aquarel

Release 0.0.5

Lukas Gienapp

Nov 10, 2022

CONTENTS:

Aquarel is a lightweight templating engine and wrapper around Matplotlibs' `rcparams` to make styling plots simple. Aquarel templates can be defined programmatically and be serialized and shared in a JSON format.

Full documentation is available at aquarel.readthedocs.io.

CHAPTER

ONE

INSTALLATION

Install via pip:

```
python -m pip install aquarel
```

CHAPTER

TWO

USAGE

2.1 Applying a style

Styles can be either applied globally

```
from aquarel import load_theme

theme = load_theme("arctic_light")
theme.apply()
# ... plotting code here
theme.apply_transforms()
```

...or with a context manager:

```
from aquarel import load_theme

with load_theme("arctic_light"):
    figure = # ... plotting code here
```

2.2 Transforms

Themes may specify *transforms*. Transforms are functions applied on the finished plot to achieve aesthetics that are not possibly by means of `rcparams` only. For example, to trim the axes, one could apply the `trim` transform:

```
from aquarel import load_theme

with load_theme("arctic_light").set_transforms(trim=True):
    figure = # ... plotting code here

# plt.show() or savefig() have to be called outside the context manager to have the␣
↪transforms correctly applied.
figure.savefig()
```

However, there is one important thing to keep in mind: since transforms require the matplotlib figure/axes object to be present and finished, they have to be applied **after** the plotting code. When using a theme with a context manager, this is automatically done in the `__exit__` call. If global usage is desired, `Theme.apply_transforms()` has to be called after every figure. This also means that calls that make use of the finished figure, i.e. `plt.show` or `plt.savefig` have to commence after transform application, so **outside** the context manager.

2.3 Customization & Theme Creation

Besides loading a predefined theme, you can create a new theme

```
from aquarel import Theme

theme = (
    Theme(name="demo", description="A demo theme.")
    .set_grid(draw=True, width=0.5)
    .set_font(family="monospace")
    .set_color(grid_color="blue")
)
```

...modify an existing one

```
from aquarel import load_theme

theme = (
    load_theme("arctic_light")
    .set_grid(width=2)
)
```

...and write and load your custom styles to and from disk:

```
from aquarel import Theme

theme = Theme.from_file("custom.json")
theme.save("custom.json")
```

If the simplified API of aquarel is not sufficient for your use-case, you can also directly modify the underlying `rcparams` with overrides:

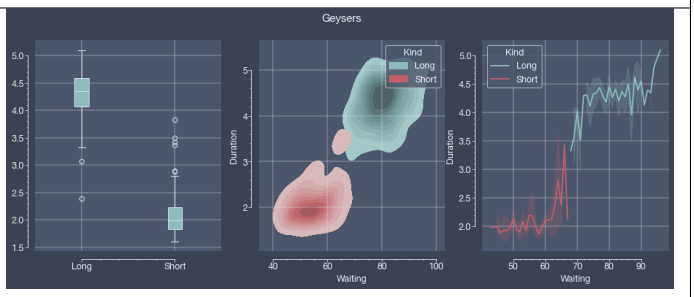
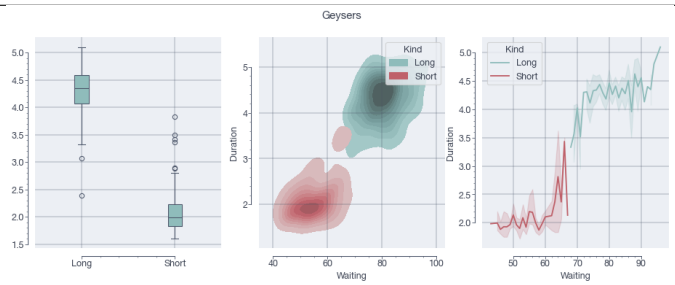
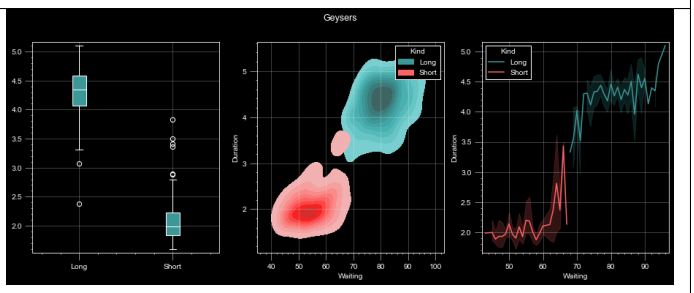
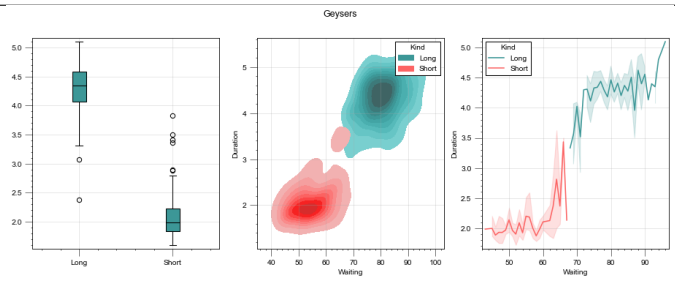
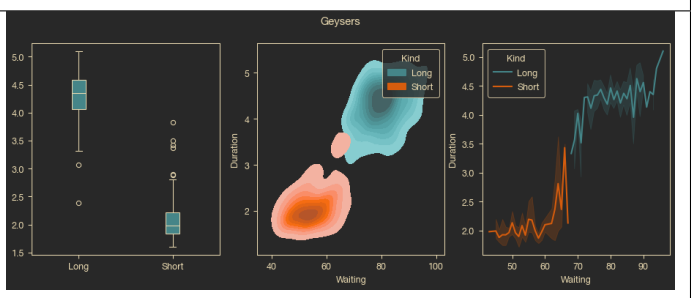
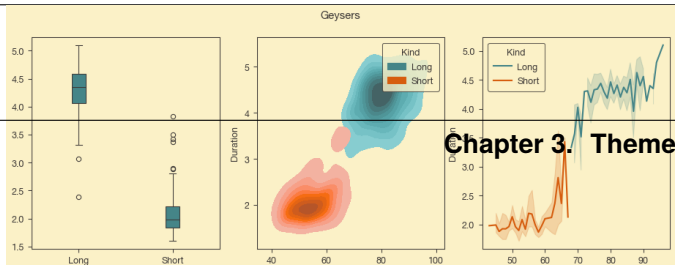
```
from aquarel import load_theme

theme = load_theme("arctic_light").set_overrides({
    "ytick.minor.visible": False,
    "xtick.minor.visible": True
})
```

CHAPTER

THREE

THEMES

aquarel ships with several pre-defined themes that are designed to showcase its templating capabilities. Add your own with a pull request!

Name	Description	Preview
`arctic_dark`	Frosty dark theme based on the nord color scheme	
`arctic_light`	Frosty dark theme based on the nord color scheme	
`boxy_dark`	Dark theme with enclosing box and grid	
`boxy_light`	Light theme with enclosing box and grid	
`gruvbox_dark`	Dark theme with pastel retro groove colors	
		

CHAPTER

FOUR

FAQ

4.1 How is this different from matplotlib style sheets?

`aquarel` is a wrapper around the stylesheets, so everything you can do with stylesheets can be achieved with `aquarel`. However there are some notable shortcomings of stylesheets that `aquarel` adresses:

1. **On-the-fly templating** – the stylesheets are applied once and are then used for every plot in the current plotting context (py-file, notebook, ipython session, . . .). `aquarel` takes a different approach here and aims to provide per-plot styling with optional temporary changes. The style `aquarel` applies lasts throughout the context manager (`with aquarel.Theme:`), and switches back to whatever is the global default style outside of it. This allows you to do plot-level temporary changes. You have one plot in your notebook that needs no minor ticks? just `with theme.set_ticks():` for this plot only.
2. **Simplified templating**: matplotlib stylesheets have a lot of redundant keys for most applications. For example, you rarely want to have different colors for both axes; while possible with a stylefile, its cumbersome to change all the different keys to achieve a uniform look. `aquarel` simplifies this with e.x. a single `set_color(ticks="#eee")` call, which changes all related and relevant keys for ticks. Note that this simplifies the API, but does not restrict capabilities: the `set_overrides` method accepts every possible stylefile key if you want to access low-level styling.
3. **Transforms**: some style elements, like trimmed axes, are not achievable with stylesheets alone (see README for more informations). `aquarel` defines a few of these transforms (and hopefully many more in the future), and makes them persistable and shareable through aquarel themes. Instead of having to apply a seaborn despine after every plot, you can have a global style that specifies a trim, and have consistent styling throughout with minimal code repetition.

CHAPTER

FIVE

DOCUMENTATION

5.1 Theme

class aquarel.theme.**Theme**(*name=None*, *description=None*)

Bases: object

Parameters

- **name** (*Optional[str]*) –
- **description** (*Optional[str]*) –

apply()

Applies the theme

apply_transforms()

Applies the themes' transforms

classmethod from_dict(*data*)

Initialize a theme from a dictionary

Parameters
data (*dict*) – theme dictionary to initialize from

Returns
cls

classmethod from_file(*filename*)

Initialize a theme from a theme file

Parameters
filename (*str*) – file to load theme dictionary from

Returns
cls

save(*path*)

Write the template to a JSON template file

Parameters
path (*str*) – file to write the template to

set_axes(*width=None*, *top=None*, *bottom=None*, *left=None*, *right=None*, *xmargin=None*, *ymargin=None*, *zmargin=None*)

Set axis styling options

Parameters

- **width** (*Optional[Union[float, int]]*) – axis line width, default: 1.0
- **top** (*Optional[bool]*) – display top axis, default: True
- **bottom** (*Optional[bool]*) – display bottom axis, default: True
- **left** (*Optional[bool]*) – display left axis, default: True
- **right** (*Optional[bool]*) – display right axis, default: True
- **xmargin** (*Optional[float]*) – padding added to the x-axis, expressed as margin times the data interval, default: 0.05
- **ymargin** (*Optional[float]*) – padding added to the y-axis, expressed as margin times the data interval, default: 0.05
- **zmargin** (*Optional[float]*) – padding added to the z-axis, expressed as margin times the data interval, default: 0.05

Returns
self

set_axis_labels(*pad=None*, *size=None*, *weight=None*)

Set axis label styling options.

Parameters

- **pad** (*Optional[Union[float, int]]*) – padding of the axis label
- **size** (*Optional[str]*) – font size of the axis label, can be {"xx-small", "x-small", "small", "medium", "large", "x-large", "xx-large"}, default: "medium"
- **weight** (*Optional[str]*) – font weight of the axis label, can be {"ultralight", "light", "normal", "regular", "book", "medium", "roman", "semibold", "demibold", "demi", "bold", "heavy", "extra bold", "black"}, default: "normal"

Returns
self

set_color(*palette=None*, *figure_background_color=None*, *plot_background_color=None*, *text_color=None*, *axes_color=None*, *axes_label_color=None*, *line_color=None*, *grid_color=None*, *tick_color=None*, *tick_label_color=None*, *legend_background_color=None*, *legend_border_color=None*)

Sets color options.

Parameters

- **palette** (*Optional[List[str]]*) – The color palette to cycle through for plot elements, should be list of valid color arguments
- **figure_background_color** (*Optional[str]*) – the background color of the whole figure
- **plot_background_color** (*Optional[str]*) – the background color of the plot only
- **text_color** (*Optional[str]*) – color of text elements (plot title, axis title)
- **axes_color** (*Optional[str]*) – the color of the axis lines
- **axes_label_color** (*Optional[str]*) – the color of the axis labels
- **line_color** (*Optional[str]*) – the line color
- **grid_color** (*Optional[str]*) – the color of the grid lines
- **tick_color** (*Optional[str]*) – the color of the ticks

- **tick_label_color** (*Optional[str]*) – the color of the tick labels
- **legend_border_color** (*Optional[str]*) – color of the legend border
- **legend_background_color** (*Optional[str]*) – color of the legend background

Returns
self

set_font(*family=None*, *cursive=None*, *fantasy=None*, *monospace=None*, *sans_serif=None*, *serif=None*, *size=None*, *style=None*, *variant=None*, *weight=None*)

Set font styling options.

Parameters

- **family** (*Optional[str]*) – font family to use, can be {}, default: sans-serif
- **cursive** (*Optional[Union[str, List[str]]]*) – which font(s) to use for cursive text
- **fantasy** (*Optional[Union[str, List[str]]]*) – which font(s) to use for fantasy text
- **monospace** (*Optional[Union[str, List[str]]]*) – which font(s) to use for monospace text
- **sans_serif** (*Optional[Union[str, List[str]]]*) – which font(s) to use for sans-serif text
- **serif** (*Optional[Union[str, List[str]]]*) – which font(s) to use for serif text
- **size** (*Optional[Union[float, int]]*) – base font size in pt that all other elements scale relative to, default: 10.0
- **style** (*Optional[str]*) – font style, can be {"normal", "roman", "italic", "oblique"}, default: normal
- **variant** (*Optional[str]*) – font variant, can be {"normal", "small-caps"}, default: normal
- **weight** (*Optional[Union[float, int, str]]*) – font weight, can be {"ultralight", "light", "normal", "regular", "book", "medium", "roman", "semibold", "demibold", "demi", "bold", "heavy", "extra bold", "black"}, default: normal

Returns
self

set_grid(*draw=None*, *axis=None*, *ticks=None*, *alpha=None*, *style=None*, *width=None*)

Set grid styling options.

Parameters

- **draw** (*Optional[bool]*) – True if grid should be drawn, False otherwise, default: False
- **axis** (*Optional[str]*) – axes along which the grid should be drawn, can be {"both", "x", "y"}, default: "both"
- **ticks** (*Optional[str]*) – which tick level to base the grid on, can be {"major", "minor", "both"}, default: "major"
- **alpha** (*Optional[float]*) – the alpha level to draw the grid with, can be float between 0 and 1, default: 1.0
- **style** (*Optional[str]*) – the line style to draw the grid with, can be {"-", "–", "-.", ":", ""}, default: "-"
- **width** (*Optional[float]*) – the line width to draw the grid with in pt, default: 0.8

Returns
self

`set_legend`(*location=None*, *round=None*, *shadow=None*, *title_size=None*, *text_size=None*, *alpha=None*, *marker_scale=None*, *padding=None*, *margin=None*, *spacing=None*)

Set legend styling options.

Parameters

- **location** (*Optional[str]*) – The location of the legend. Can be {'best', 'upper right', 'upper left', 'lower left', 'lower right', 'right', 'center left', 'center right', 'lower center', 'upper center', 'center'} or a 2-tuple giving the coordinates of the lower-left corner. Default: 'best'
- **round** (*Optional[bool]*) – indicates if legend corners should be rounded or rectangular. Default: True
- **shadow** (*Optional[bool]*) – indicates if the legend should cast a shadow. Default: False
- **title_size** (*Optional[str]*) – font size of the legend title, can be {"xx-small", "x-small", "small", "medium", "large", "x-large", "xx-large"}, default: "medium"
- **text_size** (*Optional[str]*) – font size of the legend title, can be {"xx-small", "x-small", "small", "medium", "large", "x-large", "xx-large"}, default: "medium"
- **alpha** (*Optional[float]*) – transparency of the legend patch.
- **marker_scale** (*Optional[float]*) – the relative size of legend markers. Default: 1.0
- **padding** (*Optional[Union[float, int]]*) – space between legend border and legend content in pt. Default: 0.4
- **margin** (*Optional[Union[float, int]]*) – space between legend border and axes in pt. Default: 0.5
- **spacing** (*Optional[Union[float, int]]*) – spacing of legend elements in pt. Default: 0.5

Returns
self

`set_lines`(*style=None*, *width=None*)

Set line styling options.

Parameters

- **style** (*Optional[str]*) – the style to draw lines with, can be {"-", "–", "-.", ":", ""}, default: "-"
- **width** (*Optional[float]*) – the width to draw lines with in pt, default: 1.5

Returns
self

`set_overrides`(*rc*)

Set custom overrides of rcparam parameters directly

Parameters
rc (*dict*) – Dict of valid matplotlib rcparams

Returns
self

set_tick_labels(*location=None*, *size=None*, *left=None*, *right=None*, *bottom=None*, *top=None*)

Set tick label styling options.

Parameters

- **location** (*Optional[str]*) – location of the tick labels, can be {"left", "right", "bottom", "top", "center"}, default: center
- **size** (*Optional[str]*) – size of the tick label, can be {"xx-small", "x-small", "small", "medium", "large", "x-large", "xx-large"}, default: "normal"
- **left** (*Optional[bool]*) – whether to draw the tick labels to the left of the y-axis, default: True
- **right** (*Optional[bool]*) – whether to draw the tick labels to the right of the y-axis, default: False
- **bottom** (*Optional[bool]*) – whether to draw the tick labels at the bottom of the x-axis, default: True
- **top** (*Optional[bool]*) – whether to draw the tick labels at the top of the x-axis, default: False

Returns

self

set_ticks(*x_align=None*, *y_align=None*, *direction=None*, *draw_minor=None*, *width_major=None*, *width_minor=None*, *size_major=None*, *size_minor=None*, *pad_major=None*, *pad_minor=None*)

Set styling options for ticks.

Parameters

- **x_align** (*Optional[str]*) – alignment of ticks along the horizontal axes, can be {"center", "right", "left"}, default: "center"
- **y_align** (*Optional[str]*) – alignment of ticks along the vertical axes, can be {"center", "top", "bottom", "baseline", "center_baseline"}, default: "center_baseline"
- **direction** (*Optional[str]*) – direction the ticks should be facing, can be {"in", "out", "inout"}, default: "out"
- **draw_minor** (*Optional[bool]*) – whether to draw minor ticks, default: False
- **width_major** (*Optional[Union[float, int]]*) – width of major ticks in pt, default: 0.8
- **width_minor** (*Optional[Union[float, int]]*) – width of minor ticks in pt, default: 0.6
- **size_major** (*Optional[Union[float, int]]*) – size of major ticks in pt, default: 3.5
- **size_minor** (*Optional[Union[float, int]]*) – size of minor ticks in pt, default: 2
- **pad_major** (*Optional[Union[float, int]]*) – padding of major ticks in pt, default: 3.5
- **pad_minor** (*Optional[Union[float, int]]*) – padding of minor ticks in pt, default: 3.4

Returns

self

set_title(*location=None*, *size=None*, *weight=None*, *pad=None*)

Sets title styling options.

Parameters

- **location** (*Optional[str]*) – the location of the title, one of {left, right, center} default: 'center'
- **pad** (*Optional[float]*) – pad between axes and title in pt, default: 6.0
- **size** (*Optional[Union[float, str]]*) – the font size of the title, float or one of {'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'}, default: 'large'
- **weight** (*Optional[Union[float, int, str]]*) – the font weight of the title, int in range 0-1000 or one of {'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'}, default: "normal"

Returns

self

set_transforms(*trim=None*, *offset=None*, *rotate_xlabel=None*, *rotate_ylabel=None*)

Set the transforms

Parameters

- **trim** (*Optional[bool]*) – trims the axes to the nearest major tick, can be {"x", "y", "both"}
- **offset** (*Optional[int]*) – offset shift of the axes in pt. Applies to all axes
- **rotate_xlabel** (*Optional[int]*) – rotation of x-axis labels in degrees
- **rotate_ylabel** (*Optional[int]*) – rotation of y-axis labels in degrees
- **log_axes** – set log scale for the specified axes, can be {'both', 'x', 'y'}

Returns

self

5.2 Transforms

aquarel.transforms.**offset**(*distance*)

Offsets the plot spines. Code partly taken from https://github.com/mwaskom/seaborn/blob/563e96d3be1eaee8db8dfbccf7eed1f1c66dfd31/seaborn/utils.py#L292

Parameters

distance (*int*) – offset distance int pt.

aquarel.transforms.**rotate_xlabel**(*degrees*)

Rotates the x-labels of the current plot.

Parameters

degrees (*int*) – rotation in degrees

aquarel.transforms.**rotate_ylabel**(*degrees*)

Rotates the y-labels of the current plot.

Parameters

degrees (*int*) – rotation in degrees

`aquarel.transforms.trim(axis)`

Trims axes of a plot to first and last major tick. Code partly taken from https://github.com/mwaskom/seaborn/blob/563e96d3be1eaee8db8dfbccf7eed1f1c66dfd31/seaborn/utils.py#L292

Parameters
axis (*str*) – axes to apply the trim to. Can be {"x", "y", "both"}.

5.3 Utils

`aquarel.utils.list_themes()`

Returns a list of available theme names.

Returns
a list of available theme names

`aquarel.utils.load_theme(theme_name)`

Sets the chosen style and color palette globally.

Parameters
theme_name (*str*) – name of the theme to load

Returns
the specified Theme

Raises
ValueError – if a theme is not found

`aquarel.utils.make_samples()`

Generates sample plots for all themes to be used in documentation

5.4 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX