this number. This only has an effect when `tickformat` is "SI" or "B".

minor

Code: fig.update_xaxes(minor=dict(...))

Type: dict containing one or more of the keys listed below.

dtick

Code: fig.update_xaxes(minor_dtick=<VALUE>)

Type: number or categorical coordinate string

Sets the step in-between ticks on this axis. Use with `tick0`. Must be a positive number, or special strings available to "log" and "date" axes. If the axis `type` is "log", then ticks are set every 10^(n"dtick) where n is the tick number. For example, to set a tick mark at 1, 10, 100, 1000, ... set dtick to 1. To set tick marks at 1, 100, 10000, ... set dtick to 2. To set tick marks at 1, 5, 25, 125, 625, 3125, ... set dtick to log_10(5), or 0.69897000433. "log" has several special values; "L<f>", where `f` is a positive number, gives ticks linearly spaced in value (but not position). For example `tick0` = 0.1, `dtick` = "L0.5" will put ticks at 0.1, 0.6, 1.1, 1.6 etc. To show powers of 10 plus small digits between, use "D1" (all digits) or "D2" (only 2 and 5). `tick0` is ignored for "D1" and "D2". If the axis `type` is "date", then you must convert the time to milliseconds. For

example, to set the interval between ticks to one day, set `dtick` to 86400000.0. "date" also has special values "M<n>" gives ticks spaced by a number of months. `n` must be a positive integer. To set ticks on the 15th of every third month, set `tick0` to "2000-01-15" and `dtick` to "M3". To set ticks every 4 years, set `dtick` to "M48"

gridcolor

Code: fig.update_xaxes(minor_gridcolor=<VALUE>)

Type: color

Default: "#eee"

Sets the color of the grid lines.

griddash

Code: fig.update_xaxes(minor_griddash=<VALUE>)

Type: string

Default: "solid"

Sets the dash style of lines. Set to a dash type string ("solid", "dot", "dash", "longdash", "dashdot", or "longdashdot") or a dash length list in px (eg "5px,10px,2px,2px").

gridwidth

Code: fig.update_xaxes(minor_gridwidth=<VALUE>)

Type: number greater than or equal to 0

Sets the width (in px) of the grid lines.

nticks

Code: fig.update_xaxes(minor_nticks=<VALUE>)

Type: integer greater than or equal to 0

Default: 5

Specifies the maximum number of ticks for the particular axis. The actual number of ticks will be chosen automatically to be less than or equal to `nticks`. Has an effect only if `tickmode` is set to "auto".

showgrid

Code: fig.update_xaxes(minor_showgrid=<VALUE>)

Type: boolean

Determines whether or not grid lines are drawn. If "True", the grid lines are drawn at every tick mark.

tick0

Code: fig.update_xaxes(minor_tick0=<VALUE>)

Type: number or categorical coordinate string

Sets the placement of the first tick on this axis. Use with `dtick`. If the axis `type` is "log", then you must take the log of your starting tick (e.g. to set the starting tick to 100, set the `tick0` to 2) except when `dtick`="L<f>" (see `dtick` for more info). If the axis `type` is "date", it should be a date string, like date data. If the axis `type` is "category", it should be a number, using the scale where each category is assigned a serial number from zero in the order it appears.

tickcolor

Code: fig.update_xaxes(minor_tickcolor=<VALUE>)

Type: color

Default: "#444"

Sets the tick color.

ticklen

Code: fig.update_xaxes(minor_ticklen=<VALUE>)

Type: number greater than or equal to 0

Sets the tick length (in px).

tickmode

Code: fig.update_xaxes(minor_tickmode=<VALUE>)

Type: enumerated , one of ("auto" | "linear" | "array")

Sets the tick mode for this axis. If "auto", the number of ticks is set via `nticks`. If "linear", the placement of the ticks is determined by a starting position `tick0` and a tick step `dtick` ("linear" is the default value if `tick0` and `dtick` are provided). If "array", the placement of the ticks is set via `tickvals` and the tick text is `ticktext`. ("array" is the default value if `tickvals` is provided).

ticks

Code: fig.update_xaxes(minor_ticks=<VALUE>)

Type: enumerated , one of ("outside" | "inside" | "")

Determines whether ticks are drawn or not. If "", this axis' ticks are not drawn. If "outside" ("inside"), this axis' are drawn outside (inside) the axis lines.

tickvals

Code: fig.update_xaxes(minor_tickvals=<VALUE>)

Type: list, numpy array, or Pandas series of numbers, strings, or datetimes.

Sets the values at which ticks on this axis appear. Only has an effect if `tickmode` is set to "array". Used with `ticktext`.

tickwidth

Code: fig.update_xaxes(minor_tickwidth=<VALUE>)

Type: number greater than or equal to 0

Sets the tick width (in px).

mirror

Code: fig.update_xaxes(mirror=<VALUE>)

Type: enumerated , one of (True | "ticks" | False | "all" | "allticks")

Determines if the axis lines or/and ticks are mirrored to the opposite side of the plotting area. If "True", the axis lines are mirrored. If "ticks", the axis lines and ticks are mirrored. If "False", mirroring is disable. If "all", axis lines are mirrored on all shared-axes subplots. If "allticks", axis lines and ticks are mirrored on all shared-axes subplots.

nticks

Code: fig.update_xaxes(nticks=<VALUE>)

Type: integer greater than or equal to 0

Default: 0

Specifies the maximum number of ticks for the particular axis. The actual number of ticks will be chosen automatically to be less than or equal to `nticks`. Has an effect only if `tickmode` is set to "auto".

overlaying

Code: fig.update_xaxes(overlaying=<VALUE>)

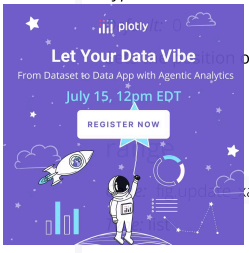
Type: enumerated , one of ("free" | "/^x([2-9]|[1-9][0-9]+)?(domain)?$/" | "/^y([2-9]|[1-9][0-9]+)?(domain)?$/")

If set a same-letter axis id, this axis is overlaid on top of the corresponding same-letter axis, with traces and axes visible for both axes. If "False", this axis does not overlay any same-letter axes. In this case, for axes with overlapping domains only the highest-numbered axis will be visible.

position

Code: fig.update_xaxes(position=<VALUE>)

Type: number between or equal to 0 and 1



of this axis in the plotting space (in normalized coordinates). Only has an effect if `anchor` is set to "free".

axes(range=list(...))

Sets the range of this axis. If the axis `type` is "log", then you must take the log of your desired range (e.g. to set the range from 1 to 100, set the range from 0 to 2). If the axis `type` is "date", it should be date strings, like date data, though Date objects and unix milliseconds will be accepted and converted to strings. If the axis `type` is "category", it should be numbers, using the scale where each category is assigned a serial number from zero in the order it appears. Leaving either or both elements `null` impacts the default `autorange`.

rangebreaks

Code: fig.update_xaxes(rangebreaks=list(...))

Type: list of dict where each dict has one or more of the keys listed below.

bounds

Parent: layout.xaxis.rangebreaks[]

Type: list

Sets the lower and upper bounds of this axis rangebreak. Can be used with `pattern`.

dvalue

Parent: layout.xaxis.rangebreaks[]

Type: number greater than or equal to 0

Default: 86400000

Sets the size of each `values` item. The default is one day in milliseconds.

enabled

Parent: layout.xaxis.rangebreaks[]

Type: boolean

Default: True

Determines whether this axis rangebreak is enabled or disabled. Please note that `rangebreaks` only work for "date" axis type.

name

Parent: layout.xaxis.rangebreaks[]

Type: string

When used in a template, named items are created in the output figure in addition to any items the figure already has in this array. You can modify these items in the output figure by making your own item with `templateitemname` matching this `name` alongside your modifications (including `visible: False` or `enabled: False` to hide it). Has no effect outside of a template.

pattern

Parent: layout.xaxis.rangebreaks[]

Type: enumerated , one of ("day of week" | "hour" | "")

Determines a pattern on the time line that generates breaks. If "day of week" - days of the week in English e.g. 'Sunday' or `sun` (matching is case-insensitive and considers only the first three characters), as well as Sunday-based integers between 0 and 6. If "hour" - hour (24-hour clock) as decimal numbers between 0 and 24. for more info. Examples: - { pattern: 'day of week', bounds: [6, 1] } or simply { bounds: ['sat', 'mon'] } breaks from Saturday to Monday (i.e. skips the weekends). - { pattern: 'hour', bounds: [17, 8] } breaks from 5pm to 8am (i.e. skips non-work hours).

templateitemname

Parent: layout.xaxis.rangebreaks[]

Type: string

Used to refer to a named item in this array in the template. Named items from the template will be created even without a matching item in the input figure, but you can modify one by making an item with `templateitemname` matching its
ngside your modifications (including `visible: False` or `enabled: False` to hide it). If there is no template or no
tem, this item will be hidden unless you explicitly show it with `visible: True`.

values

out.xaxis.rangebreaks[]

Sets the coordinate values corresponding to the rangebreaks. An alternative to `bounds`. Use `dvalue` to set the size of the values along the axis.

rangemode

Code: fig.update_xaxes(rangemode=<VALUE>)

Type: enumerated , one of ("normal" | "tozero" | "nonnegative")

Default: "normal"

If "normal", the range is computed in relation to the extrema of the input data. If "tozero"`, the range extends to 0, regardless of the input data If "nonnegative", the range is non-negative, regardless of the input data. Applies only to linear axes.

rangeselector

Code: fig.update_xaxes(rangeselector=dict(...))

Type: dict containing one or more of the keys listed below.

activecolor

Code: fig.update_xaxes(rangeselector_activecolor=<VALUE>)

Type: color

Sets the background color of the active range selector button.

bgcolor

Code: fig.update_xaxes(rangeselector_bgcolor=<VALUE>)

Type: color

Default: "#eee"

Sets the background color of the range selector buttons.

bordercolor

Code: fig.update_xaxes(rangeselector_bordercolor=<VALUE>)

Type: color

Default: "#444"

Sets the color of the border enclosing the range selector.

borderwidth

Code: fig.update_xaxes(rangeselector_borderwidth=<VALUE>)

Type: number greater than or equal to 0

Default: 0

Sets the width (in px) of the border enclosing the range selector.

buttons

Code: fig.update_xaxes(rangeselector_buttons=list(...))

Type: list of dict where each dict has one or more of the keys listed below.

count

Parent: layout.xaxis.rangeselector.buttons[]

Type: number greater than or equal to 0

Default: 1

Sets the number of steps to take to update the range. Use with `step` to specify the update interval.

label

Parent: layout.xaxis.rangeselector.buttons[]

Type: string

Sets the text label to appear on the button.

name

Parent: layout.xaxis.rangeselector.buttons[]

Type: string

When used in a template, named items are created in the output figure in addition to any items the figure already has in this array. You can modify these items in the output figure by making your own item with `templateitemname` matching this `name` alongside your modifications (including `visible: False` or `enabled: False` to hide it). Has no effect outside of a template.

step

Parent: layout.xaxis.rangeselector.buttons[]

Type: enumerated , one of ("month" | "year" | "day" | "hour" | "minute" | "second" | "all")

Default: "month"

The unit of measurement that the `count` value will set the range by.

stepmode

Parent: layout.xaxis.rangeselector.buttons[]

Type: enumerated , one of ("backward" | "todate")

Default: "backward"

Sets the range update mode. If "backward", the range update shifts the start of range back "count" times "step" milliseconds. If "todate", the range update shifts the start of range back to the first timestamp from "count" times "step" milliseconds back. For example, with `step` set to "year" and `count` set to "1" the range update shifts the start of the range back to January 01 of the current year. Month and year "todate" are currently available only for the built-in (Gregorian) calendar.

templateitemname

Parent: layout.xaxis.rangeselector.buttons[]

Type: string

Used to refer to a named item in this array in the template. Named items from the template will be created even without a matching item in the input figure, but you can modify one by making an item with `templateitemname` matching its `name`, alongside your modifications (including `visible: False` or `enabled: False` to hide it). If there is no template or no matching item, this item will be hidden unless you explicitly show it with `visible: True`.

visible

Parent: layout.xaxis.rangeselector.buttons[]

Type: boolean

Default: True

Determines whether or not this button is visible.

font

Code: fig.update_xaxes(rangeselector_font=dict(...))

Type: dict containing one or more of the keys listed below.

Sets the font of the range selector button text.

color

Code: fig.update_xaxes(rangeselector_font_color=<VALUE>)

Type: color

family

Code: fig.update_xaxes(rangeselector_font_family=<VALUE>)

Type: string

HTML font family - the typeface that will be applied by the web browser. The web browser will only be able to apply a font if it is available on the system which it operates. Provide multiple font families, separated by commas, to indicate the preference in which to apply fonts if they aren't available on the system. The Chart Studio Cloud (at https://chart-studio.plotly.com or on-premise) generates images on a server, where only a select number of fonts are installed and supported. These include "Arial", "Balto", "Courier New", "Droid Sans", "Droid Serif", "Droid Sans Mono", "Gravitas One", "Old Standard TT", "Open Sans", "Overpass", "PT Sans Narrow", "Raleway", "Times New Roman".

lineposition

Code: fig.update_xaxes(rangeselector_font_lineposition=<VALUE>)

Type: flaglist string. Any combination of "under", "over", "through" joined with a "+" OR "none".

Examples: "under", "over", "under+over", "under+over+through", "none"

Default: "none"

Sets the kind of decoration line(s) with text, such as an "under", "over" or "through" as well as combinations e.g. "under+over", etc.

shadow

Code: fig.update_xaxes(rangeselector_font_shadow=<VALUE>)

Type: string

Default: "none"

Sets the shape and color of the shadow behind text. "auto" places minimal shadow and applies contrast text font color. See https://developer.mozilla.org/en-US/docs/Web/CSS/text-shadow for additional options.

size

Code: fig.update_xaxes(rangeselector_font_size=<VALUE>)

Type: number greater than or equal to 1

style

Code: fig.update_xaxes(rangeselector_font_style=<VALUE>)

Type: enumerated , one of ("normal" | "italic")

Default: "normal"

Sets whether a font should be styled with a normal or italic face from its family.

textcase

Code: fig.update_xaxes(rangeselector_font_textcase=<VALUE>)

Type: enumerated , one of ("normal" | "word caps" | "upper" | "lower")

Default: "normal"

Sets capitalization of text. It can be used to make text appear in all-uppercase or all-lowercase, or with each word capitalized.

variant

Code: fig.update_xaxes(rangeselector_font_variant=<VALUE>)

Type: enumerated , one of ("normal" | "small-caps" | "all-small-caps" | "all-petite-caps" | "petite-caps" | "unicase")

Default: "normal"

Sets the variant of the font.

weight

Code: fig.update_xaxes(rangeselector_font_weight=<VALUE>)

Type: integer between or equal to 1 and 1000

Default: normal

Sets the weight (or boldness) of the font.

visible

Code: fig.update_xaxes(rangeselector_visible=<VALUE>)

Type: boolean

Determines whether or not this range selector is visible. Note that range selectors are only available for x axes of `type` set to or auto-typed to "date".

Code: fig.update_xaxes(rangeselector_x=<VALUE>)

Type: number between or equal to -2 and 3

Sets the x position (in normalized coordinates) of the range selector.

xanchor

Code: `fig.update_xaxes(rangeselector_xanchor=<VALUE>)`

Type: enumerated , one of (`"auto"` | `"left"` | `"center"` | `"right"`)

Default: `"left"`

Sets the range selector's horizontal position anchor. This anchor binds the `x` position to the "left", "center" or "right" of the range selector.

y

Code: `fig.update_xaxes(rangeselector_y=<VALUE>)`

Type: number between or equal to -2 and 3

Sets the y position (in normalized coordinates) of the range selector.

yanchor

Code: `fig.update_xaxes(rangeselector_yanchor=<VALUE>)`

Type: enumerated , one of (`"auto"` | `"top"` | `"middle"` | `"bottom"`)

Default: `"bottom"`

Sets the range selector's vertical position anchor This anchor binds the `y` position to the "top", "middle" or "bottom" of the range selector.

rangeslider

Code: `fig.update_xaxes(rangeslider=dict(...))`

Type: dict containing one or more of the keys listed below.

autorange

Code: `fig.update_xaxes(rangeslider_autorange=<VALUE>)`

Type: boolean

Default: `True`

Determines whether or not the range slider range is computed in relation to the input data. If `range` is provided, then `autorange` is set to "False".

bgcolor

Code: `fig.update_xaxes(rangeslider_bgcolor=<VALUE>)`

Type: color

Default: `"#fff"`

Sets the background color of the range slider.

bordercolor

Code: `fig.update_xaxes(rangeslider_bordercolor=<VALUE>)`

Type: color

Default: `"#444"`

Sets the border color of the range slider.

borderwidth

Code: `fig.update_xaxes(rangeslider_borderwidth=<VALUE>)`

Type: integer greater than or equal to 0

Default: `0`

Sets the border width of the range slider.

range

Code: `fig.update_xaxes(rangeslider_range=list(...))`

Sets the range of the range slider. If not set, defaults to the full xaxis range. If the axis `type` is "log", then you must take the log of your desired range. If the axis `type` is "date", it should be date strings, like date data, though Date objects and unix milliseconds will be accepted and converted to strings. If the axis `type` is "category", it should be numbers, using the scale where each category is assigned a serial number from zero in the order it appears.

thickness

Code: fig.update_xaxes(rangeslider_thickness=<VALUE>)

Type: number between or equal to 0 and 1

Default: 0.15

The height of the range slider as a fraction of the total plot area height.

visible

Code: fig.update_xaxes(rangeslider_visible=<VALUE>)

Type: boolean

Default: True

Determines whether or not the range slider will be visible. If visible, perpendicular axes will be set to `fixedrange`

yaxis

Code: fig.update_xaxes(rangeslider_yaxis=dict(...))

Type: dict containing one or more of the keys listed below.

range

Code: fig.update_xaxes(rangeslider_yaxis_range=list(...))

Type: list

Sets the range of this axis for the rangeslider.

rangemode

Code: fig.update_xaxes(rangeslider_yaxis_rangemode=<VALUE>)

Type: enumerated , one of ("auto" | "fixed" | "match")

Default: "match"

Determines whether or not the range of this axis in the rangeslider use the same value than in the main plot when zooming in/out. If "auto", the autorange will be used. If "fixed", the `range` is used. If "match", the current range of the corresponding y-axis on the main subplot is used.

scaleanchor

Code: fig.update_xaxes(scaleanchor=<VALUE>)

Type: enumerated , one of ("/^x([2-9]|[1-9][0-9]+)?(domain)?$/" | "/^y([2-9]|[1-9][0-9]+)?(domain)?$/" | False)

If set to another axis id (e.g. `x2`, `y`), the range of this axis changes together with the range of the corresponding axis such that the scale of pixels per unit is in a constant ratio. Both axes are still zoomable, but when you zoom one, the other will zoom the same amount, keeping a fixed midpoint. `constrain` and `constraintoward` determine how we enforce the constraint. You can chain these, ie `yaxis: {scaleanchor: "x"}, xaxis2: {scaleanchor: "y"}` but you can only link axes of the same `type`. The linked axis can have the opposite letter (to constrain the aspect ratio) or the same letter (to match scales across subplots). Loops (`yaxis: {scaleanchor: "x"}, xaxis: {scaleanchor: "y"}` or longer) are redundant and the last constraint encountered will be ignored to avoid possible inconsistent constraints via `scaleratio`. Note that setting axes simultaneously in both a `scaleanchor` and a `matches` constraint is currently forbidden. Setting `False` allows to remove a default constraint (occasionally, you may need to prevent a default `scaleanchor` constraint from being applied, eg. when having an image trace `yaxis: {scaleanchor: "x"}` is set automatically in order for pixels to be rendered as squares, setting `yaxis: {scaleanchor: False}` allows to remove the constraint).

scaleratio

Code: fig.update_xaxes(scaleratio=<VALUE>)

Type: number greater than or equal to 0

If there are multiple YAxes linked to another by `scaleanchor`, this determines the pixel to unit scale ratio. For example, if this value is 10, then every unit on this axis spans 10 times the number of pixels as a unit on the linked axis. Use this for example to create an elevation profile where the vertical scale is exaggerated a fixed amount with respect to the horizontal.

separatethousands

Code: fig.update_xaxes(separatethousands=<VALUE>)

Type: boolean

If "True", even 4-digit integers are separated

showdividers

Code: fig.update_xaxes(showdividers=<VALUE>)

Type: boolean

Default: True

Determines whether or not a dividers are drawn between the category levels of this axis. Only has an effect on "multicategory" axes.

showexponent

Code: fig.update_xaxes(showexponent=<VALUE>)

Type: enumerated , one of ("all" | "first" | "last" | "none")

Default: "all"

If "all", all exponents are shown besides their significands. If "first", only the exponent of the first tick is shown. If "last", only the exponent of the last tick is shown. If "none", no exponents appear.

showgrid

Code: fig.update_xaxes(showgrid=<VALUE>)

Type: boolean

Determines whether or not grid lines are drawn. If "True", the grid lines are drawn at every tick mark.

showline

Code: fig.update_xaxes(showline=<VALUE>)

Type: boolean

Determines whether or not a line bounding this axis is drawn.

showspikes

Code: fig.update_xaxes(showspikes=<VALUE>)

Type: boolean

Determines whether or not spikes (aka droplines) are drawn for this axis. Note: This only takes affect when hovermode = closest

showticklabels

Code: fig.update_xaxes(showticklabels=<VALUE>)

Type: boolean

Default: True

Determines whether or not the tick labels are drawn.

showtickprefix

Code: fig.update_xaxes(showtickprefix=<VALUE>)

Type: enumerated , one of ("all" | "first" | "last" | "none")

Default: "all"

If "all", all tick labels are displayed with a prefix. If "first", only the first tick is displayed with a prefix. If "last", only the last tick is displayed with a suffix. If "none", tick prefixes are hidden.

showticksuffix

axes(showticksuffix=<VALUE>)

, one of ("all" | "first" | "last" | "none")

xprefix` but for tick suffixes.

side

Code: `fig.update_xaxes(side=<VALUE>)`

Type: enumerated , one of (`"top"` | `"bottom"` | `"left"` | `"right"`)

Determines whether a x (y) axis is positioned at the "bottom" ("left") or "top" ("right") of the plotting area.

spikecolor

Code: `fig.update_xaxes(spikecolor=<VALUE>)`

Type: color

Sets the spike color. If undefined, will use the series color

spikedash

Code: `fig.update_xaxes(spikedash=<VALUE>)`

Type: string

Default: `"dash"`

Sets the dash style of lines. Set to a dash type string ("solid", "dot", "dash", "longdash", "dashdot", or "longdashdot") or a dash length list in px (eg "5px,10px,2px,2px").

spikemode

Code: `fig.update_xaxes(spikemode=<VALUE>)`

Type: flaglist string. Any combination of `"toaxis"` , `"across"` , `"marker"` joined with a `"+"`

Examples: `"toaxis"` , `"across"` , `"toaxis+across"` , `"toaxis+across+marker"`

Default: `"toaxis"`

Determines the drawing mode for the spike line If "toaxis", the line is drawn from the data point to the axis the series is plotted on. If "across", the line is drawn across the entire plot area, and supercedes "toaxis". If "marker", then a marker dot is drawn on the axis the series is plotted on

spikesnap

Code: `fig.update_xaxes(spikesnap=<VALUE>)`

Type: enumerated , one of (`"data"` | `"cursor"` | `"hovered data"`)

Default: `"hovered data"`

Determines whether spikelines are stuck to the cursor or to the closest datapoints.

spikethickness

Code: `fig.update_xaxes(spikethickness=<VALUE>)`

Type: number

Default: `3`

Sets the width (in px) of the zero line.

tick0

Code: `fig.update_xaxes(tick0=<VALUE>)`

Type: number or categorical coordinate string

Sets the placement of the first tick on this axis. Use with `dtick`. If the axis `type` is "log", then you must take the log of your starting tick (e.g. to set the starting tick to 100, set the `tick0` to 2) except when `dtick`="L<f>" (see `dtick` for more info). If the axis `type` is "date", it should be a date string, like date data. If the axis `type` is "category", it should be a number, using the scale where each category is assigned a serial number from zero in the order it appears.

tickangle

Code: `fig.update_xaxes(tickangle=<VALUE>)`

Type: angle

Default: `auto`

he tick labels with respect to the horizontal. For example, a `tickangle` of -90 draws the tick labels vertically.

tickcolor

Code: `fig.update_xaxes(tickcolor=<VALUE>)`

Type: color

Default: "#444"

Sets the tick color.

tickfont

Code: fig.update_xaxes(tickfont=dict(...))

Type: dict containing one or more of the keys listed below.

Sets the tick font.

color

Code: fig.update_xaxes(tickfont_color=<VALUE>)

Type: color

family

Code: fig.update_xaxes(tickfont_family=<VALUE>)

Type: string

HTML font family - the typeface that will be applied by the web browser. The web browser will only be able to apply a font if it is available on the system which it operates. Provide multiple font families, separated by commas, to indicate the preference in which to apply fonts if they aren't available on the system. The Chart Studio Cloud (at https://chart-studio.plotly.com or on-premise) generates images on a server, where only a select number of fonts are installed and supported. These include "Arial", "Balto", "Courier New", "Droid Sans", "Droid Serif", "Droid Sans Mono", "Gravitas One", "Old Standard TT", "Open Sans", "Overpass", "PT Sans Narrow", "Raleway", "Times New Roman".

lineposition

Code: fig.update_xaxes(tickfont_lineposition=<VALUE>)

Type: flaglist string. Any combination of "under", "over", "through" joined with a "+" OR "none".

Examples: "under", "over", "under+over", "under+over+through", "none"

Default: "none"

Sets the kind of decoration line(s) with text, such as an "under", "over" or "through" as well as combinations e.g. "under+over", etc.

shadow

Code: fig.update_xaxes(tickfont_shadow=<VALUE>)

Type: string

Default: "none"

Sets the shape and color of the shadow behind text. "auto" places minimal shadow and applies contrast text font color. See https://developer.mozilla.org/en-US/docs/Web/CSS/text-shadow for additional options.

size

Code: fig.update_xaxes(tickfont_size=<VALUE>)

Type: number greater than or equal to 1

style

Code: fig.update_xaxes(tickfont_style=<VALUE>)

Type: enumerated , one of ("normal" | "italic")

Default: "normal"

Sets whether a font should be styled with a normal or italic face from its family.

textcase

Code: fig.update_xaxes(tickfont_textcase=<VALUE>)

nerated , one of ("normal" | "word caps" | "upper" | "lower")

ormal"

lization of text. It can be used to make text appear in all-uppercase or all-lowercase, or with each word

variant

Code: fig.update_xaxes(tickfont_variant=<VALUE>)

Type: enumerated , one of ("normal" | "small-caps" | "all-small-caps" | "all-petite-caps" | "petite-caps" | "unicase")

Default: "normal"

Sets the variant of the font.

weight

Code: fig.update_xaxes(tickfont_weight=<VALUE>)

Type: integer between or equal to 1 and 1000

Default: normal

Sets the weight (or boldness) of the font.

tickformat

Code: fig.update_xaxes(tickformat=<VALUE>)

Type: string

Default: ""

Sets the tick label formatting rule using d3 formatting mini-languages which are very similar to those in Python. For numbers, see: https://github.com/d3/d3-format/tree/v1.4.5#d3-format. And for dates see: https://github.com/d3/d3-time-format/tree/v2.2.3#locale_format. We add two items to d3's date formatter: "%h" for half of the year as a decimal number as well as "%{n}f" for fractional seconds with n digits. For example, "2016-10-13 09:15:23.456" with tickformat "%H~%M~%S.%2f" would display "09~15~23.46"

tickformatstops

Code: fig.update_xaxes(tickformatstops=list(...))

Type: list of dict where each dict has one or more of the keys listed below.

dtickrange

Parent: layout.xaxis.tickformatstops[]

Type: list

range ["min", "max"], where "min", "max" - dtick values which describe some zoom level, it is possible to omit "min" or "max" value by passing "null"

enabled

Parent: layout.xaxis.tickformatstops[]

Type: boolean

Default: True

Determines whether or not this stop is used. If `False`, this stop is ignored even within its `dtickrange`.

name

Parent: layout.xaxis.tickformatstops[]

Type: string

When used in a template, named items are created in the output figure in addition to any items the figure already has in this array. You can modify these items in the output figure by making your own item with `templateitemname` matching this `name` alongside your modifications (including `visible: False` or `enabled: False` to hide it). Has no effect outside of a template.

templateitemname

Parent: layout.xaxis.tickformatstops[]

Type: string

Used to refer to a named item in this array in the template. Named items from the template will be created even without a matching item in the input figure, but you can modify one by making an item with `templateitemname` matching its `name`, alongside your modifications (including `visible: False` or `enabled: False` to hide it). If there is no template or no matching item, this item will be hidden unless you explicitly show it with `visible: True`.

value

Parent: layout.xaxis.tickformatstops[]

Type: string

Default: ""

string - dtickformat for described zoom level, the same as "tickformat"

ticklabelindex

Code: fig.update_xaxes(ticklabelindex=<VALUE>)

Type: integer or array of integers

Only for axes with `type` "date" or "linear". Instead of drawing the major tick label, draw the label for the minor tick that is n positions away from the major tick. E.g. to always draw the label for the minor tick before each major tick, choose `ticklabelindex` -1. This is useful for date axes with `ticklabelmode` "period" if you want to label the period that ends with each major tick instead of the period that begins there.

ticklabelmode

Code: fig.update_xaxes(ticklabelmode=<VALUE>)

Type: enumerated , one of ("instant" | "period")

Default: "instant"

Determines where tick labels are drawn with respect to their corresponding ticks and grid lines. Only has an effect for axes of `type` "date" When set to "period", tick labels are drawn in the middle of the period between ticks.

ticklabeloverflow

Code: fig.update_xaxes(ticklabeloverflow=<VALUE>)

Type: enumerated , one of ("allow" | "hide past div" | "hide past domain")

Determines how we handle tick labels that would overflow either the graph div or the domain of the axis. The default value for inside tick labels is "hide past domain". Otherwise on "category" and "multicategory" axes the default is "allow". In other cases the default is "hide past div".

ticklabelposition

Code: fig.update_xaxes(ticklabelposition=<VALUE>)

Type: enumerated , one of ("outside" | "inside" | "outside top" | "inside top" | "outside left" | "inside left" | "outside right" | "inside right" | "outside bottom" | "inside bottom")

Default: "outside"

Determines where tick labels are drawn with respect to the axis Please note that top or bottom has no effect on x axes or when `ticklabelmode` is set to "period". Similarly left or right has no effect on y axes or when `ticklabelmode` is set to "period". Has no effect on "multicategory" axes or when `tickson` is set to "boundaries". When used on axes linked by `matches` or `scaleanchor`, no extra padding for inside labels would be added by autorange, so that the scales could match.

ticklabelshift

Code: fig.update_xaxes(ticklabelshift=<VALUE>)

Type: integer

Default: 0

Shifts the tick labels by the specified number of pixels in parallel to the axis. Positive values move the labels in the positive direction of the axis.

ticklabelstandoff

Code: fig.update_xaxes(ticklabelstandoff=<VALUE>)

Type: integer

Default: 0

distance (in px) between the axis tick labels and their default position. A positive `ticklabelstandoff` moves the y from the plot area if `ticklabelposition` is "outside", and deeper into the plot area if `ticklabelposition` is e `ticklabelstandoff` works in the opposite direction, moving outside ticks towards the plot area and inside outside. If the negative value is large enough, inside ticks can even end up outside and vice versa.

ticklabelstep

Code: `fig.update_xaxes(ticklabelstep=<VALUE>)`

Type: integer greater than or equal to 1

Default: `1`

Sets the spacing between tick labels as compared to the spacing between ticks. A value of 1 (default) means each tick gets a label. A value of 2 means shows every 2nd label. A larger value n means only every nth tick is labeled. `tick0` determines which labels are shown. Not implemented for axes with `type` "log" or "multicategory", or when `tickmode` is "array".

ticklen

Code: `fig.update_xaxes(ticklen=<VALUE>)`

Type: number greater than or equal to 0

Default: `5`

Sets the tick length (in px).

tickmode

Code: `fig.update_xaxes(tickmode=<VALUE>)`

Type: enumerated , one of (`"auto"` | `"linear"` | `"array"` | `"sync"`)

Sets the tick mode for this axis. If "auto", the number of ticks is set via `nticks`. If "linear", the placement of the ticks is determined by a starting position `tick0` and a tick step `dtick` ("linear" is the default value if `tick0` and `dtick` are provided). If "array", the placement of the ticks is set via `tickvals` and the tick text is `ticktext`. ("array" is the default value if `tickvals` is provided). If "sync", the number of ticks will sync with the overlayed axis set by `overlaying` property.

tickprefix

Code: `fig.update_xaxes(tickprefix=<VALUE>)`

Type: string

Default: `""`

Sets a tick label prefix.

ticks

Code: `fig.update_xaxes(ticks=<VALUE>)`

Type: enumerated , one of (`"outside"` | `"inside"` | `""`)

Determines whether ticks are drawn or not. If "", this axis' ticks are not drawn. If "outside" ("inside"), this axis' are drawn outside (inside) the axis lines.

tickson

Code: `fig.update_xaxes(tickson=<VALUE>)`

Type: enumerated , one of (`"labels"` | `"boundaries"`)

Default: `"labels"`

Determines where ticks and grid lines are drawn with respect to their corresponding tick labels. Only has an effect for axes of `type` "category" or "multicategory". When set to "boundaries", ticks and grid lines are drawn half a category to the left/bottom of labels.

ticksuffix

Code: `fig.update_xaxes(ticksuffix=<VALUE>)`

Type: string

Default: `""`

Sets a tick label suffix.

ticktext

Code: `fig.update_xaxes(ticktext=<VALUE>)`

...rray, or Pandas series of numbers, strings, or datetimes.

...ayed at the ticks position via `tickvals`. Only has an effect if `tickmode` is set to "array". Used with `tickvals`.

tickvals

Code: `...axes(tickvals=<VALUE>)`

Type: list, numpy array, or Pandas series of numbers, strings, or datetimes.

Sets the values at which ticks on this axis appear. Only has an effect if `tickmode` is set to "array". Used with `ticktext`.

tickwidth

Code: fig.update_xaxes(tickwidth=<VALUE>)

Type: number greater than or equal to 0

Default: 1

Sets the tick width (in px).

title

Code: fig.update_xaxes(title=dict(...))

Type: dict containing one or more of the keys listed below.

font

Code: fig.update_xaxes(title_font=dict(...))

Type: dict containing one or more of the keys listed below.

Sets this axis' title font.

color

Code: fig.update_xaxes(title_font_color=<VALUE>)

Type: color

family

Code: fig.update_xaxes(title_font_family=<VALUE>)

Type: string

HTML font family - the typeface that will be applied by the web browser. The web browser will only be able to apply a font if it is available on the system which it operates. Provide multiple font families, separated by commas, to indicate the preference in which to apply fonts if they aren't available on the system. The Chart Studio Cloud (at https://chart-studio.plotly.com or on-premise) generates images on a server, where only a select number of fonts are installed and supported. These include "Arial", "Balto", "Courier New", "Droid Sans", "Droid Serif", "Droid Sans Mono", "Gravitas One", "Old Standard TT", "Open Sans", "Overpass", "PT Sans Narrow", "Raleway", "Times New Roman".

lineposition

Code: fig.update_xaxes(title_font_lineposition=<VALUE>)

Type: flaglist string. Any combination of "under", "over", "through" joined with a "+" OR "none".

Examples: "under", "over", "under+over", "under+over+through", "none"

Default: "none"

Sets the kind of decoration line(s) with text, such as an "under", "over" or "through" as well as combinations e.g. "under+over", etc.

shadow

Code: fig.update_xaxes(title_font_shadow=<VALUE>)

Type: string

Default: "none"

Sets the shape and color of the shadow behind text. "auto" places minimal shadow and applies contrast text font color. See https://developer.mozilla.org/en-US/docs/Web/CSS/text-shadow for additional options.

size

Code: fig.update_xaxes(title_font_size=<VALUE>)

Type: number greater than or equal to 1

style

Code: fig.update_xaxes(title_font_style=<VALUE>)

Type: enumerated , one of ("normal" | "italic")

Default: "normal"

Sets whether a font should be styled with a normal or italic face from its family.

textcase

Code: fig.update_xaxes(title_font_textcase=<VALUE>)

Type: enumerated , one of ("normal" | "word caps" | "upper" | "lower")

Default: "normal"

Sets capitalization of text. It can be used to make text appear in all-uppercase or all-lowercase, or with each word capitalized.

variant

Code: fig.update_xaxes(title_font_variant=<VALUE>)

Type: enumerated , one of ("normal" | "small-caps" | "all-small-caps" | "all-petite-caps" | "petite-caps" | "unicase")

Default: "normal"

Sets the variant of the font.

weight

Code: fig.update_xaxes(title_font_weight=<VALUE>)

Type: integer between or equal to 1 and 1000

Default: normal

Sets the weight (or boldness) of the font.

standoff

Code: fig.update_xaxes(title_standoff=<VALUE>)

Type: number greater than or equal to 0

Sets the standoff distance (in px) between the axis labels and the title text The default value is a function of the axis tick labels, the title `font.size` and the axis `linewidth`. Note that the axis title position is always constrained within the margins, so the actual standoff distance is always less than the set or default value. By setting `standoff` and turning on `automargin`, plotly.js will push the margins to fit the axis title at given standoff distance.

text

Code: fig.update_xaxes(title_text=<VALUE>)

Type: string

Sets the title of this axis.

type

Code: fig.update_xaxes(type=<VALUE>)

Type: enumerated , one of ("-" | "linear" | "log" | "date" | "category" | "multicategory")

Default: "-"

Sets the axis type. By default, plotly attempts to determined the axis type by looking into the data of the traces that referenced the axis in question.

uirevision

Code: fig.update_xaxes(uirevision=<VALUE>)

Type: number or categorical coordinate string

Controls persistence of user-driven changes in axis `range`, `autorange`, and `title` if in `editable: True` configuration. Defaults to `layout.uirevision`.

axes(visible=<VALUE>)

ide the axis while preserving interaction like dragging. Default is True when a cheater plot is present on the
lse

zeroline

Code: `fig.update_xaxes(zeroline=<VALUE>)`

Type: boolean

Determines whether or not a line is drawn at along the 0 value of this axis. If "True", the zero line is drawn on top of the grid lines.

zerolinecolor

Code: `fig.update_xaxes(zerolinecolor=<VALUE>)`

Type: color

Default: `"#444"`

Sets the line color of the zero line.

zerolinewidth

Code: `fig.update_xaxes(zerolinewidth=<VALUE>)`

Type: number

Default: `1`

Sets the width (in px) of the zero line.