webviz_site_generator Documentation

Release 0.1.0

equinor

Apr 15, 2020

Contents

CHAPTER 1

webviz package

This package contains the core functionality for putting together different *Page* instances into a *Webviz* instance. Each *Page* is a collection of *PageElement* instances, which are rendered in the input order on the corresponding page when running *Webviz.write_html()*.

```
from webviz import Webviz, Page

web = Webviz('Main title')
page = Page("Demo", icon='graph')
web.add(page)
web.write_html("./simple_webviz_example", overwrite=True, display=True)
```

This small example will create an instance `web`, add one empty page to it (in addition to the default index/front page), and write the output to a folder `./simple_webviz_example`.

class webviz.**HeaderElement**

Bases: webviz._header_element.HeaderElement

A HeaderElement describes one action taken to include a header element to a Page.

Parameters

- **tag** – The tag of the header element, such as 'script' or 'link'.
- **attributes** – Dictinary of attributes of the tag, such as *{'src': 'jquery.js'}*. Value strings can include the template value *{root_folder}* which will be substituted with the path to the root of the site.
- **content** – The content of the text (inner html).
- **source_file** – If the header element refers to a file, then the absolute path to that file.
- **target_file** – The relative path to the file, as it is referred to in the attributes.

class webviz.**Webviz**(*title*, *banner_title='Webviz'*, *banner_image=None*, *copyright_notice=None*, *theme='default'*)

Bases: object

An instance of *Webviz* is a collection of *Page* instances, and optionally also *SubMenu* instances. *Webviz* is used to build a collection of these, which can afterwards be rendered as *html*.

There is one special *Page* included as default, `index`, which is the front page in the *html* output.

add(*menu_item*)
Adds an item to the top-level navigation bar of the *Webviz* instance.

Parameters **menu_item** – A *Page* or `Submenu` to add to the *Webviz* instance.

Raises `ValueError`, if `menu_item` is neither *Page* nor a *SubMenu*.

pages
List of all `Pages` in the *Webviz* instance.

write_html(*destination*, *display=False*, *overwrite=False*)
Writes the *html* to the destination folder.

Parameters

- **destination** – Directory to write the *html* output to.
- **overwrite** – *Optional Parameter*. Whether to ignore if the given destination already exists. Content inside the folder may be deleted.
- **display** – *Optional Parameter*. Whether to open browser to the index page.

Raises `ValueError` if `overwrite` is `False` and destination folder exists.

class `webviz.`**Html**(*html*, *js_deps=[]*, *css_deps=[]*)
Bases: `webviz._page_element.PageElement`

A page element for adding *html*.

Parameters

- **html** – The *html* string to add to the page.
- **js_deps** – A list of *js* files (absolute path) to be included in the html code.
- **css_deps** – A list of *css* files (absolute path) to be included in the html code.

get_template()

Returns The corresponding `jinja2` template for the plot, which can be rendered using:

```
html = self.get_template().render(element=self)
```

class `webviz.`**Page**(*title*, *icon=None*)
Bases: `object`

Container for *PageElement* instances. In order to be rendered the *Page* should be added to a *Webviz* instance.

Parameters

- **title** – String. A title for the page.
- **icon** – *Optional parameter*. Name of an icon provided by the *webviz.Theme* used in the *Webviz* instance this page will be added to.

add_content(*content*)
Add a *PageElement* to the page.

Parameters **content** – The *PageElement* to add.

Raises `ValueError` if `content` is not a *PageElement*.

header_elements

Returns The set of *css* dependencies for all page elements in the page

resources

Returns The set of *css* dependencies for all page elements in the page

class webviz.**SubMenu**(*title*, *icon=None*)
Bases: object

A submenu is a collection of pages with its own title and icon. The pages in a submenu are grouped together in the naviagation of the *Webviz* .

Parameters

- **title** – The title of the submenu.
- **icon** – *Optional parameter*. Name of an icon provided by the *webviz.Theme* used in the *Webviz* instance this submenu will be added to.

add_page(*page*)
Adds a *Page* to the submenu.

Parameters **page** – A *Page* to add to the submenu.

Raises ValueError if page is not a *Page*.

location

Returns The location of the first page, or *None* if the submenu is empty.

class webviz.**Markdown**(*md*)
Bases: webviz._page_element.PageElement

A page element for adding *markdown*.

get_template()

Returns The corresponding jinja2 template for the plot, which can be rendered using:

```
html = self.get_template().render(element=self)
```

class webviz.**PageElement**
Bases: object

A page element with data and a template which renders to *html*.

Each element also has a unique containerId in order to make unique DOM IDs in the template.

get_template()

Returns The corresponding jinja2 template for the plot, which can be rendered using:

```
html = self.get_template().render(element=self)
```

class webviz.**Theme**
Bases: webviz._theme.Theme

A theme contains the templates and files related to building Webviz instance.

There is one entry template, main.html, which is rendered for each page.

Webviz exposes a set of jinja2 macros that set up includes the content. A minimal example of a theme is as follows:

```
{% import macros as webviz with context %}
<html>
<head>
{{ webviz.header() }}
</head>
<body>
{% call(banner) webviz.banner() %}
<img src='{{banner}}'></img>
{% endcall %}
<ul>
  {% call(loc, title, current, icon, sub) webviz.iter_menu() %}
    <li> <a href='{{loc}}'> {{icon}} {{title}}</a>
      {% if sub %}
      <ul>
          {% call(sub_loc, sub_title) webviz.iter_sub_menu(sub) %}
              <li><a href='{{sub_loc}}'> {{icon}} {{sub_title}}</a></li>
          {% endcall %}
        </li>
      </ul>
       {% endif %}
  {% endcall %}
</ul>
{{ webviz.content() }}

{% if copyright_notice %}
  {{ copyright_notice }}
{% endif%}
</body>
</html>
```

See the `webviz_default_theme` plugin for a more advanced example.

Parameters

- **`template_loader`** – A loader where the `main.html`, and all the templates it references, can be found.
- **`css_files`** – List of additional *css* files to be included on each page.
- **`js_files`** – List of additional *js* files to be included on each page.
- **`resources`** – Dictionary of additional files to be included by the template. The key is the relative location where this resource should be found. For instance, if `resources['images'] = ['/absolute/path/to/my_image.jpg']`, the image can be included in the template by the resources macro as `webviz.resources('images/my_image.jpg')`.
- **`icons`** – A dictionary of icons provided by the theme.

class webviz.**JSONPageElement**

Bases: `webviz._page_element.PageElement`

A *JSONPageElement* is a *PageElement* which stores some json-data. The data is either assigned to some key in the json store object or otherwise can be accessed as a json-string.

```
>>> json_page_element['my_json_data'] = {'key': 3}
>>> json_page_element['my_json_data']
{'key': 3}
```

(continues on next page)

(continued from previous page)

```
>>> json_page_element.get_json_dump('my_json_data')
'{"key": 3}'
```

The json data can be "dumped", i.e. stored in a key of the json store object.

```
>>> json_page_element.dump_all_jsons('/my/dir')
{'my_json_data': (json_store['123-567-8910'] = {"key": 3};')}
```

Asking for the json dump will then instead return a lookup in the json-store:

```
>>> json_page_element.get_json_dump('my_json_data')
'json_store["123-567-8910"]'
```

get_js_dep is overriden including js files with assignments to the json store.

dump_all_jsons()

Returns A map from all json-keys to assignments to the json_store.

```
>>> json_page_element.dump_all_jsons()
{'my_json_data' : 'json_store["123-567-8910"] = {"data": 3};'}
```

dump_json_key(*key*)

Dumps the given json-key.

Raises KeyError, if there is no value for the given json-key.

get_json_dump(*key*)

Returns Dumped value for the given key. Either lookup in store or a json string.

is_dumped(*key*)

Returns Whether the json-value with the given key has been dumped.

CHAPTER 2

webviz visualizations

class webviz_plotly.**FilteredPlotly**(*data*, *check_box_columns=[]*, *slider_columns=[]*, *dropdown_columns=[]*, **args*, ***kwargs*)

Bases: *webviz_plotly.Plotly*

Page Element for adding filtering controls to Plotly plots that take a dataframe. Values are grouped by labels, for instance:

```
index,value,labels
01-01-2020,3,A
02-01-2020,4,B
```

If 'labels' is chosen as a dropdown_column, then the value 4 will be chosen if the dropdown menu is set to the label B, and the value 3 will be chosen if the dropdown is set to A.

The *FilteredPlotly.process_data()* handles the generation of the plot data. For the example above, it is given the following dataframes:

```
index,value
01-01-2020,3
```

and

```
index,value,
02-01-2020,4
```

Layout and config is then generated that insert the required controls.

Parameters

- **data** – A dataframe, or list of dataframes, that can be processed by *process_data*. Each dataframe will be grouped based on check_box_columns and given as a parameter list to process data. A special label, FilteredPlotly.wildcard ('*' by default), signifies that the data should be present in all groups. If a dataframe does not contain a column it is treated as if all rows have the wildcard label.

- **check_box_columns** – Columns in the dataframes that contain labels to be filtered on by check boxes.
- **slider_columns** – Columns in the dataframe that contain labels to be filtered on by a slider.
- **dropdown_columns** – Columns in the dataframe that contain labels to be filtered on by a dropdown menu.

get_template()
overrides *webviz.PageElement.get_template()*.

names_match(*filters*, *names1*, *names2*)

process_data(**datas*)

Returns List of traces to be used a data for the Plotly Page Element.

wildcard = '*'

class webviz_plotly.**Plotly**(*data*, *layout={}*, *config={}*, ***kwargs*)
Bases: webviz._json_page_element.JSONPageElement

Plotly page element. Arguments are the same as `plotly.plot()` from *plotly.js*. See https://plot.ly/javascript/ for usage.

Parameters

- **xaxis** – Will create a label for the x-axis.
- **yaxis** – Will create a label for the y-axis.
- **logx** – boolean value to toggle x-axis logarithmic scale.
- **logy** – boolean value to toggle y-axis logarithmic scale.
- **xrange** – list of minimum and maximum value. Ex: [3, 15].
- **yrange** – list of minimum and maximum value. Ex: [3, 15].

Note: *Plotly* will not allow the modebarbuttons in *DISALLOWED_BUTTONS*, as these are not useful for the visualizations implemented in webviz.

DISALLOWED_BUTTONS = ['sendDataToCloud', 'resetScale2d']

add_annotation(***kwargs*)

get_template()
Overrides *webviz.PageElement.get_template()*.

handle_args(*title=None*, *xrange=None*, *yrange=None*, *xaxis=None*, *yaxis=None*, *logx=False*, *logy=False*)

class webviz_bar_chart.**BarChart**(*data*, *barmode='group'*, *logy=False*, **args*, ***kwargs*)
Bases: *webviz_plotly.FilteredPlotly*

Bar chart page element.

Parameters

- **data** – Either a file path to a *csv* file or a `pandas.DataFrame`. If a dataframe is given, each column is one set of bars in the chart. The dataframe index is used for the horizontal values. Similarly for the *csv* file, where a special column named `index` will be used for the horizontal values.

- **barmode** – Either `'group'`, `'stack'`, `'relative'` or `'overlay'`. Defines how multiple bars per index-value are combined. See plotly.js layout-barmode.

process_data(*data*)

Returns List of traces to be used a data for the Plotly Page Element.

class webviz_heat_map.**HeatMap**(*data*, **args*, ***kwargs*)

Bases: *webviz_plotly.FilteredPlotly*

Line chart page element.

Parameters **data** – Either a file path to a *csv* file or a `pandas.DataFrame`. Each column of the dataframe becomes one line in the chart. Similarly for the *csv* file, but a special column `index` will be used as the horizontal value.

process_data(*data*)

Returns List of traces to be used a data for the Plotly Page Element.

class webviz_histogram.**Histogram**(*data*, *xlabel=None*, *ylabel='[%]'*, *barmode='overlay'*, *histnorm='percent'*, *nbinsx=0*, *logy=False*, *logx=False*, **args*, ***kwargs*)

Bases: *webviz_plotly.FilteredPlotly*

Histogram page element.

Parameters

- **data** – Either a file path to a *csv* file or a `pandas.DataFrame`. If a dataframe is given, each column is one set of bars in the chart. The dataframe index is used for the horizontal values. Similarly for the *csv* file, where a special column named `index` will be used for the horizontal values.
- **barmode** – Either `'group'`, `'stack'`, `'relative'` or `'overlay'`. Defines how multiple bars per index-value are combined. See plotly.js layout-barmode.
- **histnorm** – Either `''`, `'percent'`, `'probability'`, `'density'` or `'probability density'`. Spesifies type of normalization used. See plotly.js histogram-histnorm.
- **nbinsx** – Maximum number of desired bins. Default value `0` will generate optimal number of bins.

process_data(*data*)

Returns List of traces to be used a data for the Plotly Page Element.

class webviz_line_chart.**LineChart**(*data*, *logy=False*, **args*, ***kwargs*)

Bases: *webviz_plotly.FilteredPlotly*

Line chart page element.

Parameters **data** – Either a file path to a *csv* file or a `pandas.DataFrame`. If a dataframe is given, each column is one line in the chart. The dataframe index is used for the horizontal values. Similarly for the *csv* file, where a special column named `index` will be used for the horizontal values.

process_data(*data*)

Returns List of traces to be used a data for the Plotly Page Element.

class webviz_pie_chart.**PieChart**(*data*, *num_per_row=4*)

Bases: *webviz_plotly.Plotly*

Pie chart page element.

Parameters

- **data** – Value for each sector, or csv file (one column for each sector). Each row (line) becomes a separate pie chart. If there is a column with the name 'pie_chart_label' it is used for the name of each pie chart.
- **num_per_row** – If more than one pie chart, number per row.

class webviz_scatter_plot.**ScatterPlot**(*data*, *logy=False*, **args*, ***kwargs*)

Bases: *webviz_plotly.FilteredPlotly*

Scatter plot page element.

Parameters **data** – Either a file path to a *csv* file or a `pandas.DataFrame`. If a dataframe is given, each column is one set of points in the chart. The dataframe index is used for the horizontal values. Similarly for the *csv* file, where a special column named `index` will be used for the horizontal values.

process_data(*data*)

Returns List of traces to be used a data for the Plotly Page Element.

class webviz_tornado_plot.**TornadoPlot**(**args*, ***kwargs*)

Bases: *webviz_plotly.FilteredPlotly*

Tornado plot page element.

Parameters

- **data** – Either a file path to a *csv* file or a `pandas.DataFrame`. There are two columns: 'low' and 'high' describing.
- **high_text** – Optional text

process_data(*data*)

Returns List of traces to be used a data for the Plotly Page Element.

CHAPTER 3

Examples

The different `webviz` visualization plugins can, when installed, be imported using e.g.

```
from webviz.page_elements import BarChart, LineChart, PieChart, ScatterPlotMatrix
```

3.1 Bar chart

```
from webviz import Webviz, Page
from webviz.page_elements import BarChart
import pandas as pd

web = Webviz('Bar Chart Example')

page = Page('Bar Chart')

bars1 = [10, 15, 13, 17]

bars2 = [16, 5, 11, 9]

bars = pd.DataFrame({'bars1': bars1, 'bars2': bars2})

page.add_content(BarChart(bars))
web.add(page)
web.write_html("./webviz_example", overwrite=True, display=False)
```

3.2 Fan chart

```
from webviz import Webviz, Page
from webviz.page_elements import FanChart
import pandas as pd
```

(continues on next page)

(continued from previous page)

```
web = Webviz('Fan Chart Example')

page = Page('Fan Chart')

index = ['2012-01-01', '2012-01-02', '2012-01-03', '2012-01-04']

name = ['line-1', 'line-1', 'line-1', 'line-1']

mean = [10, 15, 13, 17]

p10 = [11, 16, 13, 18]

p90 = [9, 14, 12, 16]

areaMax = [16, 17, 16, 19]

areaMin = [4, 1, 9, 8]

lines = pd.DataFrame({
  'index': index,
  'name': name,
  'mean': mean,
  'p10': p10,
  'p90': p90,
  'max': areaMax,
  'min': areaMin
})

observations = pd.DataFrame({
  'name': ['line-2', 'line-3'],
  'value': [4, 3],
  'error': 2
})

page.add_content(FanChart(lines, observations))
web.add(page)
web.write_html("./webviz_example", overwrite=True, display=False)
```

3.3 Heat map

```
from webviz import Webviz, Page
from webviz.page_elements import HeatMap
import pandas as pd

web = Webviz('Heat Map Example')

page = Page('Heat Map')

lines = pd.DataFrame(
        [[1, 20, 30, 50, 1], [20, 1, 60, 80, 30], [30, 60, 1, -10, 20]],
        columns=['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'],
        index=['Morning', 'Afternoon', 'Evening']
        )
```

(continues on next page)

(continued from previous page)

```
page.add_content(HeatMap(lines))
web.add(page)
web.write_html("./webviz_example", overwrite=True, display=False)
```

3.4 Histogram

```
from webviz import Webviz, Page
from webviz.page_elements import Histogram
import pandas as pd
import numpy as np

web = Webviz('Histogram Example')

page = Page('Histogram')

normal = [x for x in np.random.normal(size=1000).tolist()]
poisson = [x for x in np.random.poisson(10, 1000).tolist()]
triangular = [x for x in np.random.triangular(0, 10, 20, 1000).tolist()]

data = pd.DataFrame({'normal': normal, 'poisson': poisson,
                     'triangular': triangular})

page.add_content(Histogram(data, nbinsx=20))
web.add(page)
web.write_html("./webviz_example", overwrite=True, display=False)
```

3.5 Line chart

```
from webviz import Webviz, Page
from webviz.page_elements import LineChart
import pandas as pd

web = Webviz('Line Chart Example')

page = Page('Line Chart')

line1 = [10, 15, 13, 17]

line2 = [16, 5, 11, 9]

lines = pd.DataFrame({
    'line 1': line1,
    'line 2': line2,
    'line 3': line2,
    'line 4': line2,
    'line 5': line2,
    'line 6': line2,
    })

page.add_content(LineChart(lines))
```

(continues on next page)

(continued from previous page)

```
web.add(page)
web.write_html("./webviz_example", overwrite=True, display=False)
```

3.6 Pie chart

```
from webviz import Webviz, Page
import pandas as pd
from webviz.page_elements import PieChart

web = Webviz('Pie Chart Example')

page = Page('Pie Chart')

frame = pd.DataFrame(
    [[19, 26, 55], [33, 14, 55]],
    columns=['sector 1', 'sector 2', 'sector 3'])

page.add_content(PieChart(frame))
web.add(page)
web.write_html("./webviz_example", overwrite=True, display=False)
```

3.7 Scatter plot

```
from webviz import Webviz, Page
from webviz_scatter_plot import ScatterPlot
import pandas as pd

web = Webviz('Scatter Plot Example')

page = Page('Scatter Plot')

index = ['2012-01-01', '2012-01-02', '2012-01-03', '2012-01-04']

point1 = [10, 15, 13, 17]

point2 = [16, 5, 11, 9]

points = pd.DataFrame({
    'index': index,
    'points 1': point1,
    'points 2': point2
})

points.set_index('index', inplace=True)

page.add_content(ScatterPlot(points))
web.add(page)
web.write_html("./webviz_example", overwrite=True, display=False)
```

3.8 Scatter plot matrix

```
from webviz import Webviz, Page
from webviz.page_elements import ScatterPlotMatrix
import pandas as pd

web = Webviz('Scatter Plot Matrix Example')

page = Page('Scatter Plot Matrix')

point1 = [10.6, 15, 13.4, 17]
point2 = [16, 5, 11, 9.7]
point3 = [51, 25.6, 51, 23]
point4 = [19, 75.1, 23, 49]
name = ['name1', 'name1', 'name2', 'name2']

points = pd.DataFrame({
    'point1': point1,
    'point2': point2,
    'point3': point3,
    'point4': point4,
    'name': name
})

page.add_content(ScatterPlotMatrix(points))
web.add(page)
web.write_html("./webviz_example", overwrite=True, display=False)
```

3.9 Tornado plot

```
from webviz import Webviz, Page
from webviz.page_elements import TornadoPlot
import pandas as pd

web = Webviz('Tornado Plot Example')

page = Page('Tornado Plot')

high = [0.8, 1, 0.3, 0.4]

low = [0.5, -0.7, -.5, -0.1]

index = ['A', 'B', 'C', 'D']

bars = pd.DataFrame(
    {'low': low, 'high': high},
    index=index
)

plot = TornadoPlot(bars)
plot.add_annotation(
        x=low[1],
        y=index[1],
        ay=0,
```

(continues on next page)

(continued from previous page)

```
        ax=-20,
        text='label')
page.add_content(plot)
web.add(page)
web.write_html("./webviz_example", overwrite=True, display=False)
```

CHAPTER 4

Introduction

Welcome! You are now browsing the documentation for `webviz` - a static site generator, optionally including different kind of interactive visualizations. `webviz` facilitates automatic visualization using the popular open source libraries d3.js and plotly.js.

`webviz` creates *html* output such that the report can be viewed through a web browser. The site generator can be used in two different ways: using `yaml` and markdown, or the `webviz` Python API.

4.1 Using folder structure and markdown files

Webviz can be executed using

```
python -m webviz site_folder
```

where `site_folder` is a folder containing markdown files. See the github repository for an example. In the `site_folder`, there are two special files: `index.md` and `config.yaml`. `index.md` is the landing page for the site and `config.yaml` contains configuration info, such as which theme to use.

In markdown files, page elements (such as visualizations) can be added using:

```
{{ page_element(
     name,
     *args,
     *kwargs
 )}
```

`name` *string*: name of page element. Page elements are the following: `Html`, `FilteredPlotly`, `Plotly`, `LineChart`, `BarChart`, `PieChart`, `TornadoPlot`, `FanChart`, `ScatterPlotMatrix`, `Map`, `Histogram`, `ScatterPlot`, `HeatMap`

`*args` *args*: args of page elements method

`**kwargs` *kwargs*: kwargs of page elements method

4.2 API example

The example below creates several (currently empty) pages, linked together through a navigation menu. Further below you will see examples on how to add content to the different pages.

```
from webviz import Webviz, Page, SubMenu, Markdown

web = Webviz('Main title', theme='minimal')

ex1 = Page('Example 1')
ex2 = Page('Example 2')
ex3 = Page('Markdown example')

some_content = (r"""

# Markdown support
***

> __You can pass markdown wihin a triple-quotes__<br>
> _Also known as multiline comments_

|First Header  | Second Header  | Third Header |
|:-------------|:-------------: | ------------:|
|Content Cell  | `Content Cell` | Content      |
|Content Cell  | Content Cell   | Content      |

---

    #!python
    def hello():
        print('Hello World')
---
If you want to use math formulas, you have several different options. You can
use double dollar signs:

```
$$ \left(\frac{\sqrt x}{y^3}\right) $$
```

Result: $$ \left(\frac{\sqrt x}{y^3}\right) $$

Or you can wrap it between special commands like this:

```
\begin{equation}
\cos (2\theta) = \cos^2 \theta - \sin^2 \theta \label{my_cos_equation}.
\end{equation}
```

Result:
\begin{equation}
\cos (2\theta) = \cos^2 \theta - \sin^2 \theta \label{my_cos_equation}.
\end{equation}

All equations with labels can easily be referred to in the text as
```\eqref{my_cos_equation}```, resulting in something like
\eqref{my_cos_equation}.
```

(continues on next page)
```

(continued from previous page)

```
If you want an equation without numbering add "notag":

```
\begin{equation}
\lim_{x \to \infty} \exp(-x) = 0.\notag
\end{equation}
```

Result:
\begin{equation}
\lim_{x \to \infty} \exp(-x) = 0.\notag
\end{equation}

If you want to write multi-line equations aligned on e.g. the equal sign, you
can also do that:

```
\begin{align}
f(x) &= (x+a)(x+b) \\\\
     &= x^2 + (a+b)x + ab
\end{align}
```

Result:
\begin{align}
f(x) &= (x+a)(x+b) \\\\
     &= x^2 + (a+b)x + ab
\end{align}

The & operator indicates what to align on. You can also write in-line equations
or symbols inbetween, like ```\(\alpha \)``` (\(\alpha \)) and
```\(\gamma \)``` (\(\gamma \)).

To prevent build failing because of backslashes, use a rawstring format by
adding `r` in front of the string.

You can read more about the input format
[here](http://docs.mathjax.org/en/latest/tex.html#).

Example:

`formula = Markdown(r'$$x_{1,2} = \frac{-b \pm \sqrt{b^2-4ac}}{2b}.$$')`

Renders out to this:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2-4ac}}{2b}.$$
""")

ex3.add_content(Markdown(some_content))

submenu1 = SubMenu('Menu 1')
submenu2 = SubMenu('Menu 2')
submenu3 = SubMenu('Menu 3')
```

(continues on next page)
```

(continued from previous page)

```
submenu1.add_page(ex1)
submenu2.add_page(ex2)
submenu3.add_page(ex3)

web.add(submenu1)
web.add(submenu2)
web.add(submenu3)

web.write_html("./webviz_example", overwrite=True, display=False)
```

When the site is created by running *webviz.Webviz.write_html()*, the output is a folder containing all the files needed for opening and running the site in a browser.

For information about how to use the webviz Python API, see the *webviz package*.

Python Module Index

W

Index

R

S

T

W