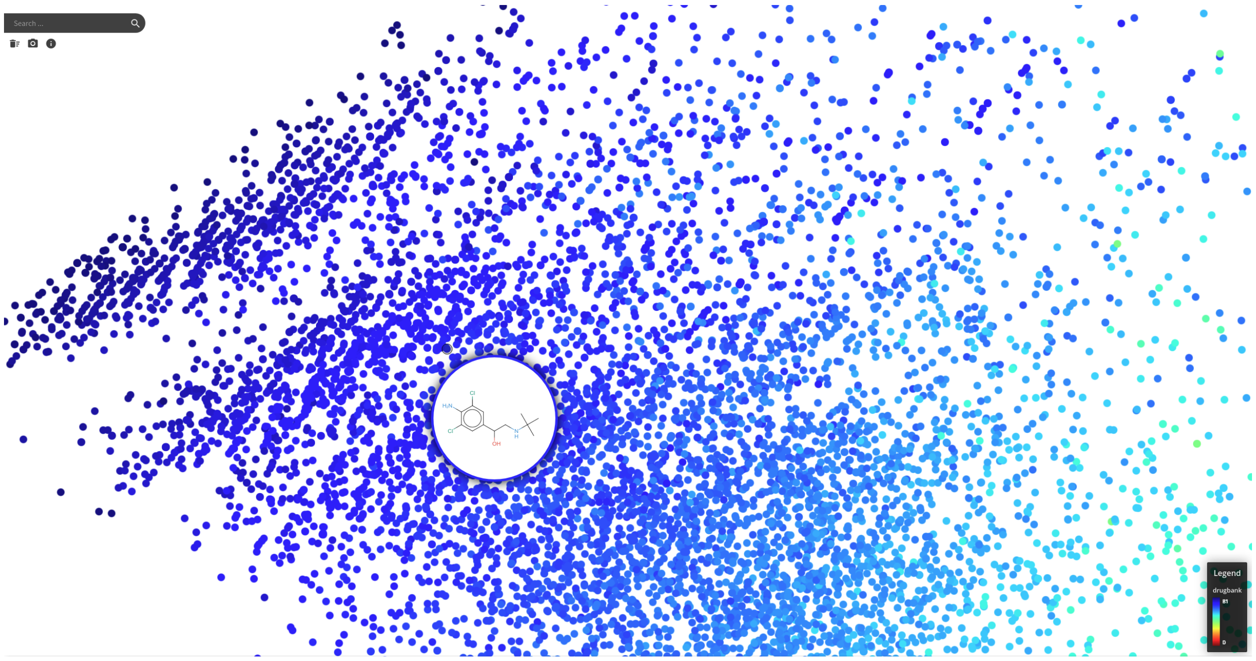
Faerun

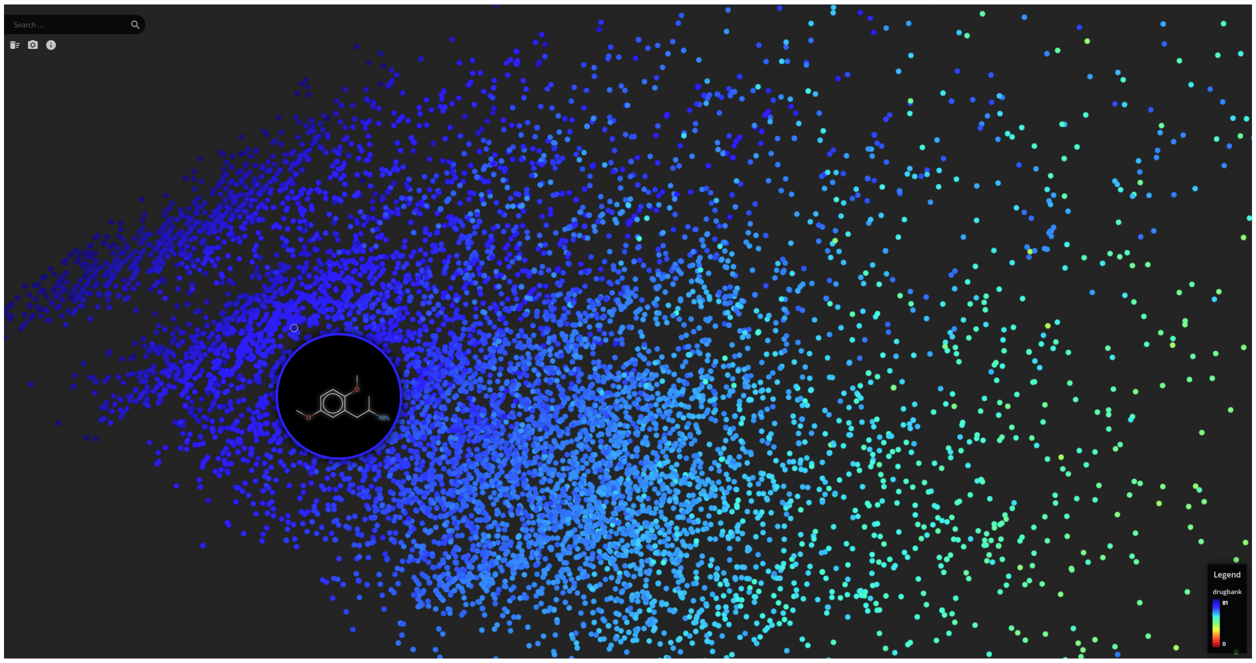
Release 1.0

Jun 21, 2019

CONTENTS

Here are some teasers.





CHAPTER

ONE

TABLE OF CONTENTS

1.1 Getting Started

1.1.1 Installation

Faerun can be installed using pip.

```
pip install faerun
```

In order to use it in a script, the class `Faerun` has to be imported from the package.

```python
from faerun import Faerun
```

That's it for the installation.

1.1.2 Create a Plot Document

In order to plot, a plot document has to be created. To do so, create an instance of the class `Faerun`

```python
from faerun import Faerun

f = Faerun(title='faerun-example', clear_color='#222222', coords=False, view='free')
```

Here, we set the `title` of the plot document. This will be used as the title of the HTML document. The clear color of the canvas, which is the background color of the plot, is set to `'#222222'` (which is the hex-code for a dark gray). The drawing of coordinate axes is disabled by setting `coords=False` and since we want to draw 3D data, the argument `view` is set to `'free'` to enable the user to pan and rotate the plot.

1.1.3 Preparing Data for Plotting

The next step is to prepare the data which is to be plotted. In the tutorial, we will just generate some nice looking data using `numpy`.

```python
import numpy as np

x = np.linspace(0, 12.0, 326)
y = np.sin(np.pi * x)
z = np.cos(np.pi * x)
c = np.random.randint(0, 6, len(x))
```

This data can then be wrapped in a `dict`. In addition, `DataFrame` from the `pandas` package are also supported by faerun. In the example, the same values are used for the colors `c` and labels `labels`.

```
data = {'x': x, 'y': y, 'z': z, 'c': c, 'labels': c}
```

1.1.4 Adding a Scatter Layer

Given the `Faerun` instance and the data, a scatter plot can be created using the method `add_scatter`.

```
f.add_scatter('helix', data, shader='sphere', colormap='Dark2', point_scale=5.0,
              categorical=True, has_legend=True, legend_labels=[(0, 'Zero'), (1, 'One
↪')])
```

The data is added as a scatter layer named helix. The chose shader will render the data points as spheres (with diffuse and specular lighting) of size 5.0 with colors from the `matplotlib` colormap `'Dark2'`. As the `c` is categorical, the parameter `categorical` is set to `True`, otherwise `matplotlib` will mess up the values.

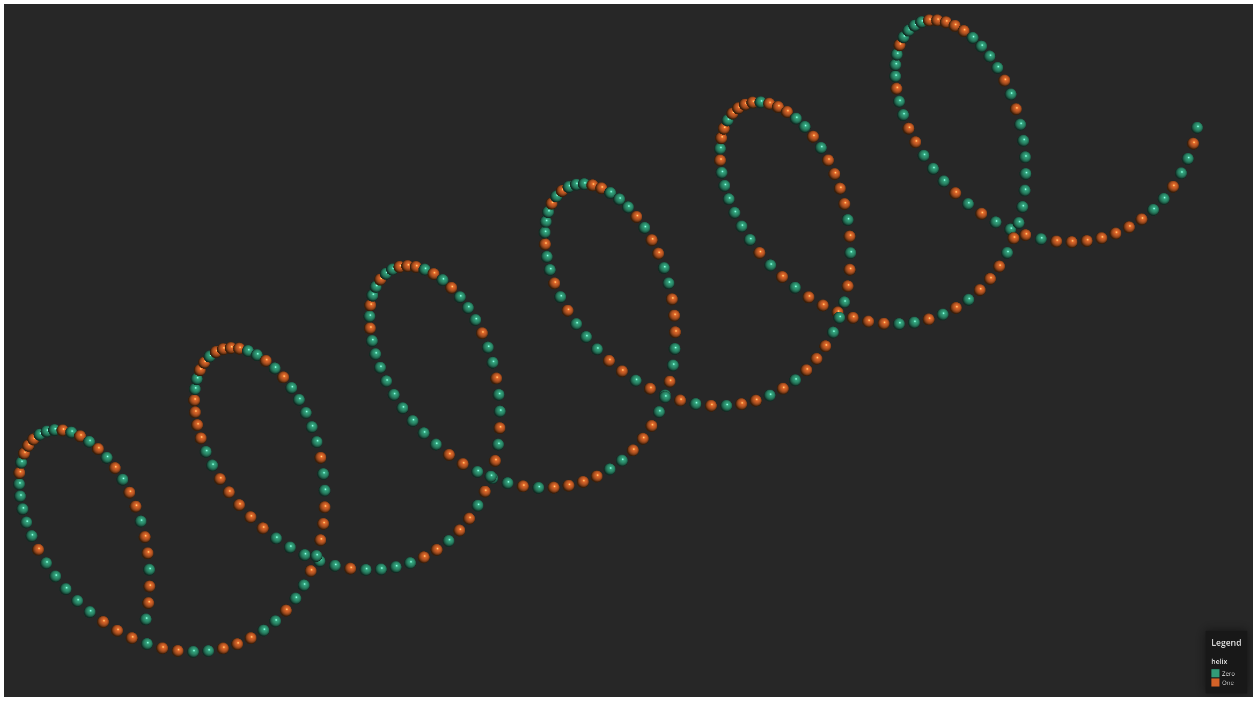
Finally, `has_legend=True` adds the scatter layer to the legend and `legend_labels` is a `list` of `tuple`, associating values with a label.

1.1.5 Saving to HTML

The faerun document can the be plotted to an HTML document with an accompanying JavaScript data file.

```
f.plot('helix')
```

This saves the plot as `helix.html` and ``helix.js`. The files can be opened locally or hosted on any web server.



1.1.6 Saving to Faerun Data File

The faerun document can also be exported to a faerun data file, which in turn can then be hosted using the `web` module.

```
import pickle
```

(continues on next page)

(continued from previous page)

```python
with open('helix.faerun', 'wb+') as handle:
    pickle.dump(f.create_python_data(), handle, protocol=pickle.HIGHEST_PROTOCOL)
```

1.1.7 Complete Example

```python
import pickle
import numpy as np
from faerun import Faerun

def main():
    f = Faerun(title='faerun-example', clear_color='#222222', coords=False, view='free
↪')

    x = np.linspace(0, 12.0, 326)
    y = np.sin(np.pi * x)
    z = np.cos(np.pi * x)
    c = np.random.randint(0, 2, len(x))

    data = {'x': x, 'y': y, 'z': z, 'c': c, 'labels': c}

    f.add_scatter('helix', data, shader='sphere', colormap='Dark2', point_scale=5.0,
                  categorical=True, has_legend=True, legend_labels=[(0, 'Zero'), (1,
↪'One')])

    f.plot('helix')

    with open('helix.faerun', 'wb+') as handle:
        pickle.dump(f.create_python_data(), handle, protocol=pickle.HIGHEST_PROTOCOL)

if __name__ == '__main__':
    main()
```

1.2 Web

1.2.1 Introduction

While small plots can easily be created and stored in a single HTML and JavaScript file, which are then loaded completely in the browsers memory, this is not possible for larger data sets due to browser limitations. In order to solve this problem, Faerun includes a small HTTP server (based on cherrypy) to serve the data to the browser.

1.2.2 Creating Faerun Data Files

As shown in *Getting Started*, Faerun can save data as `.faerun` data files using `pickle`.

```python
with open('helix.faerun', 'wb+') as handle:
    pickle.dump(f.create_python_data(), handle, protocol=pickle.HIGHEST_PROTOCOL)
```

1.2.3 Starting a Faerun Web Server

```python
from faerun import host

host('helix.faerun', label_type='default',
     theme='dark')
```

1.2.4 Formatting Labels

Labels can be formatted by defining a custom `label_formatter`. If no `label_formatter` is provided to the `host` function, the default is used:

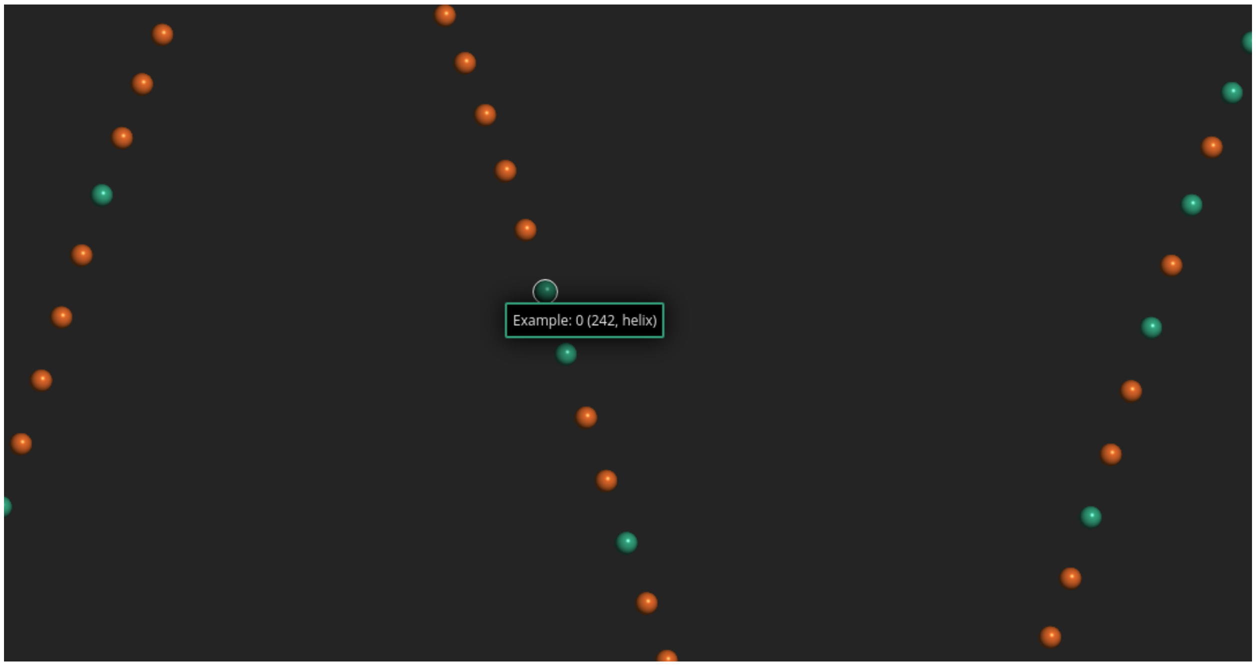
```
label_formatter = lambda label, index, name: label.split('__')[0]
```

This default splits the label value on `'__'` to store different labels and enable search on different values the the displayed labels. See "Searching" for details. Defining a custom label formatter is straight forward. As an example, let's prefix each label with a string and add their index and layer name:

```
def custom_label_formatter(label, index, name):
    return f'Example: {label} ({index}, {name})'

host('helix.faerun', label_type='default',
     theme='dark', label_formatter=custom_label_formatter)
```

This function is then called whenever a label is requested from the server. In addition to the argument `label`, the arguments `index` and `name` can be used to further customize the displayed label and represent the integer index of the data point and the data layer they belong to (e.g. the name defined with `add_scatter`).



1.2.5 Adding Hyperlinks

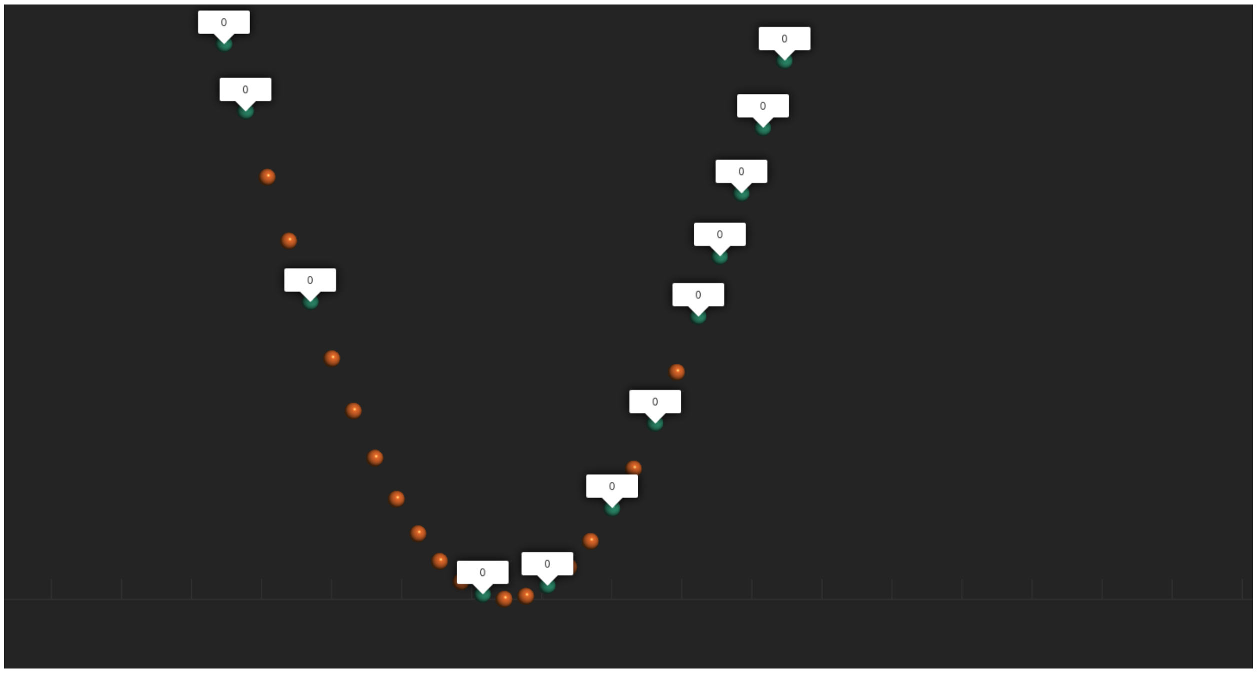
Faerun allows to link the data to an arbitrary URL which can be visited upon double-clicking a data point. In order to do this, a `link_formatter` has to be provided. This works similar to customizing the label.

```
def custom_link_formatter(label, index, name):
    return f'https://www.google.com/search?q={label}'

host('helix.faerun', label_type='default',
     theme='dark', link_formatter=custom_link_formatter)
```

1.2.6 Searching

The hosted version of a Faerun visualization also allows for searching. As a default, the search searches for exact matches in labels (substring or regex searches are not possible at this time).



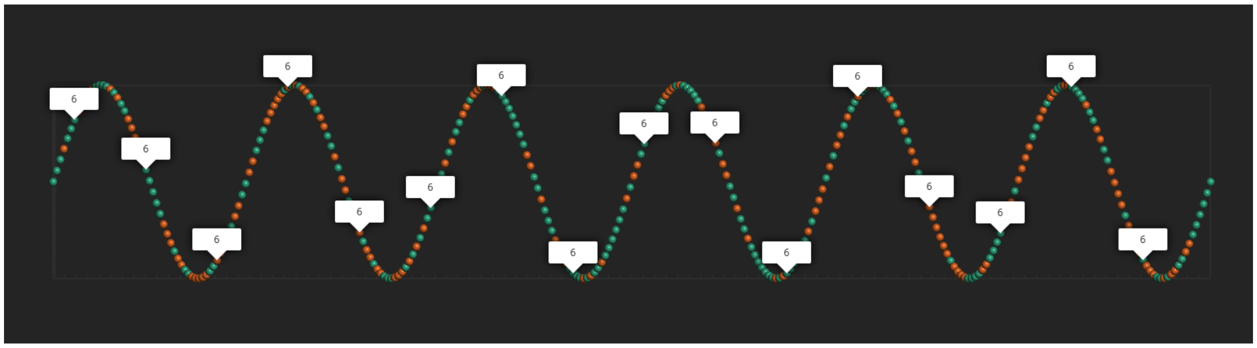
However, the search can be customized. As described in "Formatting Labels", additional label values can be added by separating them using '__'.

```
c = np.random.randint(0, 2, len(x))
labels = [''] * len(c)

for i, e in enumerate(c):
    labels[i] = str(e) + '__' + str(i % 20)

data = {'x': x, 'y': y, 'z': z, 'c': c, 'labels': labels}
```

The above examples adds an additional label value and as default, the second label value is then used by the search.



If there are additional label values, the search index can be set using the `search_index` argument.

1.2.7 Add Info / Documentation

As the visualization is ready to be deployed to a publicly accessible web server, it might be of interest to add a documentation. The `host` method supports the argument `info` that accepts a (markdown formatted) string. This information is the desplayed on the generated web page.

```
info = ('#Welcome to Fearun',
        'This is a small Faerun example.'
        '',
        'Yay markdown! This means that you can easily:',
        '- Add lists',
        '- Build tables',
        '- Insert images and links',
        '- Add code examples',
        '- ...'
       )

host('helix.faerun', label_type='default', theme='dark',
    label_formatter=custom_label_formatter, link_formatter=custom_link_formatter,
    info='\n'.join(info))
```

An info button is then shown next to the screenshot button, which upon click opens a window containing the info.

1.2.8 Complete Example

```
import pickle
import numpy as np
from faerun import Faerun, host


def main():
    f = Faerun(title='faerun-example', clear_color='#222222', coords=False, view='free
↪')

    x = np.linspace(0, 12.0, 326)
    y = np.sin(np.pi * x)
    z = np.cos(np.pi * x)
    c = np.random.randint(0, 2, len(x))

    labels = [''] * len(c)

    for i, e in enumerate(c):
        labels[i] = str(e) + '__' + str(i % 20)

    data = {'x': x, 'y': y, 'z': z, 'c': c, 'labels': labels}

    f.add_scatter('helix', data, shader='sphere', colormap='Dark2', point_scale=5.0,
                categorical=True, has_legend=True, legend_labels=[(0, 'Zero'), (1,
↪'One')])

    f.plot('helix')

    with open('helix.faerun', 'wb+') as handle:
        pickle.dump(f.create_python_data(), handle, protocol=pickle.HIGHEST_PROTOCOL)

    def custom_label_formatter(label, index, name):
        return f'Example: {label} ({index}, {name})'
```

(continues on next page)

(continued from previous page)

```
    def custom_link_formatter(label, index, name):
        return f'https://www.google.com/search?q={label}'

    info = ('#Welcome to Fearun',
            'This is a small Faerun example.'
            '',
            'Yay markdown! This means that you can easily:',
            '- Add lists',
            '- Build tables',
            '- Insert images and links',
            '- Add code examples',
            '- ...'
        )

    host('helix.faerun', label_type='default', theme='dark',
        label_formatter=custom_label_formatter, link_formatter=custom_link_formatter,
        info='\n'.join(info))


if __name__ == '__main__':
    main()
```

1.3 Documentation

1.3.1 Faerun

class faerun.**Faerun**(*title: str = 'python-faerun'*, *clear_color: str = '#111111'*, *coords: bool = True*, *coords_color: str = '#888888'*, *coords_box: bool = False*, *view: str = 'free'*, *scale: float = 750.0*)

Creates a faerun object which is an empty plotting surface where layers such as scatter plots can be added.

Constructor for Faerun.

Keyword Arguments

- **title** (str, optional) – The title of the generated HTML file
- **clear_color** (str, optional) – The background color of the plot
- **coords** (bool, optional) – Show the coordinate axes in the plot
- **coords_color** (str, optional) – The color of the coordinate axes
- **coords_box** (bool, optional) – Show a box around the coordinate axes
- **view** (str, optional) – The view (front, back, top, bottom, left, right, free)
- **scale** (float, optional) – To what size to scale the coordinates (which are normalized)

__init__(*title: str = 'python-faerun'*, *clear_color: str = '#111111'*, *coords: bool = True*, *coords_color: str = '#888888'*, *coords_box: bool = False*, *view: str = 'free'*, *scale: float = 750.0*)

Constructor for Faerun.

Keyword Arguments

- **title** (str, optional) – The title of the generated HTML file
- **clear_color** (str, optional) – The background color of the plot
- **coords** (bool, optional) – Show the coordinate axes in the plot

- **coords_color** (`str`, optional) – The color of the coordinate axes
- **coords_box** (`bool`, optional) – Show a box around the coordinate axes
- **view** (`str`, optional) – The view (front, back, top, bottom, left, right, free)
- **scale** (`float`, optional) – To what size to scale the coordinates (which are normalized)

add_scatter(*name: str, data: Union[dict, pandas.core.frame.DataFrame], mapping: dict = {'c': 'c', 'cs': 'cs', 'labels': 'labels', 's': 's', 'x': 'x', 'y': 'y', 'z': 'z'}, colormap: Union[str, matplotlib.colors.Colormap] = 'plasma', shader: str = 'sphere', point_scale: float = 1.0, max_point_size: float = 100.0, fog_intensity: float = 0.0, saturation_limit: float = 0.2, categorical: bool = False, interactive: bool = True, has_legend: bool = False, legend_title: str = None, legend_labels: dict = None*)

Add a scatter layer to the plot.

Parameters

- **name** (`str`) – The name of the layer
- **data** (`dict` or `DataFrame`) – A Python dict or Pandas DataFrame containing the data

Keyword Arguments

- **mapping** (`dict`, optional) – The keys which contain the data in the input dict or the column names in the pandas `DataFrame`
- **colormap** (`str` or `Colormap`, optional) – The name of the colormap (can also be a matplotlib Colormap object)
- **shader** (`str`, optional) – The name of the shader to use for the data point visualization
- **point_scale** (`float`, optional) – The relative size of the data points
- **max_point_size** (`int`, optional) – – The maximum size of the data points when zooming in
- **fog_intensity** (`float`, optional) – – The intensity of the distance fog
- **saturation_limit** (`float`, optional) – – The minimum saturation to avoid "gray soup"
- **categorical** (`bool`, optional) – – Whether this scatter layer is categorical
- **interactive** (`bool`, optional) – – Whether this scatter layer is interactive
- **has_legend** (`bool`, optional) – – Whether or not to draw a legend
- **legend_title** (`str`, optional) – – The title of the legend
- **legend_labels** (`dict`, optional) – – A dict mapping values to legend labels

add_tree(*name: str, data: Union[dict, pandas.core.frame.DataFrame], mapping: dict = {'c': 'c', 'from': 'from', 'to': 'to', 'x': 'x', 'y': 'y', 'z': 'z'}, color: str = '#666666', colormap: Union[str, matplotlib.colors.Colormap] = 'plasma', fog_intensity: float = 0.0, point_helper: str = None*)

Add a tree layer to the plot.

Parameters

- **name** (`str`) – The name of the layer
- **data** (`dict` or `DataFrame`) – A Python dict or Pandas DataFrame containing the data

Keyword Arguments

- **`mapping`** (`dict`, optional) – The keys which contain the data in the input dict or DataFrame
- **`color`** (`str`, optional) – The default color of the tree
- **`colormap`** (`str` or `Colormap`, optional) – The name of the colormap (can also be a matplotlib Colormap object)
- **`fog_intensity`** (`float`, optional) – The intensity of the distance fog
- **`point_helper`** (`str`, optional) – The name of the scatter layer to associate with this tree layer (the source of the coordinates)

`create_data``()` → str

Returns a JavaScript string defining a JavaScript object containing the data.

Returns JavaScript code defining an object containing the data

Return type `str`

`create_python_data``()` → dict

Returns a Python dict containing the data

Returns The data defined in this Faerun instance

Return type `dict`

`static discrete_cmap`(*n_colors: int*, *base_cmap: str*) → matplotlib.colors.Colormap

Create an N-bin discrete colormap from the specified input map.

Parameters **`n_colors`** (`int`) – The number of discrete colors to generate

Keyword Arguments **`base_cmap`** (`str`) – The colormap on which to base the discrete map

Returns The discrete colormap

Return type `Colormap`

`get_min_max``()` → tuple

Get the minimum an maximum coordinates from this plotter instance

Returns The minimum and maximum coordinates

Return type `tuple`

`plot`(*file_name: str = 'index'*, *path: str = './'*, *template: str = 'default'*, *legend_title: str = 'Legend'*, *legend_orientation: str = 'vertical'*)

Plots the data to an HTML / JS file.

Keyword Arguments

- **`file_name`** (`str`, optional) – The name of the HTML / JS file
- **`path`** (`str`, optional) – The path to which to write the HTML / JS file
- **`template`** (`str`, optional) – The name of the template to use
- **`legend_title`** (`str`, optional) – The legend title
- **`legend_orientation`** (`str`, optional) – The orientation of the legend ('vertical' or 'horizontal')

1.3.2 Web

`faerun.host` (*path: str, label_type: str = 'smiles', theme: str = 'light', label_formatter: Callable[[str, int, str], str] = None, link_formatter: Callable[[str, int, str], str] = None, info: str = None, legend: bool = False, legend_title: str = 'Legend', view: str = 'front', search_index: int = 1*)

Start a cherrypy server hosting a Faerun visualization.

Parameters **`path`** (`str`) – The path to the fearun data file

Keyword Arguments

- **`label_type`** (`str`) – The type of the labels
- **`theme`** (`str`) – The theme to use in the front-end
- **`label_formatter`** (`Callable[[str, int, str], str]`) – A function used for formatting labels
- **`link_formatter`** (`Callable[[str, int, str], str]`) – A function used for formatting links
- **`info`** (`str`) – A string containing markdown content that is shown as an info in the visualization
- **`legend`** (`bool`) – Whether or not to show the legend
- **`legend_title`** (`str`) – The title of the legend
- **`view`** (`str`) – The view type ('front', 'back', 'top', 'bottom', 'right', 'left', or 'free')
- **`search_index`** (`int`) – The index in the label values that is used for searching

INDEX