

---

# pltEditorTool Usage Guide

## Calling the Plot Editor Tool

The Plot Editor Tool can be installed by running `pip install pltEditorTool`. After installation, using the tool requires only a single import. The code snippet below is the minimum required to call the tool in order to create a single plot.

---

```
# Import the plot editor
from pltEditorTool import plotEditor
# Define the data
x_data = [[1,2,3,4,5]]
y_data = [[10,9,8,7,6]]
label_data = ['test1']
# Call the editor, it will open in a separate window
plotEditor(x=x_data, y=y_data, labels=label_data)
```

---

While the code above is the minimum required for plotting the full list of inputs are as follows:

**x:** The x-data is a required input and must be provided for every plot.

**y:** The y-data is a required input and must be provided for every plot.

**x\_err:** The x-error data can be used to plot errorbars relative to the x-axis of the plot.

**y\_err:** The The y-error data can be used to plot errorbars relative to the y-axis of the plot.

**fill:** The fill data can be used to plot a symmetrical fill region around the y-value of the data, in other words, the filled region will cover the area of  $y+fill$  to  $y-fill$  at each x position.

**fill\_alt:** The fill-alternate data can be used to plot an asymmetrical fill region around the y-value of the data, in other words, the filled region will cover the area of  $y+fill$  to  $y-fill\_alt$  at each x position.

**labels:** The label data is required. Unique labels are required for each set of data or else data sets will be overwritten.

From version 2.0.7 of the tool there are now several ways to send the data to the tool. These are:

**Original Method (List input):** In this method the data is provided to the tool as lists, with each plot being one entry in each of the input lists. As such, if only one plot requires one of the inputs, empty arrays for that input must be provided for the other plots. For example, the following code snippet shows how to add three plots with only the third plot having fill data. This manner of forming the data was to allow for vectors of different lengths to be added to the plot editor without needing to worry about any length compatibility issues. Error checking is done on the inputs and if one of the inputs has an incorrect length a message will be shown that will give

the index of the plot that has an issue. The following code snippet shows how using the list type of input would be achieved. While this example uses numpy arrays, it would be possible to use Python Lists of data as well.

---

```
data = pd.read_excel("test_plot_data.xlsx", header=0)
x_data = [np.array(data.loc[:, 'x']), np.array(data.loc[:, 'x.1'])]
y_data = [np.array(data.loc[:, 'y']), np.array(data.loc[:, 'y.1'])]
x_err_data = [np.array(data.loc[:, 'x_err']), np.array(data.loc[:, 'x_err.1'])]
y_err_data = [np.array(data.loc[:, 'y_err']), np.array(data.loc[:, 'y_err.1'])]
fill_data = [np.array(data.loc[:, 'fill']), np.array(data.loc[:, 'fill.1'])]
fill_alt_data = [np.array(data.loc[:, 'fill_alt']), np.array(data.loc[:, 'fill_alt.1'])]
labels_data = ['Experimental', 'Computation']
print("Case 1: Ordinary List Style Input")
plotEditor(x=x_data, y=y_data, x_err=x_err_data, y_err=y_err_data,
           fill=fill_data, fill_alt=fill_alt_data, labels=labels_data)
```

---

**Numpy Arrays:** There are three ways to provide data with numpy arrays.

1. **Single vectors:** A single numpy vector can be provided for each of the inputs. These arrays must be 1-D Numpy arrays of the same length for each input. This can be used if the intent is to only plot one data-set.
2. **2-D Array:** For multiple data-sets, the inputs can be specified as 2-D Numpy arrays with the samples as columns of data.
3. **3-D Array:** An 3-D Numpy array can be used as the input for multiple data-sets. In this case the indices of the array ( $[i, j, 6]$ ) are;  $i$  is the number of samples, and  $j$  is the number of data points in each sample.

The following code snippet shows how the inputs for each of the Numpy Array input setups would need to be created.

---

```
data = pd.read_excel("test_plot_data.xlsx", header=0)
x_data = np.array(data.loc[:, 'x.1'])
y_data = np.array(data.loc[:, 'y.1'])
x_err_data = np.array(data.loc[:, 'x_err.1'])
y_err_data = np.array(data.loc[:, 'y_err.1'])
fill_data = np.array(data.loc[:, 'fill.1'])
fill_alt_data = np.array(data.loc[:, 'fill_alt.1'])
labels_data = 'Computation'

print("Case 2: Single Numpy Array Input")
plotEditor(x=x_data, y=y_data, x_err=x_err_data, y_err=y_err_data,
           fill=fill_data, fill_alt=fill_alt_data, labels=labels_data)

x_data = np.array([np.array(data.loc[:, 'x']), np.array(data.loc[:, 'x.1'])]).transpose()
y_data = np.array([np.array(data.loc[:, 'y']), np.array(data.loc[:, 'y.1'])]).transpose()
x_err_data = np.array([np.array(data.loc[:, 'x_err']),
                       np.array(data.loc[:, 'x_err.1'])]).transpose()
y_err_data = np.array([np.array(data.loc[:, 'y_err']),
                       np.array(data.loc[:, 'y_err.1'])]).transpose()
fill_data = np.array([np.array(data.loc[:, 'fill']),
                       np.array(data.loc[:, 'fill.1'])]).transpose()
fill_alt_data = np.array([np.array(data.loc[:, 'fill_alt']),
                           np.array(data.loc[:, 'fill_alt.1'])]).transpose()
labels_data = ['Experimental', 'Computation']

print("Case 3: 2D Numpy Array Input")
```

---

```
plotEditor(x=x_data, y=y_data, x_err=x_err_data, y_err=y_err_data,
          fill=fill_data, fill_alt=fill_alt_data, labels=labels_data)
```

```
input_data = np.zeros((2,101,6))
input_data[0,:,0] = x_data[:,0]
input_data[1,:,0] = x_data[:,1]
input_data[0,:,1] = y_data[:,0]
input_data[1,:,1] = y_data[:,1]
input_data[0,:,2] = x_err_data[:,0]
input_data[1,:,2] = x_err_data[:,1]
input_data[0,:,3] = y_err_data[:,0]
input_data[1,:,3] = y_err_data[:,1]
input_data[0,:,4] = fill_data[:,0]
input_data[1,:,4] = fill_data[:,1]
input_data[0,:,5] = fill_alt_data[:,0]
input_data[1,:,5] = fill_alt_data[:,1]
```

```
print("Case 4: 3D Numpy Array Input")
plotEditor(x=input_data, labels=labels_data)
```

---

**Pandas Dataframe:** A pandas dataframe can be used to input the data, in which case the column labels on the data are used to separate the data into the correct vectors in the Tool. Column labels must be the following ('x', 'y', 'x\_err', 'y\_err', 'fill', 'fill\_alt'). the data-set label can be added after these labels with a period (':') as the separator, for example 'x.experimental data'. Only the data-set label from the 'x' column is used when importing the data, so please ensure that the data is in the correct order in the DataFrame.

The following code snippet shows how to import a Pandas DataFrame into the Tool.

---

```
data = pd.read_excel("test_plot_data.xlsx", header=0)
labels_data = ['Experimental', 'Computation']
print("Case 5: Pandas DataFrame Input")
plotEditor(x=data, labels=labels_data)
```

---

The Excel file used in these examples is in the main Github Repository and can be downloaded for testing of the module.



Figure 1: Plot Editor Tool Window

Running any of these codes on Windows or Linux operating systems will bring up the window

shown in Figure 1. Since changing the UI API to the PyQt5 interface system, the window should look the same on MacOS devices as well.

## Components of the Plot Editor Window

The plot editor window consists of two main sections, the plot details section and the axis details section.

### Plot Details:

**Plot Selection** The plot selection frame is shown in Figure 2. The menus in this frame are used to select which plot is to be edited in the Plot Details section.

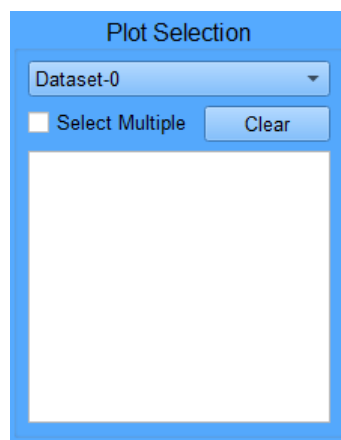


Figure 2: Plot Selection

- The drop-down menu at the very top of the frame is used to select the plot that is to be edited. This menu will contain a list with all the labels that were input in the `labels` list when calling the tool.
- The check-box is used in order to select multiple plots at the same time. This is used if it is necessary to have multiple plots with identical settings. Activating the check-box will allow selection of multiple plots from the drop-down menu. Deselecting the check-box will update all the selected plots with the parameters displayed in the Plot Details section. The only exception to this is that all the plots will retain their individual label data.
- The Clear Selection button can be used to remove plots from the selection pane should a plot be added incorrectly. Multiple plots can be selected at once by clicking on each in order. Using this button will remove the plot from the list without updating any parameters.

**Plot Labels** The plot label entry boxes are shown in Figure 2. These two entry boxes are used to modify the legend entries for the line and fill plots.

- Data Label: This entry box will change the label of the line part of the plot in the legend. Leaving this entry blank will remove the entry from the legend.
- Fill Label: This entry box changes the label of the fill region for the plot in the legend. Again, leaving the entry blank will remove the entry from the legend.
- Note: Scatter plots do not add either of these titles to the legend, and therefore scatter plots will not have legend entries.

Figure 3: Plot labels

**Errorbar Settings** If error bar data is provided for any of the plots submitted to the tool, the error bar settings frame (Figure 4) can be used to modify the error bars.

Figure 4: Errorbar Settings

- The checkbox at the top of the frame allows for easy toggling of errorbars in the plot. If error bar data is provided and the box is checked the error bars will be shown. If the box is not checked, the error bars will not be shown.
- Clicking the color option button will provide a color selection window where the color of the error bars can be specified. This can be done by selecting the color from either preset values, a color map, or by providing RGB or hexadecimal color parameters.
- Selecting the checkbox next to the color button will ensure that any changes to the color of the errorbars is propagated to the colors of all the other elements of that particular plot (line, marker and fill). This simplifies the operation when needing to have all elements of the plot with matching colors.
- The width option adjusts the width of the error bar.
- The cap size option adjusts the width of the error bar cap. This is the length of the perpendicular bar at either end of error bar.
- The cap thickness adjusts the thickness of the cap in the direction of the error bar.

**Line Settings** Since the x and y values are required for every plot, the line settings will always be active when starting the editor tool. These settings can be adjusted using the line settings frame (Figure 5).

- The line checkbox is automatically ticked when creating the tool. To disable the plotting of the line uncheck the box at the top of the frame.
- The color of the line can be adjusted using the color button. This button will display the same color selection window as other color buttons.
- Selecting the checkbox next to the color button will ensure that any changes to the color are propagated to the colors of all the other elements of that particular plot.

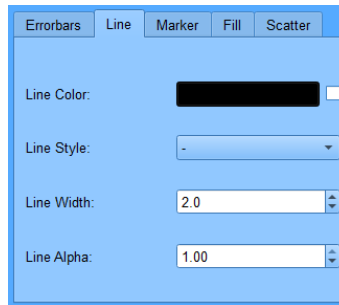


Figure 5: Line Settings

- The line style determines the style of the line. Currently there are four styles, solid line, dashed, dash and dot, and dots. Selecting the blank style has the same effect as unchecking the line checkbox.
- The width of the line can be modified using the width selection.
- The alpha value determines the transparency of the line. A value of 1 is opaque while a value of 0 is completely transparent.

**Marker Settings** The marker settings checkbox is also automatically activated for all plots when the tool is started. The other settings in the marker frame (Figure 6) can be used to adjust the appearance of the markers. Certain marker settings also affect the appearance of errorbars and or the scatter plot markers.

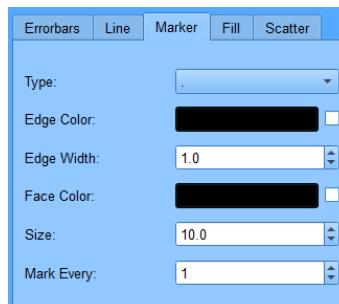


Figure 6: Marker Settings

- The marker checkbox can be used to easily toggle the display of markers for the plot.
- The marker type can be chosen from a large selection of possible marker types.
- The edge color determines the color of the border of the marker. Use the button to display the color selection window.
- The edge width determines how thick the edge of the marker is. Set this to zero to remove the edge on the marker.
- The face color determines the color of the face of the marker. Use the button to display the color selection window.
- Selecting the checkbox next to the either of the color buttons will ensure that any changes to the color are propagated to the colors of all the other elements of that particular plot

- The marker size can be adjusted using the size option. This option will also affect the size of the markers if the scatter plot option is selected.
- A choice to leave spaces between the points that are marked can be made using the "Mark Every" option. Leaving this value at 1 will create a marker for every data point. This value will also affect how frequently error bars are plotted.

**Fill settings** For plots that have fill data provided, the fill frame (Figure 7) can be used to modify the parameters of the fill.

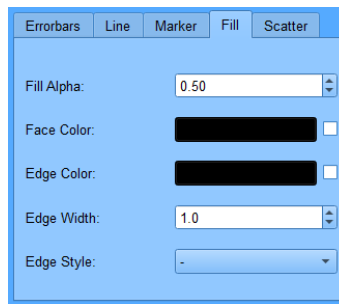


Figure 7: Fill Settings

- The fill checkbox can be used to toggle the fill plot.
- The alpha value determines the transparency of the face of the fill area.
- The face color button can be used to change the color of the filled region.
- The edge color button can be used to change the color of the line plotted on the boundary of the filled region.
- Selecting the checkbox next to the either color button will ensure that any changes to the color are propagated to the colors of all the other elements of that particular plot
- The edge width option varies the thickness of the line plotted on the boundary of the filled region.
- The edge style changes the style of the line plotted on the boundary of the fill region.

**Scatter Plot Settings** Selecting the scatter plot checkbox (Figure 8) will deactivate the error bar, line, marker and fill checkboxes, and will change the plot to a scatter plot. The scatter plot can have both the color and size determined by a vector of values. All possible vectors for doing this are automatically added when the tool is created.

- As already mentioned, the scatter checkbox will deactivate all other plot functions when selected. Selecting any other plot functions after that will deactivate the scatter plot selection.
- The marker options allow for changing the marker type as well as a limited selection of marker edge colors.
- The alpha value determines the transparency of the marker face color.
- The color vector can be chosen from the drop down list. As already mentioned, all possible vectors (i.e. all vectors with the same length as the x-vector of the current plot) have been added to this list.

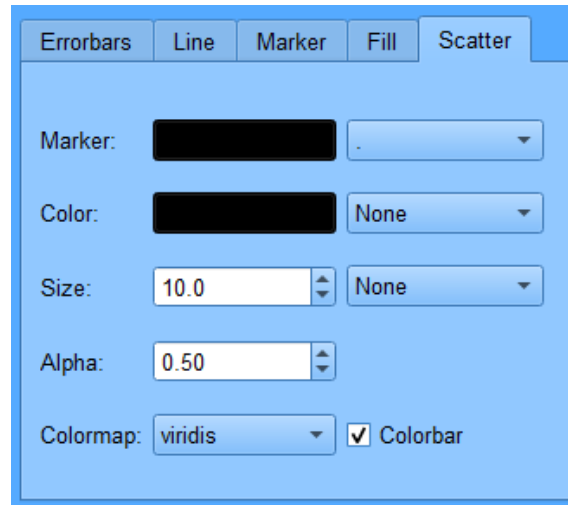


Figure 8: Scatter Plot Settings

- The size vector can be chosen from the size drop down list. Again, all vectors are automatically added. The size of the markers in the scatter plot can be manipulated by changing the marker size in the marker frame (Figure 6), this is true even when a size vector has been selected.
- The color bar check box toggles the plotting of a color bar.
- The color map of the scatter plot can be selected from a wide selection of possible color maps.

### Axis Details:

The final figure is created using the subplot and grid functions of the Matplotlib package. In terms of terminology, each axis in the tool is a single subplot in the figure. As such, each of the plots defined in the previous section can be added to the axes and these axes can be arranged using the grid functions.

The first section of the axis details determines the total size of the grid.

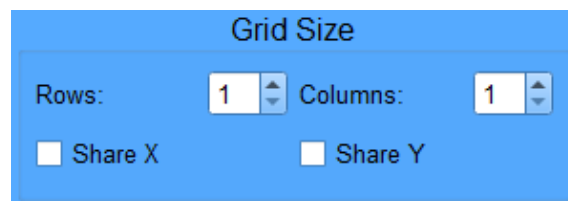


Figure 9: Plot Grid Size Settings

- The number of rows and columns in the grid can be changed using the respective options. Care should be taken since all grid spots do need to be filled in the final figure.
- The share x and share y options will determine whether the axes share x and or y values. Selecting these options requires the subplots to be arranged in a square grid with all subplots having equal length and height. The values of the left column and bottom row are used for the x and y axis limits.



**Creating and Deleting Axes** The first axis is automatically created when the tool is set up and this axis can never be deleted.

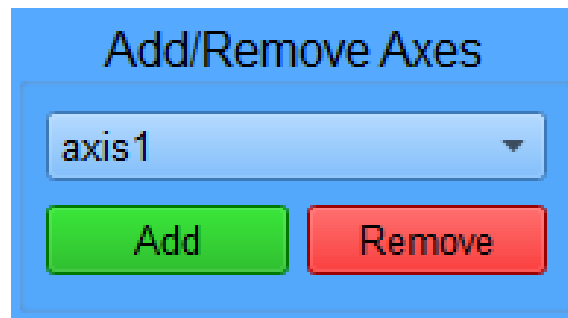


Figure 10: Axis Creation and Deletion

- The add axis button adds a new axis to the list.
- The delete axis button deletes the currently selected axis from the list.

**Data Selection** This section is used to add plot data to the axes. And changes here affect the currently selected axis in the Create or Delete Axes section.

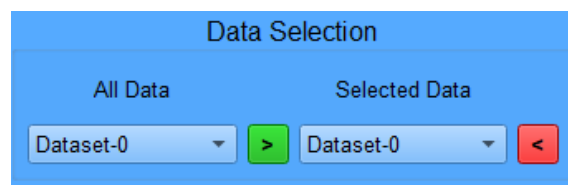
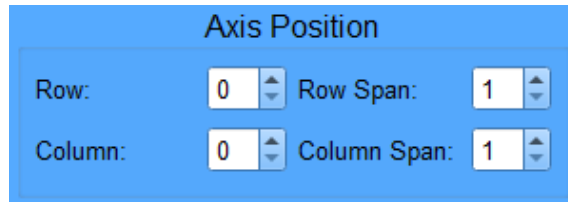


Figure 11: Axis Data Selection

- The left drop down menu contains all the plot labels
- Pressing the green arrow will add the currently selected plot to the axis as long as it does not exist on that axis already.
- The right drop down menu contains the list of currently selected plots for the current axis.
- Clicking the red arrow will remove the currently displayed plot in the Selected Data drop down menu from the axis.

**Axis Position** Axes can be give a position that consists of the row and column position as well as how many rows and columns the plot spans.

- Row and column positions are determined by the Row and Column selections. The positions are zero indexed, so Row 0 is the first row, and column 0 is the first column.
- The row and column span options can be used to make the subplot span more than one row or column in the grid specified in the first section of the Axis Options.

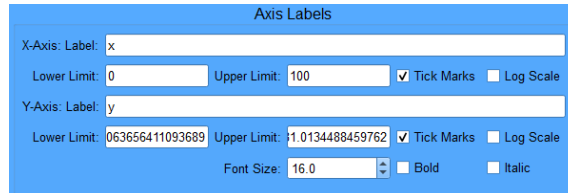


Axis Position

Row: 0 Row Span: 1

Column: 0 Column Span: 1

Figure 12: Axis Position Settings



Axis Labels

X-Axis: Label: x Lower Limit: 0 Upper Limit: 100 ☒ Tick Marks ☐ Log Scale

Y-Axis: Label: y Lower Limit: 063656411093689 Upper Limit: 1.0134488459762 ☒ Tick Marks ☐ Log Scale

Font Size: 16.0 ☒ Bold ☐ Italic

Figure 13: Axis Label Settings

**Axis Labels** This sections allows for manipulation of the axis labels as well as the axis limits in these labels it is possible to use latex notation (e.g. " $\sigma$ ") to obtain special symbols. Deleting all text will remove the label from the plot.

- The first two lines control the x-axis label, the x-axis limits, and whether the x-axis ticks are shown. The final option changes the x-axis to log-scale.
- The second two lines provide the same features for the y-axis.
- The final line adjusts the font size of the labels and whether the label text is bold or italic. To ensure that the ticks also increase when the label size is increased, the tick font size is automatically set 3 points below the label font size.

**Plot Title** Each subplot has its own title and this can be modified in the Plot Title Frame (Figure 14). As with the axis labels, leaving the title field blank will remove the title from the subplot.



Axis Title

Label: Plot

Font Size: 16.0 ☒ Bold ☐ Italic

Figure 14: Plot Title Settings

- The label field sets the value of the plot title. Using " $\n$ " in the label will create multiple lines in the title.
- The font size, and whether the text is bold (automatically selected) and italic can be modified.

**Legend Settings** All the plot labels will be included in the legend the Legend frame Figure 15 can be used to modify the text size and position of the legend.

- The position drop down provides all the standard options for positioning of the legend.
- The font size of the legend can be modified using the Font Size option.



Figure 15: Legend Settings

**Show and Save Plot** The tool provides two methods for viewing the plot (Figure 16). The first is to use the show plot, and the second is to save the plot.



Figure 16: Show and Save Plot Buttons

- The show plot option will display the plot in a separate window. On each subsequent click the plot is deleted and redrawn. This figure uses 150 dpi.
- The save plot option will trigger a save dialog box where the location and filename can be set. The tool will save three files. First is the figure at 600 dpi, the second is a ".npy" file with a dictionary of all the data that was used to generate the figure, and a python code file with the code that can regenerate the plot. This allows for further manipulation of the data if required.

## Advanced Editing of Saved Plot Code

The code that is saved when the Save Plot button is clicked has several functions that use a dictionary to regenerate the plots. This code can be manipulated to either change the plot properties, or alternatively to plot a new data set with the same properties as have been used for the existing plot. To aid in this process this section presents the layout of the dictionary that is saved in the Numpy data file saved with the plot. The following assumes the user already has a working knowledge of Python Dictionary Objects.

First, the last few lines of the saved code are as follows:

```
if __name__ == '__main__':
    data_dict = np.load('plot1_plot_data.npy',allow_pickle='TRUE').item()
    print(data_dict.keys())
    plot_obj = plot_class(data_dict, 'plot1.png')
    plot_obj.show_plot(False)
```

A few notes before diving into the layout of the dictionary. The first line imports the dictionary from the '\*.npy' file listed. Secondly, the second to last line specifies the name of the file as '\*.png', this can be modified to prevent overwriting the original plot image. Finally, the code is automatically set up to only display the plot. To save the plot change the 'False' in the last line to 'True'.

---

To continue then, 'data\_dict' is a Python Dictionary. The entries of this dictionary are shown in the code below:

---

```
data_dict = {'axes': Python List,
            'fig_size': Python List,
            'gsc': Integer,
            'gsr': Integer,
            'sharex': Integer,
            'sharey': Integer,
            'axis_data': Dictionary (axis_dict)}
```

---

- 'axes' is a Python List of axis names.
- 'fig\_size' is a two element list with the height and width of the figure in inches.
- 'gsc' and 'gsr' are the number of columns and rows in the figure grid.
- 'sharex' and 'sharey' are integer boolean values (0 or 1) that indicate whether the axes share x or y values.
- 'axis data' is the axis data dictionary that is detailed in the next section.

---

```
Axis_dict = {'axis1':{'legend': String,
                    'legendFontSize': Float,
                    'position': Python List,
                    'title': String,
                    'title_text': {'Size': Float, 'Bold': Integer, 'Italic': Integer},
                    'x_label': String,
                    'x_lim': Python List,
                    'xscale': Integer,
                    'xticks': Integer,
                    'y_label':String,
                    'y_lim': Python List,
                    'yscale': Integer,
                    'yticks': Integer,
                    'axis_text': {'Size': Float, 'Bold': Integer, 'Italic': Integer},
                    'plots': Python List,
                    'plots_data': Python List}}
```

---

The items in the axis\_dict are fairly self explanatory, however some of the items that might cause confusion are explained belowed.

- 'axis1' There will be an entry for each of the axes in the 'axes' list of data\_dict.
- 'position' is the position of the axis in the figure grid, and has 4 entries, row start, column start, row span, and column span.
- 'x\_lim' and 'y\_lim' are 2 entrie lists with the lower and upper limits of the x and y axes.
- 'xscale' and 'yscale' are integer values with 0 as linear and 1 as log scale.
- 'xticks' and 'yticks' are integer values where 0 removes the ticks and 1 shows them.
- 'plots' is a list of plot names used on this axis.
- the 'plots\_data' entry is a list of dictionaries that correspond to the labels in the plot names list.

For each of the plot\_data dictionaries, the layout is shown below.

---

```
Plot_dict = {'colorbar': Integer,  
            'dif_bot': Numpy Array,  
            'dif_top': Numpy Array,  
            'ebar': Dictionary,  
            'fill': Dictionary,  
            'fill-label': String,  
            'label': String,  
            'line': Dictionary,  
            'marker': Dictionary,  
            'scatter': Dictionary,  
            'x': Numpy Array,  
            'x_err': Numpy Array,  
            'y': Numpy Array,  
            'y_err': Numpy Array}
```

---

- 'colorbar' must be 1 to show the colorbar on a scatter plot
- 'ebar', 'fill', 'line', 'marker', 'scatter' are the dictionaries that define the properties of the various plot options.
- 'x' is the x data
- 'y' is the y data
- 'x\_err' is the half length of the x error bars
- 'y\_err' is the half length of the y error bars
- 'dif\_bot' is the distance between the y value and the bottom of the fill region
- 'dif\_top' is the distance between the y value and the top of the fill region
- 'label', and 'fill-label' are the legend entries for the line and fill region

The dictionaries for the properties of the plot options are shown here for completeness, however no discussion of them will be provided.

---

```
Ebar_dict = {'exist': Integer,  
            'capthick': Integer,  
            'color': String,  
            'capsize': Integer,  
            'linewidth': Integer}
```

```
Fill_dict = {'exist': Integer,  
            'alpha': Float,  
            'edge_col': String,  
            'face_col': String,  
            'line_sty': String,  
            'line_wid': Integer}
```

```
Line_dict = {'exist': Integer,  
            'alpha': Float,  
            'color': String,  
            'style': String,  
            'width': Integer}
```

```
Marker_dict = {'exist': Integer,  
              'edge_col': String,  
              'edge_wid': Integer,  
              'face_col': String,
```

---

```
        'markevery': Integer,
        'size': Integer,
        'type': String}

Scatter_dict = {'exist': Integer,
                'alpha': Float,
                'cmap': String,
                'color_vector_names': Python List,
                'color_vectors': Python List,
                'current_color': String,
                'current_size': String,
                'edge': String,
                'size_vector_names': Python List,
                'size_vectors': Python List,
                'type': String}
```

---

## Final Notes

The code attempts to catch as many errors as possible, however, most of the remaining errors will not result in the plot editor tool crashing.

To report any problems or suggest improvements please submit an issue on the project's Github page:

<https://github.com/RichardCouperthwaite/plt-editor-tool>