plotly

Star 23,446

Dash Python > ***Image Annotations***

Image Annotations with Dash

This tutorial shows how to annotate images with different drawing tools in plotly figures, and how to use such annotations in Dash apps.

Annotation Tools in Plotly Figures

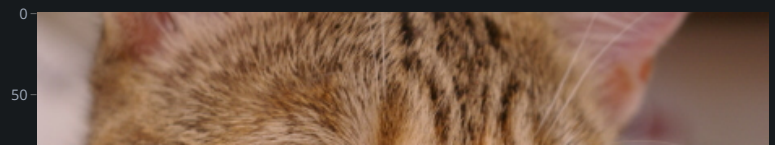
With the plotly graphing library, **it is possible to draw annotations on Cartesian axes**, which are recorded as shape elements of the figure layout.

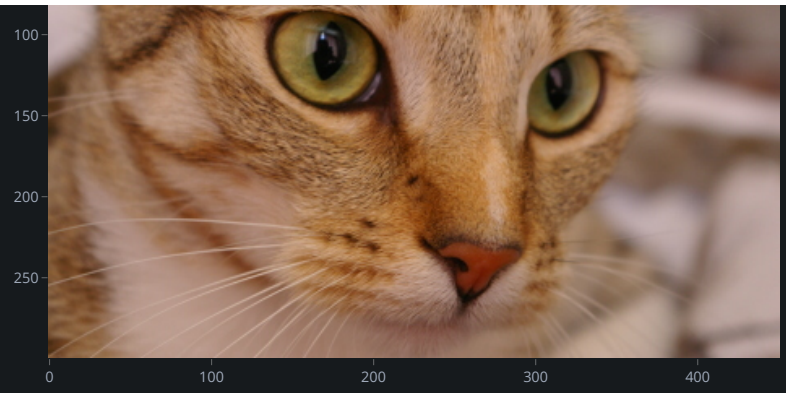
In order to use the drawing tools of a plotly figure, one must set its dragmode to one of the available drawing tools. This can be done programmatically, by setting the `dragmode` attribute of the figure `layout`, or by selecting a drawing tool in the modebar of the figure. Since buttons corresponding to drawing tools are not included in the default modebar, one must specify the buttons to add in the `config` prop of the `dcc.Graph` containing the plotly figure.

In the figure below, you can try to draw a rectangle by left-clicking and dragging, then you can try the other drawing buttons of the modebar.

```
import plotly.express as px
from dash import Dash, dcc, html
from skimage import data

img = data.chelsea()
fig = px.imshow(img)
fig.update_layout(dragmode="drawrect")
config = {
    "modeBarButtonsToAdd": [
        "drawline",
        "drawopenpath",
        "drawclosedpath",
        "drawcircle",
        "drawrect",
        "eraseshape",
    ]
}

app = Dash()
app.layout = html.Div(
    [html.H3("Drag and draw annotations"), dcc.Graph(figure=fig, config=config),]
)

if __name__ == "__main__":
    app.run(debug=True)
```

Drag and draw annotations



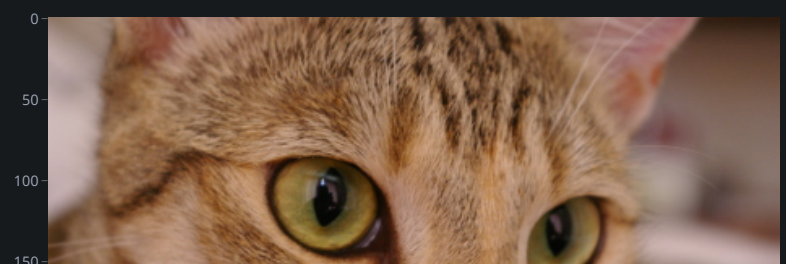


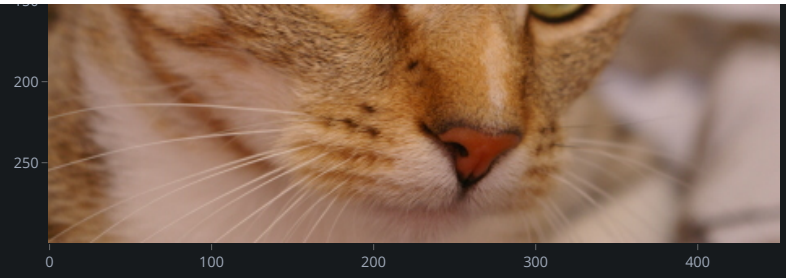
Dash Callback Triggered When Drawing Annotations

When using a plotly figure in a `dcc.Graph` component in a Dash app, drawing a shape on the figure will modify the `relayoutData` property of the `dcc.Graph`. You can therefore define a callback listening to `relayoutData`. In the example below we display the content of `relayoutData` inside an `html.Pre`, so that we can inspect the structure of `relayoutData` (when developing your app, you can also just print the variable inside the callback to inspect it).

```
import plotly.express as px
from dash import Dash, dcc, html, Input, Output, no_update, callback
from skimage import data
import json

img = data.chelsea()
fig = px.imshow(img)
fig.update_layout(dragmode="drawrect")

app = Dash()
app.layout = html.Div(
    [
        html.H3("Drag and draw rectangle annotations"),
        dcc.Graph(id="graph-picture", figure=fig),
        dcc.Markdown("Characteristics of shapes"),
        html.Pre(id="annotations-data"),
    ]
)

@callback(
    Output("annotations-data", "children"),
    Input("graph-picture", "relayoutData"),
    prevent_initial_call=True,
)
def on_new_annotation(relayout_data):
    if "shapes" in relayout_data:
        return json.dumps(relayout_data["shapes"], indent=2)
    else:
        return no_update

if __name__ == "__main__":
    app.run(debug=True)
```

Drag and draw rectangle annotations





Characteristics of shapes

In the example below, we add all the available drawing tools to the modebar, so that you can inspect the characteristics of drawn shapes for the different types of shapes: rectangles, circles, lines, closed and open paths.

Rectangles, circles or ellipses and lines are all defined by their bounding-box rectangle, that is by the coordinates of the start and end corners of the rectangle, `x0`, `y0`, `x1` and `y1`.

As for paths, open and closed, their geometry is defined as an **SVG path**.

```
import plotly.express as px
from dash import Dash, dcc, html, Input, Output, no_update, callback
from skimage import data
import json

img = data.chelsea()
fig = px.imshow(img)
fig.update_layout(dragmode="drawclosedpath")
config = {
    "modeBarButtonsToAdd": [
        "drawline",
        "drawopenpath",
        "drawclosedpath",
        "drawcircle",
        "drawrect",
        "eraseshape",
    ]
}

# Build App
app = Dash()
app.layout = html.Div(
    [
        html.H4(
            "Drag and draw annotations - use the modebar to pick a different drawing tool"
        ),
        dcc.Graph(id="graph-pic", figure=fig, config=config),
        dcc.Markdown("Characteristics of shapes"),
        html.Pre(id="annotations-data-pre"),
    ]
)

@callback(
    Output("annotations-data-pre", "children"),
    Input("graph-pic", "relayoutData"),
    prevent_initial_call=True,
)
def on_new_annotation(relayout_data):
    if "shapes" in relayout_data:
        return json.dumps(relayout_data["shapes"], indent=2)
    else:
        return no_update

if __name__ == "__main__":
    app.run(mode="inline")
```

Drag and draw annotations - use the modebar to pick a different drawing tool

Characteristics of shapes

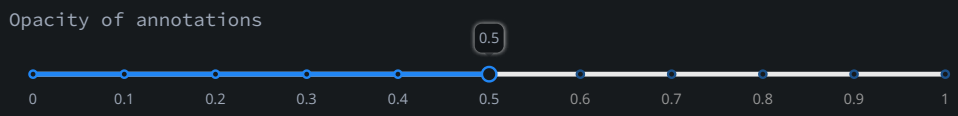
Changing The Style of Annotations

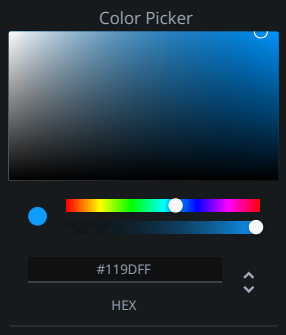
The style of annotations can be changed thanks to interactive components such as sliders, dropdowns, or color pickers. Their values can be used in a callback to define the `newshape` attribute of the figure layout, as in the following example.

```
import plotly.express as px
from dash import Dash, dcc, html, Input, Output, callback
import dash_daq as daq
from skimage import data

img = data.chelsea()
fig = px.imshow(img)
fig.update_layout(
    dragmode="drawrect",
    newshape=dict(fillcolor="cyan", opacity=0.3, line=dict(color="darkblue", width=8)),
)

app = Dash()
app.layout = html.Div(
    [
        html.H3("Drag and draw annotations"),
        dcc.Graph(id="graph-styled-annotations", figure=fig),
        html.Pre('Opacity of annotations'),
        dcc.Slider(id="opacity-slider", min=0, max=1, value=0.5, step=0.1, tooltip={'always_vis
        daq.ColorPicker(
            id="annotation-color-picker", label="Color Picker", value=dict(hex="#119DFF")
        ),
    ]
)

@callback(
    Output("graph-styled-annotations", "figure"),
    Input("opacity-slider", "value"),
    Input("annotation-color-picker", "value"),
    prevent_initial_call=True,
)
def on_style_change(slider_value, color_value):
    fig = px.imshow(img)
    fig.update_layout(
        dragmode="drawrect",
        newshape=dict(opacity=slider_value, fillcolor=color_value["hex"]),
    )
    return fig
```

```
if __name__ == "__main__":
    app.run(debug=True)
```

Drag and draw annotations





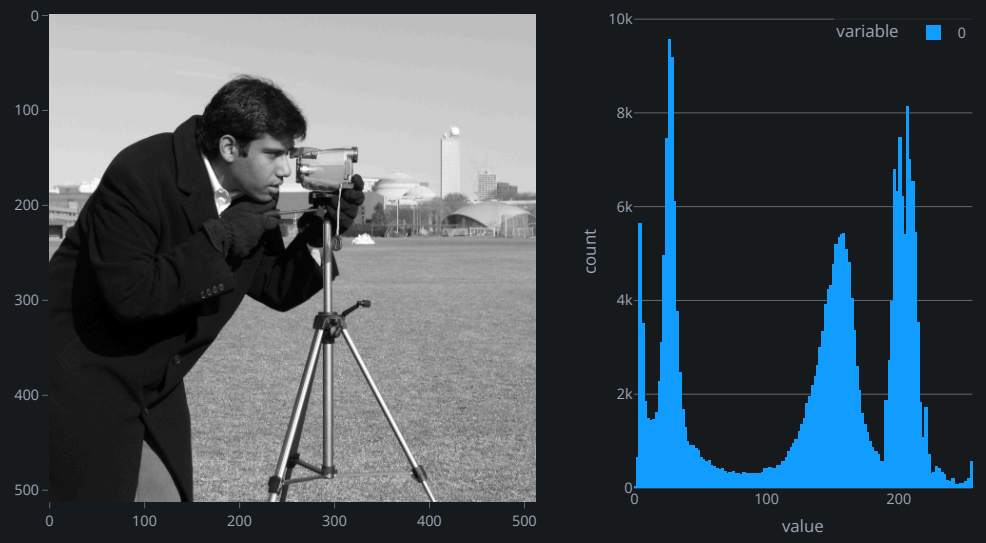
Extracting an Image Subregion Defined By an Annotation

Rather than the geometry of annotations, one is often interested in extracting the region of interest of the image delineated by the shape. The two examples show how to do this first for rectangles, and then for a closed path. In these two examples, the histogram of the region delineated by the latest shape is displayed.

```
import plotly.express as px
from dash import Dash, dcc, html, Input, Output, no_update, callback
from skimage import data

img = data.camera()
fig = px.imshow(img, binary_string=True)
fig.update_layout(dragmode="drawrect")

fig_hist = px.histogram(img.ravel())

# Build App
app = Dash()
app.layout = html.Div(
    [
        html.H3("Drag a rectangle to show the histogram of the ROI"),
        html.Div(
            [dcc.Graph(id="graph-pic-camera", figure=fig),],
            style={"width": "60%", "display": "inline-block", "padding": "0 0"},
        ),
```

```
        html.Div(
            [dcc.Graph(id="histogram", figure=fig_hist),],
            style={"width": "40%", "display": "inline-block", "padding": "0 0"},
        ),
    ]
)

@callback(
    Output("histogram", "figure"),
    Input("graph-pic-camera", "relayoutData"),
    prevent_initial_call=True,
)
def on_new_annotation(relayout_data):
    if "shapes" in relayout_data:
        last_shape = relayout_data["shapes"][-1]
        # shape coordinates are floats, we need to convert to ints for slicing
        x0, y0 = int(last_shape["x0"]), int(last_shape["y0"])
        x1, y1 = int(last_shape["x1"]), int(last_shape["y1"])
        roi_img = img[y0:y1, x0:x1]
        return px.histogram(roi_img.ravel())
    else:
        return no_update

if __name__ == "__main__":
    app.run(debug=True)
```

Drag a rectangle to show the histogram of the ROI



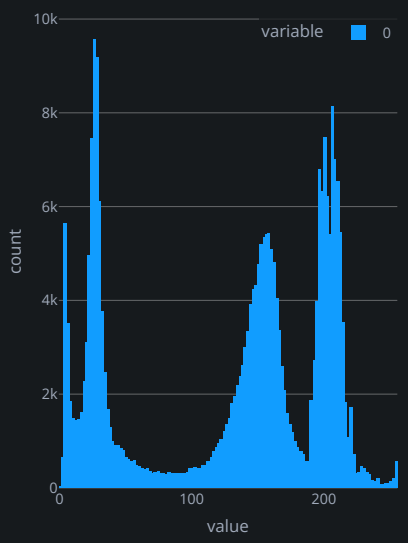
For a path, we need the following steps

- we retrieve the coordinates of the vertices of the path from the SVG path
- we use the function `skimage.draw.polygon` to obtain the coordinates of pixels covered by the path
- then we use the function `scipy.ndimage.binary_fill_holes` in order to set to `True` the pixels enclosed by the path.

```
import numpy as np
import plotly.express as px
from dash import Dash, html, dcc, Input, Output, no_update, callback
from skimage import data, draw
from scipy import ndimage

def path_to_indices(path):
    """From SVG path to numpy array of coordinates, each row being a (row, col) point
    """
    indices_str = [
        el.replace("M", "").replace("Z", "").split(",") for el in path.split("L")
    ]
```

```
    return np.rint(np.array(indices_str, dtype=float)).astype(np.int)

def path_to_mask(path, shape):
    """From SVG path to a boolean array where all pixels enclosed by the path
    are True, and the other pixels are False.
    """
    cols, rows = path_to_indices(path).T
    rr, cc = draw.polygon(rows, cols)
    mask = np.zeros(shape, dtype=np.bool)
    mask[rr, cc] = True
    mask = ndimage.binary_fill_holes(mask)
    return mask

img = data.camera()
fig = px.imshow(img, binary_string=True)
fig.update_layout(dragmode="drawclosedpath")

fig_hist = px.histogram(img.ravel())

app = Dash()
app.layout = html.Div(
    [
        html.H3("Draw a path to show the histogram of the ROI"),
        html.Div(
            [dcc.Graph(id="graph-camera", figure=fig),],
            style={"width": "60%", "display": "inline-block", "padding": "0 0"},
        ),
        html.Div(
            [dcc.Graph(id="graph-histogram", figure=fig_hist),],
            style={"width": "40%", "display": "inline-block", "padding": "0 0"},
        ),
    ]
)

@callback(
    Output("graph-histogram", "figure"),
    Input("graph-camera", "relayoutData"),
    prevent_initial_call=True,
)
```

Draw a path to show the histogram of the ROI



Modifying Shapes and Parsing `relayoutData`

When adding a new shape, the `relayoutData` variable consists in the list of all layout shapes. It is also possible to delete a shape by selecting an existing shape, and by clicking the "delete shape" button in the modebar.

Also, existing shapes can be modified if their `editable` property is set to `True`. In the example below, you can

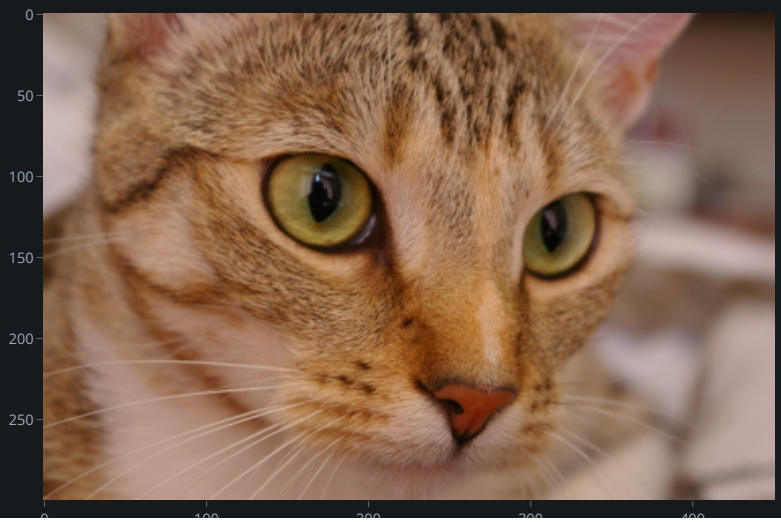
- draw a shape

- then click on the shape perimeter to select the shape
- drag one of its vertices to modify the shape

Observe that when modifying the shape, only the modified geometrical parameters are found in the `relayoutData`.

```
import plotly.express as px
from dash import Dash, dcc, html, Input, Output, no_update, callback
from skimage import data
import json

img = data.chelsea()
fig = px.imshow(img)
fig.update_layout(dragmode="drawclosedpath")
config = {
    "modeBarButtonsToAdd": [
        "drawline",
        "drawopenpath",
        "drawclosedpath",
        "drawcircle",
        "drawrect",
        "eraseshape",
    ]
}

# Build App
app = Dash()
app.layout = html.Div(
    [
        html.H4("Draw a shape, then modify it"),
        dcc.Graph(id="fig-image", figure=fig, config=config),
        dcc.Markdown("Characteristics of shapes"),
        html.Pre(id="annotations-pre"),
    ]
)

@callback(
    Output("annotations-pre", "children"),
    Input("fig-image", "relayoutData"),
    prevent_initial_call=True,
)
def on_new_annotation(relayout_data):
    for key in relayout_data:
        if "shapes" in key:
            return json.dumps(f'{key}: {relayout_data[key]}', indent=2)
    return no_update

if __name__ == "__main__":
    app.run(debug=True)
```

Draw a shape, then modify it

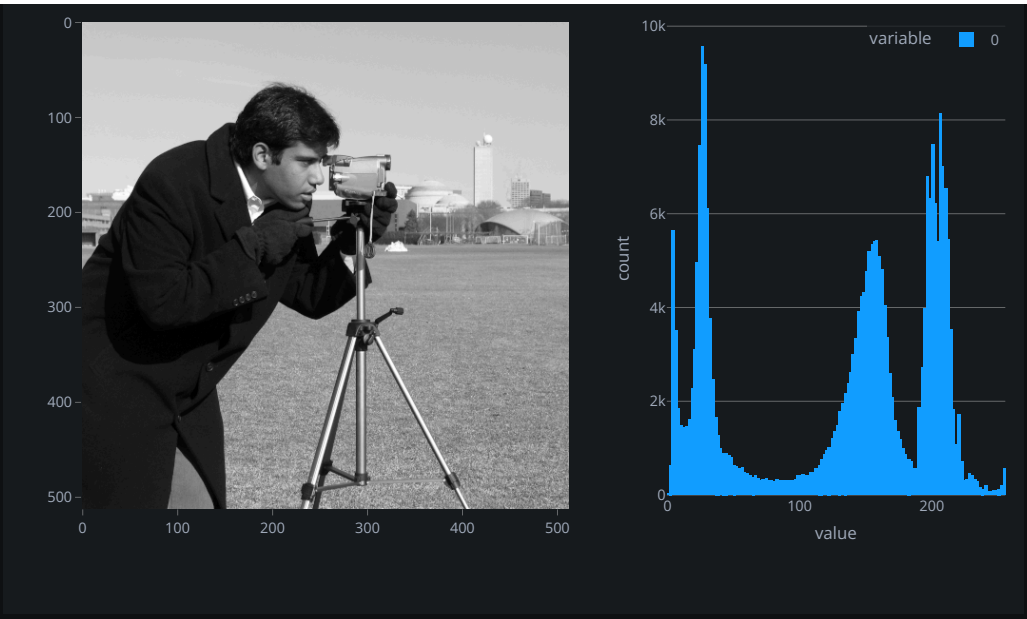


Characteristics of shapes

The example below extends on the previous one where the histogram of a ROI is displayed. Here, we tackle both the case where a new shape is drawn, and where an existing shape is modified.

```
import plotly.express as px
import plotly.graph_objects as go
from dash import Dash, dcc, html, Input, Output, no_update, callback
from skimage import data, exposure
import json

img = data.camera()
fig = px.imshow(img, binary_string=True)
fig.update_layout(dragmode="drawrect")

fig_hist = px.histogram(img.ravel())

# Build App
app = Dash()
app.layout = html.Div(
    [
        html.H3("Draw a shape, then modify it."),
        html.Div(
            [dcc.Graph(id="fig-pic", figure=fig),],
            style={"width": "60%", "display": "inline-block", "padding": "0 0"},
        ),
        html.Div(
            [dcc.Graph(id="graph-hist", figure=fig_hist),],
            style={"width": "40%", "display": "inline-block", "padding": "0 0"},
        ),
        html.Pre(id="annotations"),
    ]
)

@callback(
    Output("graph-hist", "figure"),
    Output("annotations", "children"),
    Input("fig-pic", "relayoutData"),
    prevent_initial_call=True,
)
def on_relayout(relayout_data):
    x0, y0, x1, y1 = (None,) * 4
    if "shapes" in relayout_data:
        last_shape = relayout_data["shapes"][-1]
        x0, y0 = int(last_shape["x0"]), int(last_shape["y0"])
        x1, y1 = int(last_shape["x1"]), int(last_shape["y1"])
        if x0 > x1:
            x0, x1 = x1, x0
        if y0 > y1:
            y0, y1 = y1, y0
    elif any(["shapes" in key for key in relayout_data]):
        x0 = int([relayout_data[key] for key in relayout_data if "x0" in key][0])
        x1 = int([relayout_data[key] for key in relayout_data if "x1" in key][0])
        y0 = int([relayout_data[key] for key in relayout_data if "y0" in key][0])
        y1 = int([relayout_data[key] for key in relayout_data if "y1" in key][0])
    if all((x0, y0, x1, y1)):
        roi_img = img[y0:y1, x0:x1]
```

Draw a shape, then modify it.



Dash Python ***> Image Annotations***