Styling Plotly Express Figures in Python

Figures made with Plotly Express can be customized in all the same ways as figures made with graph objects, as well as with PX-specific function arguments.

Styling Figures made with Plotly Express

Plotly Express (/python/plotly-express/) is the easy-to-use, high-level interface to Plotly, which operates on a variety of types of data (/python/px-arguments/). Every Plotly Express function returns a `plotly.graph_objects.Figure` object (/python/graph-objects/) whose `data` and `layout` has been pre-populated according to the provided arguments.

> You can style and customize figures made with Plotly Express *in all the same ways* as you can style figures made more manually by explicitly assembling `graph_objects` into a figure.

More specifically, here are the 4 ways you can style and customize figures made with Plotly Express:

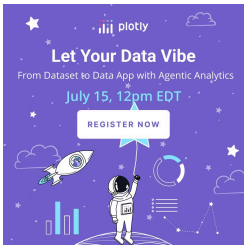
1. Control common parameters like width & height, titles, labeling and colors using built-in Plotly Express function arguments
2. Updating the figure attributes using update methods or by directly setting attributes (/python/creating-and-updating-figures/)
3. Using Plotly's theming/templating mechanism (/python/templates/) via the `template` argument to every Plotly Express function
4. Setting default values for common parameters using `px.defaults`

Built-in Plotly Express Styling Arguments

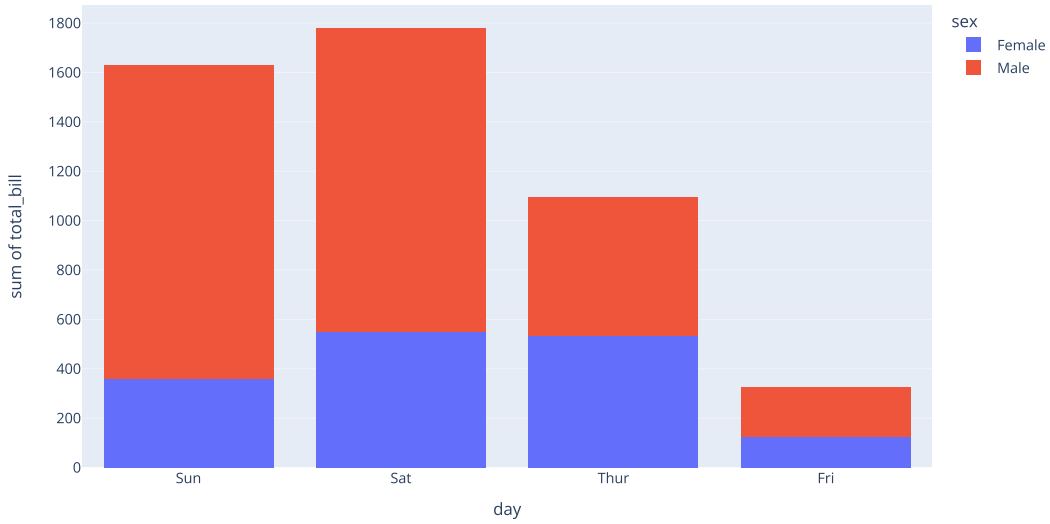
Many common styling options can be set directly in the px function call. Every Plotly Express function accepts the following arguments:

- `title` to set the figure title
- `width` and `height` to set the figure dimensions
- `template` to set many styling parameters at once (/python/templates/) (see below for more details)
- `labels` to override the default axis and legend labels behaviour, which is to use the data frame column name if available, and otherwise to use the label name itself like "x", "y", "color" etc. `labels` accepts a `dict` whose keys are the label to rename and whose values are the desired labels. These labels appear in axis labels, legend and color bar titles, and in hover labels.
- `category_orders` to override the default category ordering behaviour, which is to use the order in which the data appears in the input. `category_orders` accepts a `dict` whose keys are the column name to reorder and whose values are a `list` of values in the desired order. These orderings apply everywhere categories appear: in legends, on axes, in bar stacks, in the order of facets, in the order of animation frames etc.
- `hover_data` and `hover_name` to control which attributes appear in the hover label and how they are formatted.
- Various color-related attributes (/python/colorscales/) such as `color_continuous_scale`, `color_range`, `color_discrete_sequence` and/or `color_discrete_map` set the colors used in the figure. `color_discrete_map` accepts a dict whose keys are values mapped to `color` and whose values are the desired CSS colors.

To illustrate each of these, here is a simple, default figure made with Plotly Express. Note the default orderings for the x-axis categories and the usage of lowercase & snake_case data frame columns for axis labelling.

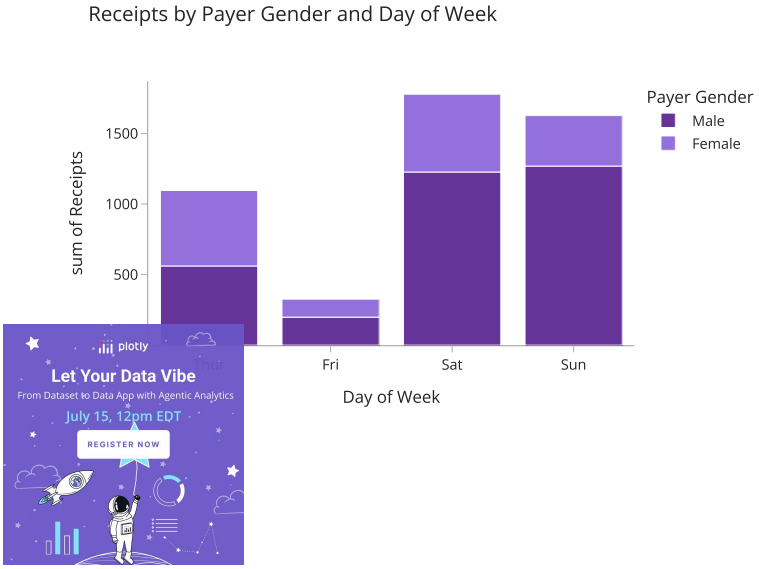


```
import plotly.express as px
df = px.data.tips()
fig = px.histogram(df, x="day", y="total_bill", color="sex")
fig.show()
```



Here is the same figure, restyled by adding some extra parameters to the initial Plotly Express call:

```
import plotly.express as px
df = px.data.tips()
fig = px.histogram(df, x="day", y="total_bill", color="sex",
            title="Receipts by Payer Gender and Day of Week",
            width=600, height=400,
            labels={ # replaces default labels by column name
                "sex": "Payer Gender",  "day": "Day of Week", "total_bill": "Receipts"
            },
            category_orders={ # replaces default order by column name
                "day": ["Thur", "Fri", "Sat", "Sun"], "sex": ["Male", "Female"]
            },
            color_discrete_map={ # replaces default color mapping by value
                "Male": "RebeccaPurple", "Female": "MediumPurple"
            },
            template="simple_white"
            )
fig.show()
```

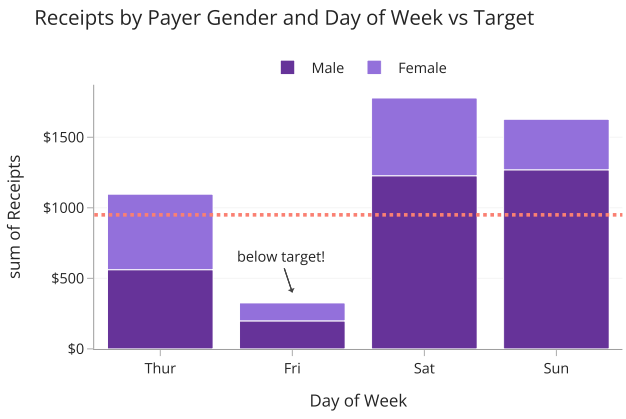


Updating or Modifying Figures made with Plotly Express

If none of the built-in Plotly Express arguments allow you to customize the figure the way you need to, you can use the `update_*` and `add_*` methods (/python/creating-and-updating-figures/) on the `plotly.graph_objects.Figure` object (/python/graph-objects/) returned by the PX function to make any further modifications to the figure. This approach is the one used throughout the Plotly.py documentation to customize axes (/python/axes/), control legends (/python/legend/) and colorbars (/python/colorscales/), add shapes (/python/shapes/) and annotations (/python/text-and-annotations/) etc.

Here is the same figure as above, with some additional customizations to the axes and legend via `.update_yaxes()`, and `.update_layout()`, as well as some annotations added via `.add_shape()` and `.add_annotation()`.

```
import plotly.express as px
df = px.data.tips()
fig = px.histogram(df, x="day", y="total_bill", color="sex",
             title="Receipts by Payer Gender and Day of Week vs Target",
             width=600, height=400,
             labels={"sex": "Payer Gender",  "day": "Day of Week", "total_bill": "Receipts"},
             category_orders={"day": ["Thur", "Fri", "Sat", "Sun"], "sex": ["Male", "Female"]},
             color_discrete_map={"Male": "RebeccaPurple", "Female": "MediumPurple"},
             template="simple_white"
             )

fig.update_yaxes( # the y-axis is in dollars
    tickprefix="$", showgrid=True
)

fig.update_layout( # customize font and legend orientation & position
    font_family="Rockwell",
    legend=dict(
        title=None, orientation="h", y=1, yanchor="bottom", x=0.5, xanchor="center"
    )
)

fig.add_shape( # add a horizontal "target" line
    type="line", line_color="salmon", line_width=3, opacity=1, line_dash="dot",
    x0=0, x1=1, xref="paper", y0=950, y1=950, yref="y"
)

fig.add_annotation( # add a text callout with arrow
    text="below target!", x="Fri", y=400, arrowhead=1, showarrow=True
)

fig.show()
```



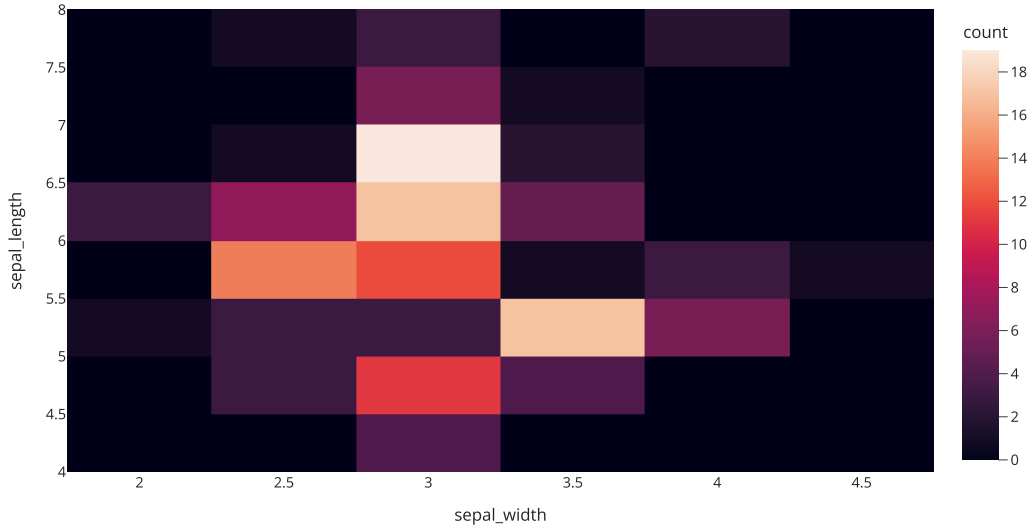
How Plotly Express Works with Templates

Plotly has a theming system based on templates (/python/templates/) and figures created with Plotly Express interact smoothly with this system:

- Plotly Express methods will use the default template if one is set in `plotly.io` (by default, this is set to `plotly`) or in `plotly.express.defaults` (see below)
- The template in use can always be overridden via the `template` argument to every PX function
- The default `color_continuous_scale` will be the value of `layout.colorscales.sequential` in the template in use, unless it is overridden via the corresponding function argument or via `plotly.express.defaults` (see below)
- The default `color_discrete_sequence` will be the value of `layout.colorway` in the template in use, unless it is overridden via the corresponding function argument or via `plotly.express.defaults` (see below)

By way of example, in the following figure, simply setting the `template` argument will automatically change the default continuous color scale, even though we have not specified `color_continuous_scale` directly.

```
import plotly.express as px
df = px.data.iris()
fig = px.density_heatmap(df, x="sepal_width", y="sepal_length", template="seaborn")
fig.show()
```



Setting Plotly Express Styling Defaults

Plotly Express supports a simple default-configuration system via the `plotly.express.defaults` singleton object. The values of the properties set on this object are used for the rest of the active session in place of None as the default values for any argument to a PX function with a matching name:

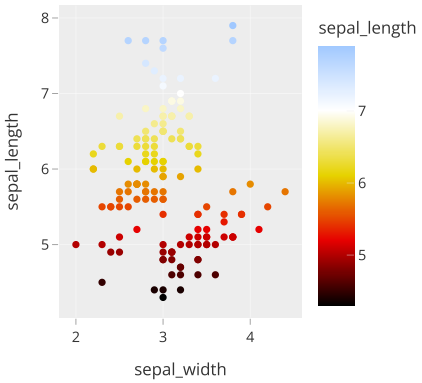
- `width` and `height` can be set once globally for all Plotly Express functions
- `template` can override the setting of `plotly.io.templates.default` for all Plotly Express functions
- `color_continuous_scale` and `color_discrete_scale` can override the contents of the template in use for all Plotly Express functions that accept these arguments
- `line_dash_sequence`, `symbol_sequence` and `size_max` can be set once globally for all Plotly Express functions that accept these arguments

To illustrate this "defaults hierarchy", in the following example:

- we set the Plotly-wide default template to `simple_white`, but
- we override the default template for Plotly Express to be `ggplot2`, but
- we also set the default `color_continuous_scale`, and
- we set the default `height` and `width` to 400 by 600, but
- we override the default `width` to 400 via the function argument.

The figure produced with Plotly Express thereafter uses the `ggplot2` settings for all attributes except for the continuous color scale (visible because the figure has a plot background, and neither the `simple_white` nor `ggplot2` template uses Blackbody as a color scale), and uses the Plotly Express defaults for the figure height and plot width (visible because the figure height is the same as the figure width, despite the default).

```
import plotly.express as px
import plotly.io as pio

pio.templates.default = "simple_white"

px.defaults.template = "ggplot2"
px.defaults.color_continuous_scale = px.colors.sequential.Blackbody
px.defaults.width = 600
px.defaults.height = 400

df = px.data.iris()
fig = px.scatter(df, x="sepal_width", y="sepal_length", color="sepal_length", width=400)
fig.show()
```



What About Dash?

Dash (https://dash.plot.ly/) is an open-source framework for building analytical applications, with no Javascript required, and it is tightly integrated with the Plotly graphing library.

Learn about how to install Dash at https://dash.plot.ly/installation (https://dash.plot.ly/installation).

Everywhere in this page that you see fig.show(), you can display the same figure in a Dash application by passing it to the figure argument of the Graph component (https://dash.plot.ly/dash-core-components/graph) from the built-in dash_core_components package like this:

```
import plotly.graph_objects as go # or plotly.express as px
fig = go.Figure() # or any Plotly Express function e.g. px.bar(...)
# fig.add_trace( ... )
# fig.update_layout( ... )

from dash import Dash, dcc, html

app = Dash()
app.layout = html.Div([
    dcc.Graph(figure=fig)
])

app.run(debug=True, use_reloader=False)  # Turn off reloader if inside Jupyter
```

JOIN OUR MAILING LIST

Sign up to stay in the loop with all things Plotly — from Dash Club to product updates, webinars, and more!

SUBSCRIBE (HTTPS://GO.PLOT.LY/SUBSCRIPTION)

Products

Dash (https://plotly.com/dash/)
Consulting and Training (https://plotly.com/consulting-and-oem/)

Pricing

Enterprise Pricing (https://plotly.com/get-pricing/)

ıress
ıents
e with
plates
ts

About Us

Careers (https://plotly.com/careers)
Resources (https://plotly.com/resources/)
Blog (https://medium.com/@plotlygraphs)

Support

Community Support (https://community.plot.ly/)
Documentation (https://plotly.com/graphing-libraries)

Terms of Service (https://community.plotly.com/tos) Privacy Policy (https://plotly.com/privacy/)