TrendVis: an Elegant Interface for dense, sparkline-like, quantitative visualizations of multiple series using matplotlib

Mellissa Cross[‡][*]

https://www.youtube.com/watch?v=tklAFsce7eg

Abstract—TrendVis is a plotting package that uses matplotlib to create information-dense, sparkline-like, quantitative visualizations of multiple disparate data sets in a common plot area against a common variable. This plot type is particularly well-suited for time-series data. We discuss the rationale behind and the challenges associated with adapting matplotlib to this particular plot style, the TrendVis API and architecture, and various features available for users to customize and enhance the readability of their figures while walking through a sample workflow.

Index Terms—time series visualization, matplotlib, plotting

Introduction

Data visualization and presentation is a key part of scientific communication, and many disciplines depend on the visualization of multiple time-series or other series datasets. The field of paleoclimatology (the study of past climate and climate change), for example, relies heavily on plots of multiple time-series or "depth series", where data are plotted against depth in an ice core or stalagmite, for example. These plots are critical to place new data in regional and global contexts and they facilitate interpretations of the nature, timing, and drivers of climate change. Figure 1, created using TrendVis, compares stalagmite records of climate and hydrological changes that occurred during the last two deglaciations, or "terminations". Ice core records of carbon dioxide (black) and methane (pink) [Petit] concentrations and Northern Hemisphere summer insolation (the amount of solar energy received on an area, gray) are also included.

Creating such plots can be difficult, however. Many scientists depend on expensive software such as SigmaPlot and Adobe Illustrator. With pure matplotlib [matplotlib], users have two options: display data in a grid of separate subplots or overlaid using twinned axes. This works for two or three traces, but does not scale well. The ideal style in cases with larger datsets is the style shown in Figure 1: a densely-plotted figure that facilitates direct comparison of curve features. The key aim of TrendVis, available on GitHub, is to enable the creation and readability of

* Corresponding author: cros0324@umn.edu, mellissa.cross@gmail.com
‡ Department of Earth Sciences, University of Minnesota

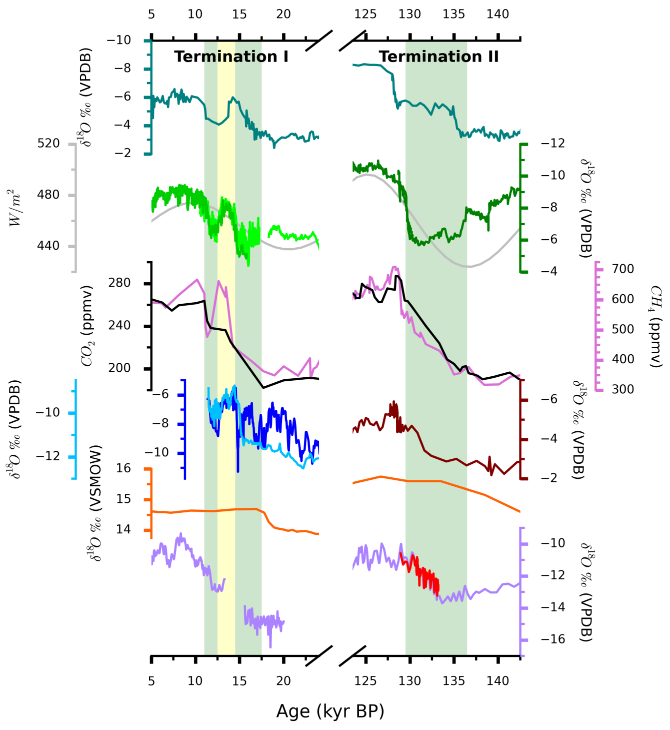

Fig. 1: A TrendVis figure illustrating the similarities and differences among climate records from Israel [BarMatthews], China [Wang], [Dykoski], [Sanbao]; Italy [Drysdale], the American Southwest [Wagner], [Asmerom], and Great Basin region [Winograd0], [Winograd1], [Lachniet], [Shakun] between the last deglaciation and the penultimate deglaciation (respectively known as Termination I and Termination II). Most of these records are stalagmite oxygen isotope records - oxygen isotopes, depending on the location, may record temperature changes, changes in precipitation seasonality, or other factors. All data are available online as supplementary materials or through the National Climatic Data Center.

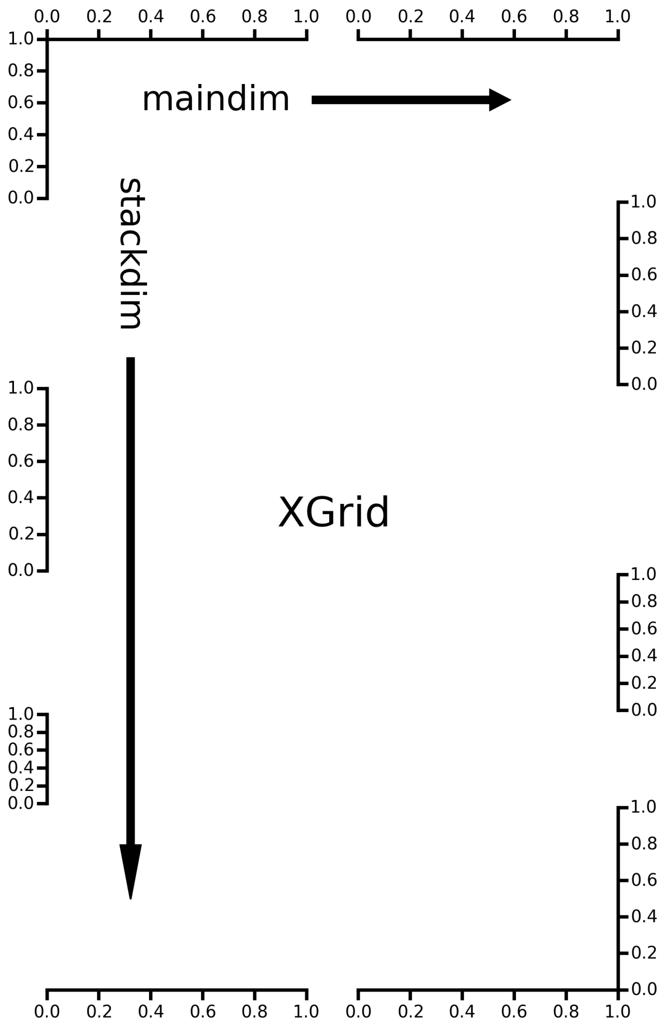

Fig. 2: In `XGrid`*,* `stackdim` *refers to number of rows of y axes and* `maindim` *indicates the number of columns. This is reversed in* `YGrid`*. Both dimension labels begin in* `XGrid.axes[0][0]`*.*

these plots in the scientific Python ecosystem using a matplotlib-based workflow. Here we discuss how TrendVis interfaces with matplotlib to construct and format this complex plot type as well as several challenges faced while we walk through the creation of Figure 1.

The TrendVis Figure Framework

The backbone of TrendVis is the `Grid` class, in which the figure, basic attributes, and orientation-agnostic methods are initialized. `Grid` should only be initialized through one of its two subclasses, `XGrid` and `YGrid`. As a common application of these types of plots is time-series data, we will examine TrendVis from the perspective of `XGrid`. In `XGrid`, the x axis is shared among all the datasets, and y axes are individual - in the terminology of TrendVis, x axes are the main axes, and y axes are the stacked axes. This is reversed for `YGrid`. A graphical representation of `XGrid` is shown in Figure 2.

TrendVis figures appear to consist of a common plot space. This, however, is an illusion carefully crafted via a framework of axes and a mechanism to systematically hide extra axes spines, ticks, and labels. This framework is created when the figure is initialized:

```
paleofig = XGrid([7, 8, 8, 6, 4, 8], xratios=[1, 1],
                 figsize=(6,10))
```

First, let's examine the construction of this framework. The overall area of the figure is determined by `figsize`, which is passed to matplotlib. The relative sizes of the rows (`ystack_ratios`, the first argument), however, is determined by the contents of `ystack_ratios` and the sum of `ystack_ratios` (`self.gridrows`), which in this case is 41. Similarly, the contents and sum of `xratios` (`self.gridcols`) determine the relative sizes of the columns. So, all axes in `paleofig` are initialized on a 41 row, 2 column grid within the 6 x 10 inch space set by `figsize`. The axis in position 0,0, (2) spans 7/41 unit rows (0 through 6) and the first unit column; the next axis created spans the same unit rows and the second unit column, finishing the first row of `paleofig`. The next row spans 8 unit rows, numbers 7 through 15, and so on. All axes in the same row share a y axis, and all axes in the same column share an x axis. This axes creation process, shown in the code below, is repeated for all the values in `ystack_ratios` and `xratios`, yielding a figure with 6 rows and 2 columns of axes. The code below and all other unnumbered snippets indicate an internal process rather than part of the `paleofig` workflow.

```
xpos = 0
ypos = 0

# Create axes row by row
for rowspan in self.yratios:
    row = []

    for c, colspan in enumerate(self.xratios):
        sharex = None
        sharey = None

        # All ax in row share y with first ax in row
        if xpos > 0:
            sharey = row[0]

        # All ax in col share x with first ax in col
        if ypos > 0:
            sharex = self.axes[0][c]

        ax = plt.subplot2grid((self.gridrows,
                               self.gridcols),
                              (ypos, xpos),
                              rowspan=rowspan,
                              colspan=colspan,
                              sharey=sharey,
                              sharex=sharex)

        ax.patch.set_visible(False)

        row.append(ax)
        xpos += colspan

    self.axes.append(row)

    # Reset x position to left, move to next y pos
    xpos = 0
    ypos += rowspan
```

Axes are stored in `paleofig.axes` as a nested list, where the sublists contain axes in the same rows. Next, two parameters that dictate spine visibility are initialized:

`paleofig.dataside_list`
: This list indicates where each row's y axis spine, ticks, and label are visible. This by default alternates sides from left to right (top to bottom in `YGrid`), starting at left, unless indicated otherwise during the

initialization of `paleofig`, or changed later on by the user.

`paleofig.stackpos_list`
: This list controls the x (main) axis visibility. Each row's entry is based on the physical location of the axis in the plot; by default only the x axes at the top and bottom of the figure are shown and the x axes of middle rows are invisible. Each list is exposed and can be user-modified, if desired, to meet the demands of the particular figure.

These two lists serve as keys to TrendVis formatting dictionaries and as arguments to axes (and axes child) methods. At any point, the user may call:

```
paleofig.cleanup_grid()
```

and this method will systematically adjust labelling and limit axis spine and tick visibility to the positions indicated by `paleofig.dataside_list` and `paleofig.stackpos_list`, transforming the mess in Figure 3 to a far clearer and more readable format in Figure 2.

Creating Twinned Axes

Although for large datasets, using twinned axes as the sole plotting tool is unadvisable, select usage of twinned axes can improve data visualization. In the case of `XGrid`, a twinned axis is a new axis that shares the x axis of the original axis *but* has a different y axis on the opposite side of the original y axis. Using twins allows the user to directly overlay datasets. TrendVis provides the means to easily and systematically create and manage entire rows (`XGrid`) or columns (`YGrid`) of twinned axes.

In our `paleofig`, we need four new rows:

```
paleofig.make_twins([1, 2, 3, 3])
paleofig.cleanup_grid()
```

This creates twinned x axes, one per column, across the four rows indicated and hides extraneous spines and ticks, as shown in Figure 4. As with the original axes, all twinned axes in a column share an x axis, and all twinned axes in the twin row share a y axis. The twin row information is appended to `paleofig.dataside_list` and `paleofig.stackpos_list` and twinned axes are stored at the end of the list of axes, which previously contained only original rows. If the user decides to get rid of twin rows (`paleofig.remove_twins()`), `paleofig.axes`, `paleofig.dataside_list`, and `paleofig.stackpos_list` are returned to their state prior to adding twins.

Accessing Axes

Retrieving axes, especially when dealing with twin axes in a figure with many hapazardly created twins, can sometimes be non-straightforward. The following means are available to return individual axes from a TrendVis figure:

`paleofig.fig.axes[axes index]`
: Matplotlib stores axes in a 1D list in `Figure` in the order of creation. This method is easiest to use when dealing with an `XGrid` of only one column.

`paleofig.axes[row][column]`
: An `XGrid` stores axes in a nested list in the order of creation, no matter its dimensions. Each sublist contains all axes that share the same y axis- a row. The row index corresponds to the storage position in the list, not the actual physical position on the grid, but in original axes (those created when `paleofig` was initialized) these are the same.

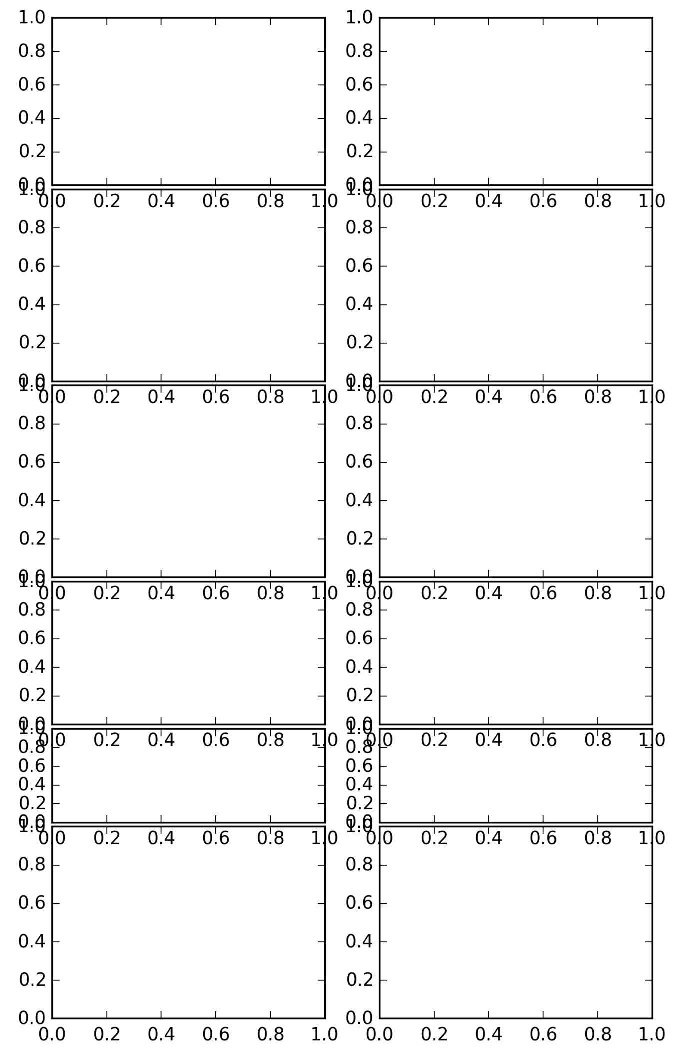

***Fig. 3:** Freshly initialized `XGrid`. After running `XGrid.cleanup_Grid()` (and two formatting calls adjusting the spinewidth and tick appearance), the structure of Figure 2 is left, in which stack spines are staggered, alternating sides according to `XGrid.dataside_list`, starting at left.*

`paleofig.get_axis()`
: Any axis can be retrieved from `paleofig` by providing its physical row number (and if necessary, column position) to `paleofig.get_axis()`. Twins can be parsed with the keyword argument `is_twin`, which directs `paleofig.twin_rownum()` to find the index of the sublist containing the twin row.

In the case of `YGrid`, the row, column indices are flipped: `YGrid.axes[column][row]`. Sublists correspond to columns rather than rows.

Plotting and Formatting

The original TrendVis procedurally generated a simple, 1-column version of `XGrid`. Since the figure was made in a single function

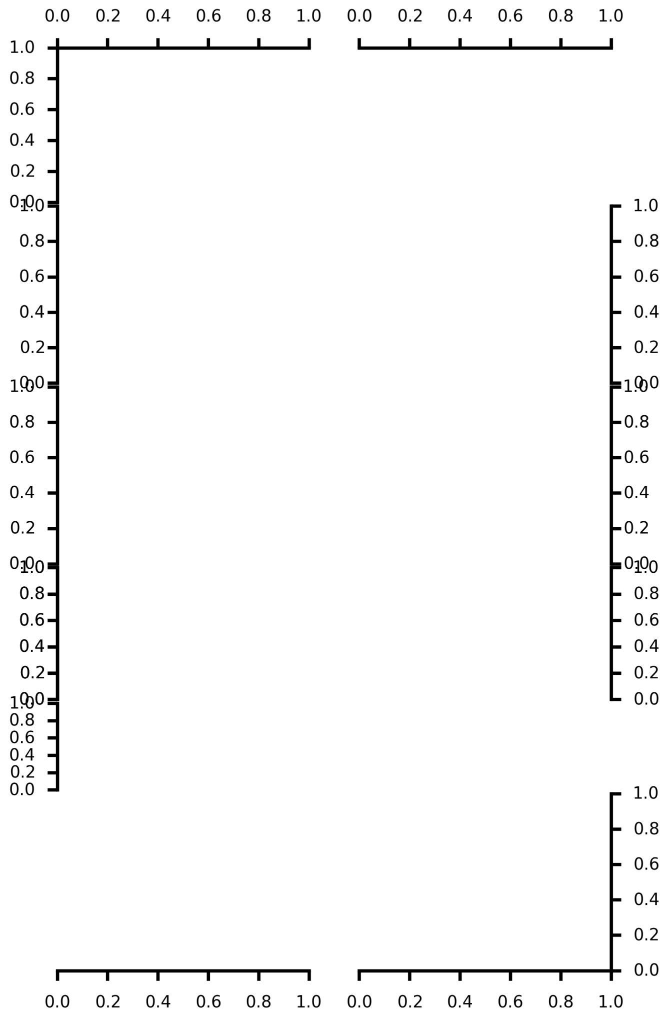

Fig. 4: The results of `paleofig.make_twins()`*, performing another grid cleanup and some minor tick/axis formatting.*

call, all data had to be provided at once in order, and it all had to be line/point data, as only `Axes.plot()` was called. TrendVis still provides convenience fuctions `make_grid()` and `plot_data()` to enable easy figure initialization and quick line plotting on all axes with fewer customization options. The regular object-oriented API is designed to be a highly flexible wrapper around matplotlib. Axes are readily exposed via the matplotlib and TrendVis methods described above, and so the user can determine the most appropriate plotting functions for their figure. The author has personally used `Axes.errorbar()`, `Axes.fill_betweenx()`, and `Axes.plot()` on two published TrendVis figures (see figures 3 and 4 in [Cross]), which required the new object-oriented API. Rather than make individual calls to plot on each axis, we will use the convenience function `plot_data`. The datasets have been loaded from a spreadsheet into individual 1D NumPy [NumPy] arrays containing age information or climate information:

```
plot_data(paleofig,[[(sorq_age, sorq, '#008080')],
                    [(hu_age, hu, '#00FF00',[0]),
                     (do_age, do, '#00CD00', [0]),
                     (san_age, san, 'green', [1])],
                    [(co2age, co2, 'black')],
                    [(cor_age, cor, 'maroon', [1])],
                    [(dh_age, dh, '#FF6103')],
                    [(gb_age, gb, '#AB82FF'),
                     (leh_age, leh, 'red', [1])],
                    [(insol_age, insol, '0.75')],
                    [(ch4_age, ch4, 'orchid')],
                    [(fs_age, fs, 'blue')],
                    [(cob_age, cob, '#00BFFF')]],
          marker=None, lw=2, auto_spinecolor=False)
```

Using `plot_data`, simple line plotting only requires a tuple of the x and y values and the color in a sublist in the appropriate row order. Some tuples have a fourth element that indicates which column the dataset should be plotted on. Without this element, the dataset will be plotted on all, or in this case both columns. Setting different x axis limits for each column will mask this fact.

Although plots individualized on a per axis basis may be important to a user, most aspects of axis formatting should generally be uniform. In deference to that need and to potentially the sheer number of axes in play, TrendVis contains wrappers designed to expedite these repetitive axis formatting tasks, including setting major and minor tick locators and dimensions, axis labels, and axis limits.

```
paleofig.set_ylim([(3, -7, -2), (4, 13.75, 16),
                   (5, -17, -9),
                   (6, 420, 520, (7, 300, 725),
                   (8, -11.75, -5))])

paleofig.set_xlim([(0, 5, 24), (1, 123.5, 142.5)])

paleofig.reverse_yaxis([0, 1, 3])

paleofig.set_all_ticknums([(5, 2.5), (5, 2.5)],
                          [(2,1),(2,1),(40,20),(2,1),
                           (1,0.5), (2,1),(40,20),
                           (100,25),(2,1),(2,1)])

paleofig.set_ticks(major_dim=(7, 3), labelsize=11,
                   pad=4, minor_dim=(4, 2))

paleofig.set_spinewidth(2)

# Special characters for axis labels
d18o = r'$\delta^{18}\!O$'
d13c = r'$\delta^{13}\!C$'
d234u = r'$\delta^{234}\!U_{initial}$'
co2label = r'$CO_{2}$'
ch4label = r'$CH_{4}$'
mu = ur'$\u03BC$'
vpdb = ' ' + ur'$\u2030$'+ ' (VPDB)'
vsmow =' ' + ur'$\u2030$'+' (VSMOW)'

paleofig.fig.suptitle('Age (kyr BP)', y=0.065,
                      fontsize=16)
paleofig.set_ylabels([d18o + vpdb, d18o + vpdb,
                      co2label +' (ppmv)',
                      d18o + vpdb,
                      d18o + vsmow, d18o + vpdb,
                      r'$W/m^{2}$',
                      ch4label + ' (ppmv)', '',
                      d18o + vpdb, d13c + vpdb],
                      fontsize=13)
```

In this plot style, there are two other formatting features that are particularly useful: moving data axis spines, and automatically coloring spines and ticks. The first involves the lateral movement of data axis (y axis in `XGrid`, x axis in `YGrid`) spines into or out of the plot space. Although the default TrendVis behavior is alternating the data axis spines from left to right, resulting in space between data axis spines, adding twin rows disrupts this pattern and spacing, as shown in Figure 5. This problem is exacerbated when compacting the figure, which is a typical procedure in this plot type, to improve both the look of the figure and its readability. The solution in `XGrid` plots is to move spines laterally- along the

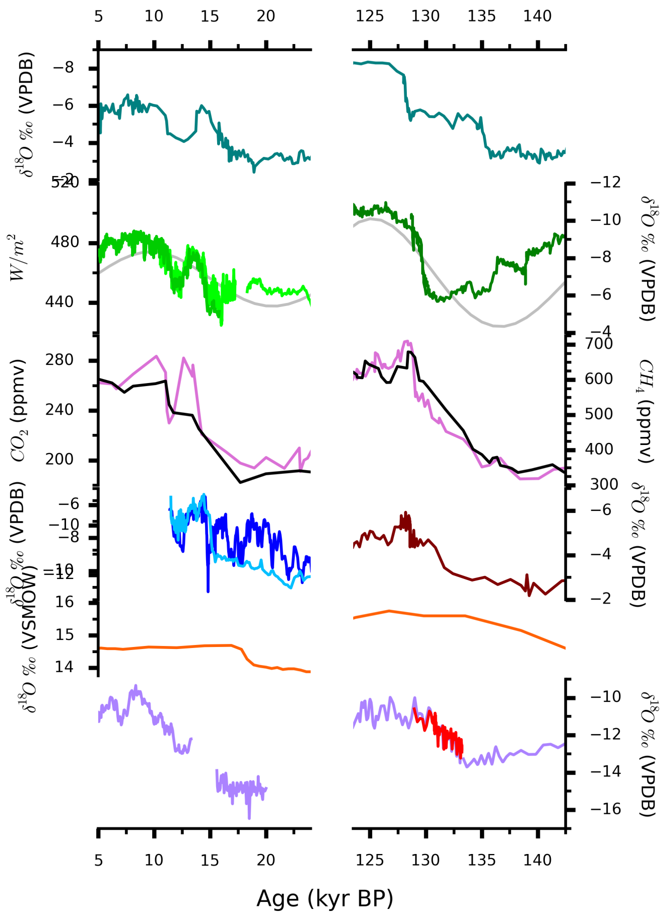

Fig. 5: Figure after plotting paleoclimate time series records, editing the axes limits, and setting the tick numbering and axis labels. At this point it is difficult to see which dataset belongs to which axis and to clearly make out the twin axis numbers and labels.

x dimension- out of the way of each other, into or out of the plot space. TrendVis provides means to expedite the process of moving spines:

```
# Make figure more compact:
paleofig.fig.subplots_adjust(hspace=-0.4)

# Move spines
# Shifts are in fractions of figure
# Absolute position calc as 0 - shift (ax at left)
# or 1 + shift (for ax at right)
paleofig.move_spines(twin_shift=[0.45, 0.45,
                                 -0.2, 0.45])
```

In the above code, all four of the twinned visible y axis spines are moved by an individual amount; the user may set a universal `twin_shift` or move the y axis spines of the original axes in the same way. Alternatively, all TrendVis methods and attributes involved in `paleofig.move_spines()` are exposed, and the user can edit the axis shifts manually and then see the results via `paleofig.execute_spineshift()`. As the user-provided shifts are stored, if the user changes the arrangement of visible y axis spines (via `paleofig.set_dataside()` or by directly altering `paleofig.dataside_list`), then all the user needs to do to get the old relative shifts applied to the new arrangement is get TrendVis to calculate new spine positions (`paleofig.absolute_spineshift()`) and perform the shift (`paleofig.execute_spineshift()`).

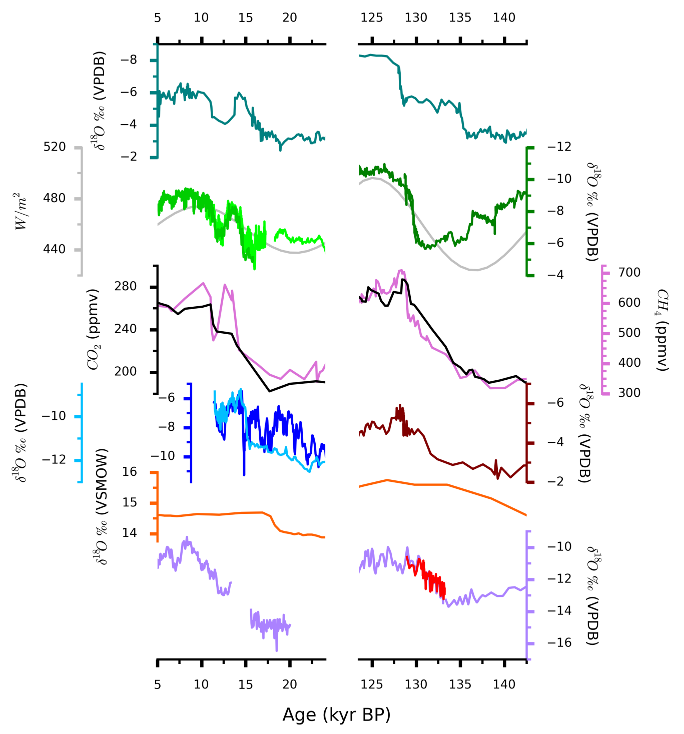

Fig. 6: Although the plot is very dense, the lateral movement of spines and coloring them to match the curves has greatly improved the readability of this figure relative to Figure 5. The spacing between subplots has also been decreased.

Although the movement of y axis spines allows the user to read each axis, there is still a lack of clarity in which curve belongs with which axis, which is a common problem for this plot type. TrendVis' second useful feature is automatically coloring the data axis spines and ticks to match the color of the first curve plotted on that axis. As we can see in Figure 6, this draws a visual link between axis and data, permitting most viewers to easily see which curve belongs against which axis.

```
paleofig.autocolor_spines()
```

Visualizing Trends

Large stacks of curves are overwhelming to viewers. In complicated figures, it is critical to not only keep the plot area tidy and link axes with data, as we saw above, but also to draw the viewer's eye to essential features. This can be accomplished with shapes that span the entire figure, highlighting areas of importance or demarcating particular spaces. In `paleofig`, we are interested in the glacial terminations. Termination II coincided with a North Atlantic cold period, while during Termination I there were two cold periods interrupted by a warm interval:

```
# Termination I needs three bars, get axes that will
# hold the lower left, upper right corners of bar
ll = paleofig.get_axis(5)
ur = paleofig.get_axis(0)
alpha = 0.2

paleofig.draw_bar(
 ll, ur, (11, 12.5), alpha=alpha,
 edgecolor='none', facecolor='green')
paleofig.draw_bar(
 ll, ur, (12.5, 14.5), alpha=alpha,
 edgecolor='none', facecolor='yellow')
```

```
paleofig.draw_bar(
 ll, ur, (129.5, 136.5), alpha=alpha,
 edgecolor='none', facecolor='green')

# Draw bar for Termination II, in column 1
paleofig.draw_bar(paleofig.get_axis(5, xpos=1),
                  paleofig.get_axis(0, xpos=1),
                  (129.5, 136.5), alpha=alpha,
                  facecolor='green',
                  edgecolor='none')

# Label terminations
ax2 = paleofig.get_axis(0, xpos=1)
paleofig.ax2.text(133.23, -8.5, 'Termination II',
                  fontsize=14, weight='bold',
                  horizontalalignment='center')

ax1 = paleofig.get_axis(0)
paleofig.ax1.text(14, -8.5, 'Termination I',
                  fontsize=14, weight='bold',
                  horizontalalignment='center')
```

The user provides the axes containing the lower left corner of the bar and the upper right corner of the bar. In the vertical bars of `paleofig` the vertical limits consist of the upper limit of the upper right axis and the lower limit of the lower left axis. The horizontal upper and lower limits are provided in data units, for example (11, 12.5). The default zorder is -1 in order to place the bar behind the curves, preventing data from being obscured.

As these bars typically span multiple axes, they must be drawn in Figure space rather than on the axes. This presents two challenges. The first is converting data coordinates to figure coordinates. In the private function `_convert_coords()`, we transform data coordinates (`dc`) into axes coordinates, and then into figure coordinates:

```
ac = ax.transData.transform(dc)

fc = self.fig.transFigure.inverted().transform(ac)
```

The figure coordinates are then used to determine the width, height, and positioning of the Rectangle in figure space.

TrendVis strives to be as order-agnostic as possible. However, a patch drawn in Figure space is completely divorced from the data the patch is supposed to highlight. If axes limits are changed, or the vertical or horizontal spacing of the plot is adjusted, then the bar will no longer be in the correct position relative to the data.

As a solution, for each bar drawn with TrendVis, the upper and lower horizontal and vertical limits, the upper right and lower left axes, and the index of the patch in XGrid.fig.patches are all stored as XGrid attributes. Storing the patch index allows the user to make other types of patches that are exempt from TrendVis' patch repositioning. When any of TrendVis' wrappers around matplotlib's subplot spacing adjustment, x or y limit settings, etc are used, the user can stipulate that the bars automatically be adjusted to new figure coordinates. The stored data coordinates and axes are converted to figure space, and the x, y, width, and height of the existing bars are adjusted. Alternatively, the user can make changes to axes space relative to figure space without adjusting the bar positioning and dimensions each time or without using TrendVis wrappers, and simply adjust the bars at the end.

TrendVis also enables a special kind of bar, a frame. The frame is designed to visually anchor data axis spines, and appears around an entire column (row in `YGrid`) of data axes under the spines. However, for `paleofig` we will use a softer division of our the columns by using cut marks on the main axes to signify a broken axis:

```
paleofig.draw_cutout(di=0.075)
```

Similar to bars, frames are drawn in figure space and can sometimes be moved out of place when axes positions are changed relative to figure space, thus they are handled in the same way. Cutouts, however, are actual line plots on the axes that live in axes space and will not be affected by adjustments in axes limits or subplot positioning. With the cut marks drawn on `paleofig`, we have completed the dense but highly readable plot shown in Figure 1.

Conclusions and Moving Forward

TrendVis is a package that expedites the process of creating complex figures with multiple x or y axes against a common y or x axis. It is largely order-agnostic and exposes most of its attributes and methods in order to promote highly-customizable and reproducible plot creation in this particular style. In the long-term, with the help of the scientific Python community, TrendVis aims to become a widely-used higher level tool for the matplotlib plotting library and alternative to expensive software such as SigmaPlot and MATLAB, and to time-consuming, error-prone practices like assembling multiple Excel plots in vector graphics editing software.

References

[Petit] J. R. Petit et al. *Climate and Atmospheric History of the Past 420,000 years from the Vostok Ice Core, Antarctica* Nature, 399:429-436, 1999.

[BarMatthews] M. Bar-Matthews et al. *Sea--land oxygen isotopic relationships from planktonic foraminifera and speleothems in the Eastern Mediterranean region and their implication for paleorainfall during interglacial intervals*, Geochimica et Cosmochimica Acta, 67(17):3181-3199, 2003.

[Drysdale] R. N. Drysdale et al. *Stalagmite evidence for the onset of the Last Interglacial in southern Europe at 129 $pm$1 ka*, Geophysical Research Letters, 32(24), 2005.

[Wang] Y. J. Wang et al. *A high-resolution absolute-dated late Pleistocene monsoon record from Hulu Cave, China*, Science, 294(5550):2345-2348, 2001.

[Dykoski] C. A. Dykoski et al., *A high-resolution, absolute-dated Holocene and deglacial Asian monsoon record from Dongge Cave, China*, Earth and Planetary Science Letters, 233(1):71-86, 2005.

[Sanbao] Y. J. Wang et al. *Millennial-and orbital-scale changes in the East Asian monsoon over the past 224,000 years*, Nature, 451(7182):1090-1093, 2008.

[Wagner] J. D. M. Wagner et al. *Moisture variability in the southwestern United States linked to abrupt glacial climate change*, Nature Geoscience, 3:110-113, 2010.

[Asmerom] Y. Asmerom et al. *Variable winter moisture in the southwestern United States linked to rapid glacial climate shifts*, Nature Geoscience, 3:114-117, 2010.

[Winograd0] I. J. Winograd et al. *Continuous 500,000-year climate record from vein calcite in Devils Hole, Nevada*, Science, 258(5080):255-260, 1992.

[Winograd1] I. J. Winograd et al. *Devils Hole, Nevada, $delta$ 18 O record extended to the mid-Holocene*, Quaternary Research, 66(2):202-212, 2006.

[Lachniet] M. S. Lachniet et al. *Orbital control of western North America atmospheric circulation and climate over two glacial cycles*, Nature Communications, 5, 2014.

[Shakun] J. D. Shakun et al. *Milankovitch-paced Termination II in a Nevada speleothem?* Geophysical Research Letters, 38(18), 2011.

[matplotlib] J. D. Hunter. *Matplotlib: A 2D Graphics Environment*, Computing in Science & Engineering, 9:90-95, 2007.

[Cross] M. Cross et al. *Great Basin hydrology, paleoclimate, and connections with the North Atlantic: A speleothem stable isotope and trace element record from Lehman Caves, NV*, Quaternary Science Reviews, in press.

[NumPy] S. van der Walt et al. *The NumPy Array: A Structure for Efficient Numerical Computation*, Computing in Science & Engineering, 13:22-30, 2011.