# Closed-Source Development Deliverable 4

Justin Abrokwah
Yi Shiun Chiu
Spencer McCoubrey
Parth Thakkar

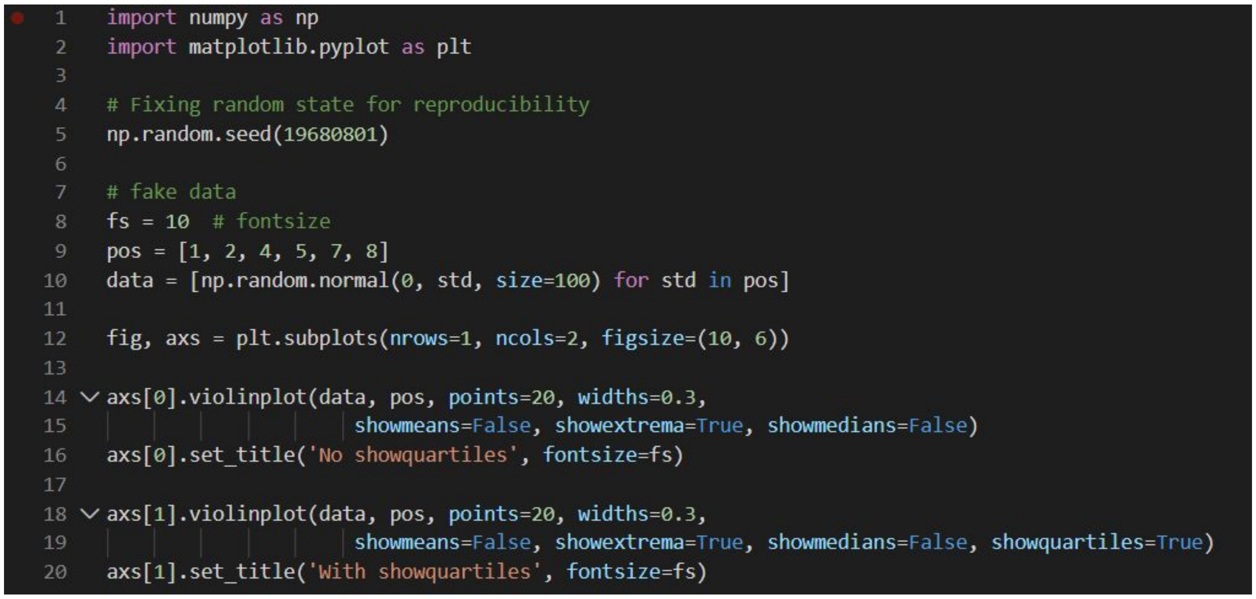
## User Guide: Feature Documentation

Violin plot is a method of plotting the full distribution of a dataset. Similar to a box plot, a violin plot displays the mean, median, extrema, and interquartile range of a set of data. In addition, it has the power to display the probability density of the distribution using a kernel density estimator, which is a method for estimating the probability density and gives our plot a violin-like figure. Overall, violin plots provide more rich information about a dataset, compared to a box plot.

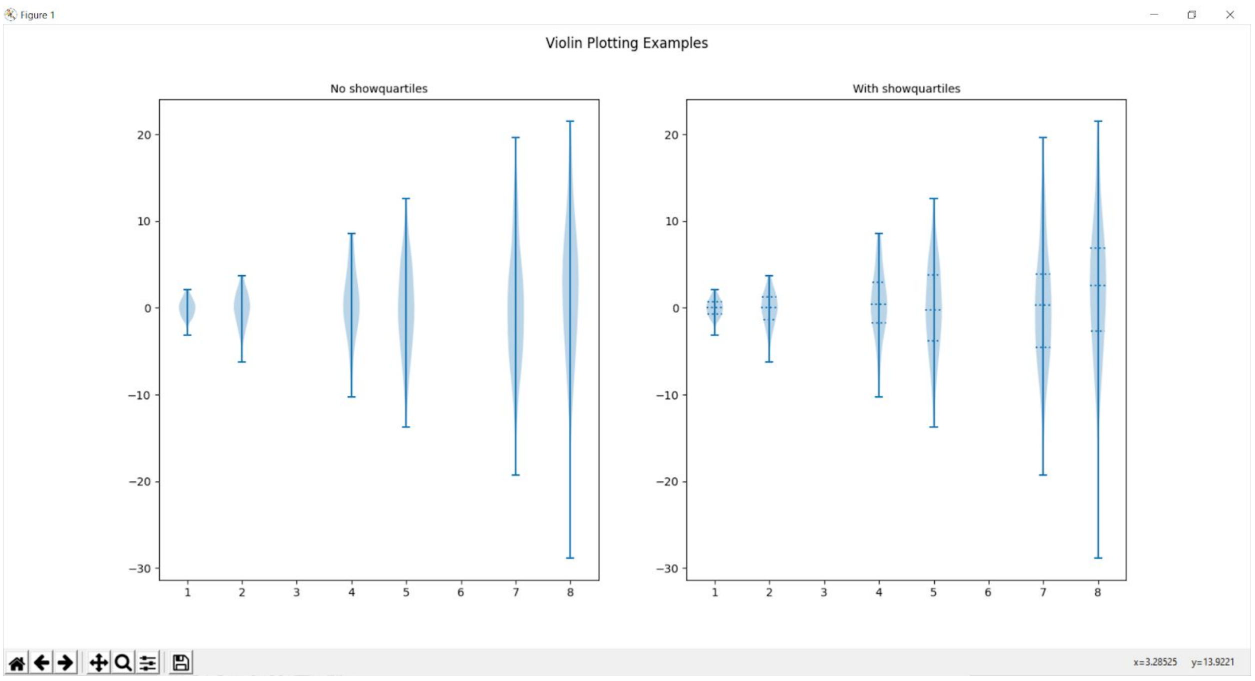
The Matplotlib library has the ability to produce a violin plot with information such as extrema, median and specific quantiles. However, it lacked the ability to to display the quartiles (and therefore interquartile range) of a distribution easily. Currently, to do so, the user would have to populate the *quantiles* parameter with an array of arrays including the values [0.25, 0.50, 0.75] for each column in the dataset alongside the other quantile values they desire to plot. The downside of this is that the user would always have to populate the quantiles parameter with these standard values. Also, if rendering more than these 3 quantiles, there would be no distinction between the quartiles (quantiles at [0.25, 0.50, 0.75]), and the other quantiles, making the violin plot less effective for visualization.

The feature implemented is a way to neutralize these problems by allowing the user to show quartiles for their desired plot. Newly introduced is a boolean parameter into the *violinplot* function called *showquartiles*. Similar to the other *show *field** parameters like *showextrema*, or *showmedians*, which when set true, render their specific features on the plot. The new parameter *showquartiles* does the same thing essentially and renders the quartiles of the distribution when set true, in a way that is visually distinctive from other quantile lines desired by the user. These quartile lines extend to the end of the violin and are dotted, to showcase their importance and distinction. The parameter is optional and defaults to False if the parameter is not provided.

An example usage of the *showquartiles* parameter is shown here:



```python
import numpy as np
import matplotlib.pyplot as plt

# Fixing random state for reproducibility
np.random.seed(19680801)

# fake data
fs = 10  # fontsize
pos = [1, 2, 4, 5, 7, 8]
data = [np.random.normal(0, std, size=100) for std in pos]

fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(10, 6))

axs[0].violinplot(data, pos, points=20, widths=0.3,
                  showmeans=False, showextrema=True, showmedians=False)
axs[0].set_title('No showquartiles', fontsize=fs)

axs[1].violinplot(data, pos, points=20, widths=0.3,
                  showmeans=False, showextrema=True, showmedians=False, showquartiles=True)
axs[1].set_title('With showquartiles', fontsize=fs)
```



## Feature Design Documentation

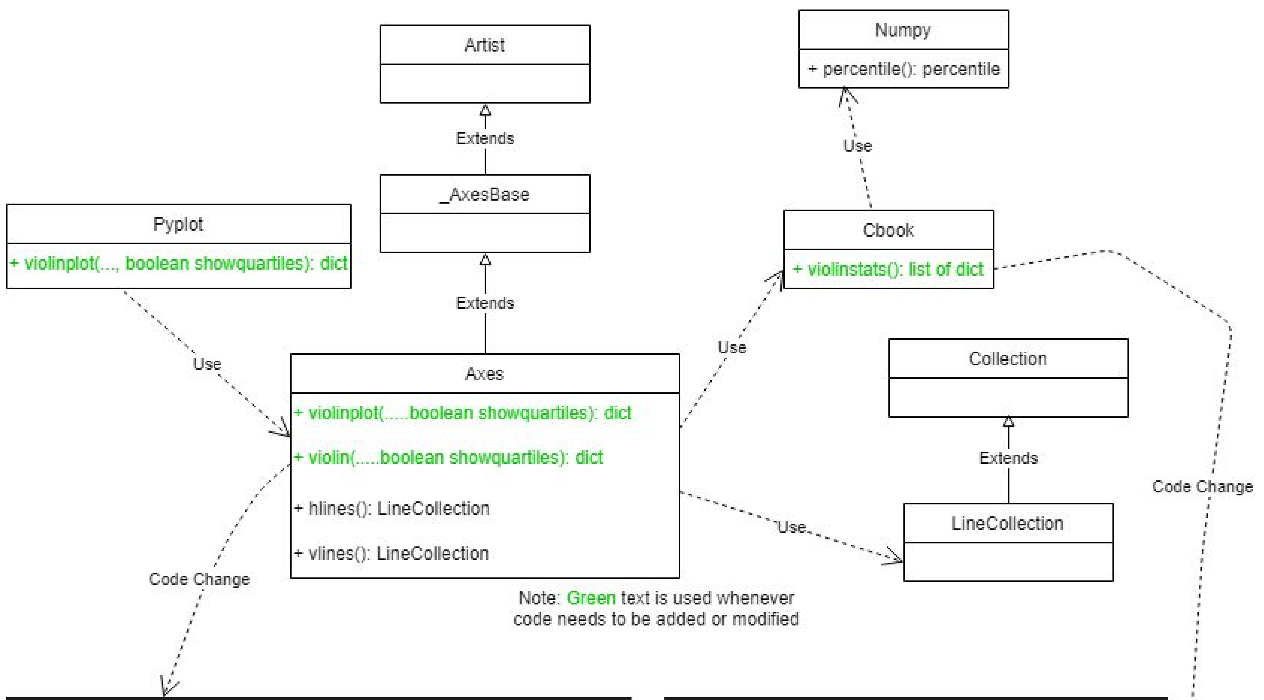
The implementation of this feature was developed with an aim of consistency with the rendering of other significant violin plot features, like the median, or extrema of the plot. The function provides a “toggle” option for the user to render them by passing in boolean parameters that are set to true for each feature they would like to plot. Our implementation closely follows this approach, producing a familiar experience for the user without introducing a learning curve.

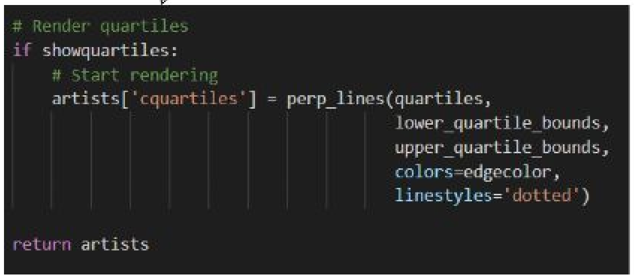
The beginning of our solution starts with adding a new boolean parameter to the Axes.violinplot and Pyplot.violinplot functions called *showquartiles*. Axes.violinplot calls upon a cbook.violin_stats function to organize the input dataset into a list of dictionaries containing statistics for each plot. To display the quartiles, we must first calculate the quartiles, which are equal to the 25th, 50th, and 75th percentiles of each column in the dataset. This is done using numpy and the values added to each dictionary in the list as a new entry called *quartiles*, pointing to a list.

The list of dictionaries is returned by the Axes.violinplot function and gets passed to the Axes.violinplot.violin function to generate the Artists for the plot. In this function, a few changes are made. Since a key feature of our implementation was to have our quartile lines extend to the edge of the violin, we create an empty array which holds sublists, representing the quartiles in each column in the dataset. Also, we create 2 empty arrays to hold the bounds of these lines, representing the ends of the lines. In the main for loop of the Axes.violinplot.violin function which iterates through our dictionary from the cbook.violin_stats function and creates the artists for each column of the dataset, on each cycle we retrieve our new entry (quartiles) from the dictionary and appends them to the quartiles array. It also calculates the closest point on the violin curve to each quartile using the coordinates array generated in cbook.violin_stats to plot the curve. This is how the quartile lines will extend to the edge of the violin.

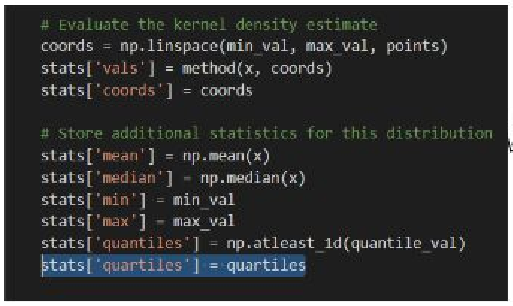
After this loop concludes, we check our *showquartiles* parameter. If it’s true, we create a new entry in our *artists* dictionary being returned by Axes.violinplot.violin (named ‘cquartiles’ to stay consistent with the other dictionary names being returned. It is equal to a perpendicular line (to the violin) using our lower and upper quartile bounds (to extend the lines to the end of the violin) that is dotted (to show the difference between quartiles and any quantiles that may have been rendered as well).

We made slight deviations from our design plan but kept the overall gist of consistency. In our original scheme, we planned to create multiple entries in the vpstats dictionary created in the cbook.violin_stats function named: *first_quartile* and *third_quartile*, representing their respective values. We were able to circumvent this and only add 1 new entry to the dictionary, *quartiles*, which is a numpy array of the quartiles (e.g. [1st quartile, 2nd quartile, 3rd quartile]). This allowed us to not bloat the current implementation with a lot of new variables to deal with. Also, in our original plan we didn't account for the length of the quartile lines. We first intended to use the same approach as the quantiles lines where a static length is attached to each quantile. However, we saw that the Matplotlib maintainers requested a more dynamic approach where the quartile lines extend to the end of the violin, regardless of its shape. So for our implementation, we were able to use the coordinates of the violin curve to determine the necessary length for our quartile lines. This was done in the for loop in the Axes.violinplot.violin function where the lower and upper quartile bounds are calculated. The diagram below shows the UML around our implementation and where the main changes occurred:


Artist
Extends
_AxesBase
Extends
Axes
+ violinplot(.....boolean showquartiles): dict
+ violin(.....boolean showquartiles): dict
+ hlines(): LineCollection
+ vlines(): LineCollection
Pyplot
+ violinplot(..., boolean showquartiles): dict
Use
Numpy
+ percentile(): percentile
Use
Cbook
+ violinstats(): list of dict
Use
Collection
Extends
LineCollection
Use
Code Change
Code Change
Note: Green text is used whenever code needs to be added or modified



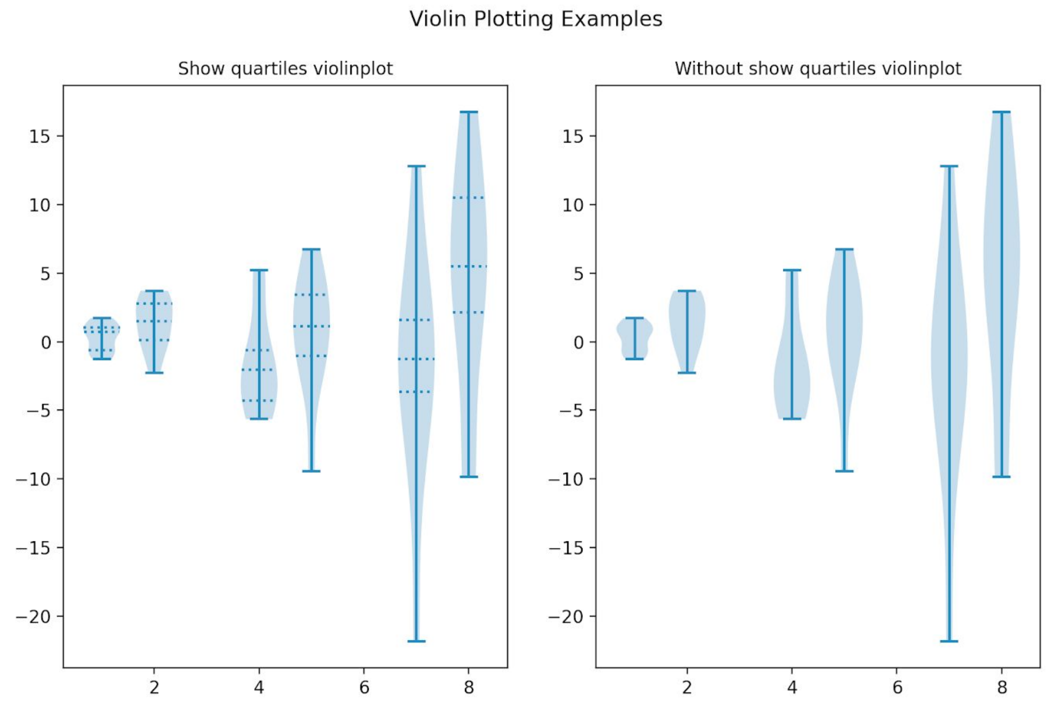
```
# Render quartiles
if showquartiles:
    # Start rendering
    artists['cquartiles'] = perp_lines(quartiles,
                                       lower_quartile_bounds,
                                       upper_quartile_bounds,
                                       colors=edgecolor,
                                       linestyles='dotted')

return artists
```



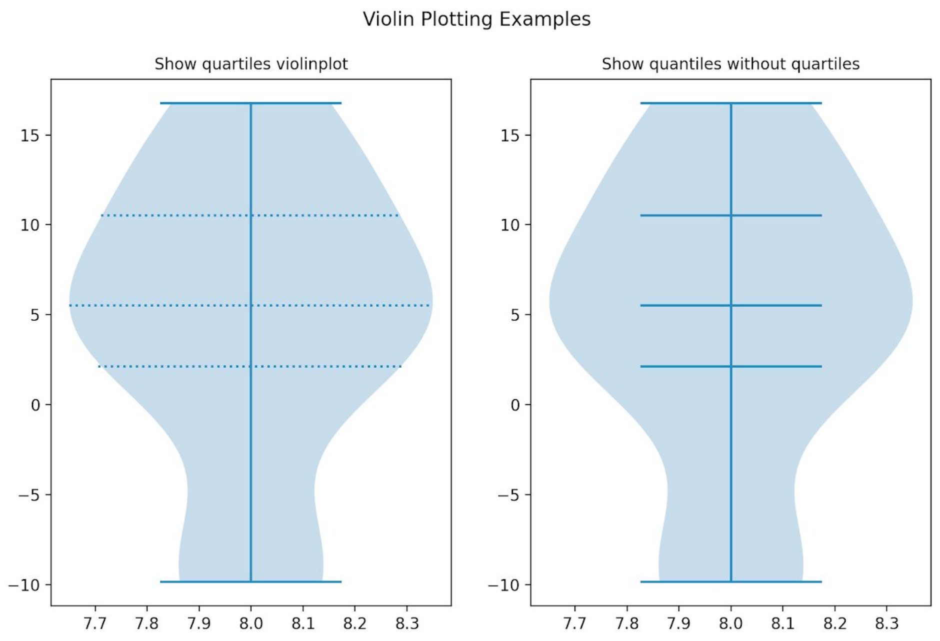
```
# Evaluate the kernel density estimate
coords = np.linspace(min_val, max_val, points)
stats['vals'] = method(x, coords)
stats['coords'] = coords

# Store additional statistics for this distribution
stats['mean'] = np.mean(x)
stats['median'] = np.median(x)
stats['min'] = min_val
stats['max'] = max_val
stats['quantiles'] = np.atleast_1d(quantile_val)
stats['quartiles'] = quartiles
```

## Acceptance Test Suite

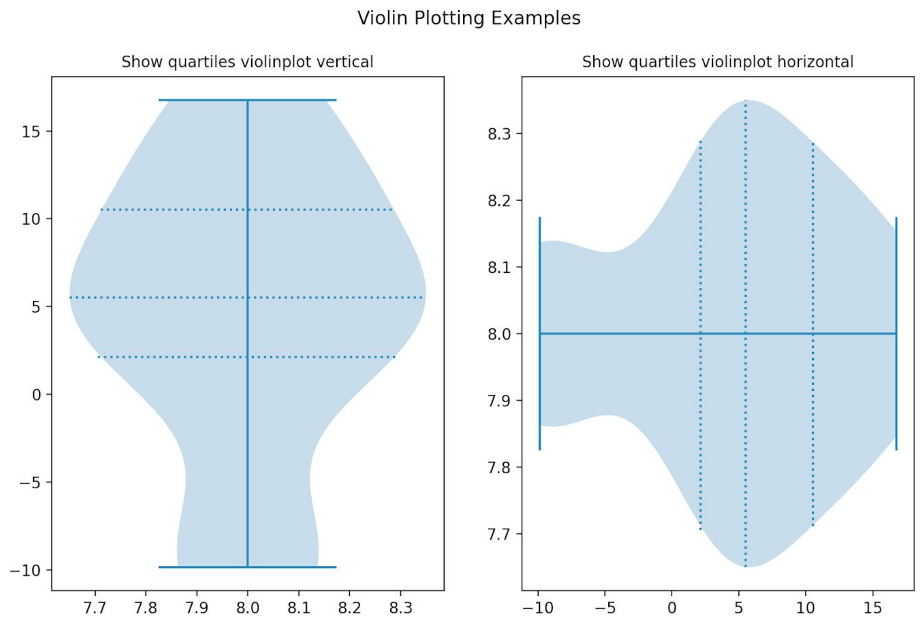
For our implementation, we have to perform a number of tests to validate our code's correctness. For all our tests, we have to construct datasets and provide positioning to the *violinplot* function. We will pass in our new parameter *showquartiles* as True. Using the *violinplot* function, we will render our plot and check whether the quartiles are drawn in the right places and extend to the edge of the violins. To verify, in another violin plot with the same dataset and positioning, we set our *showquartiles* to False, and we pass in quantiles of 0.25, 0.5, and 0.75. The quantiles should be rendered in the same places as the quartiles but will not extend to the edge of the violins. Then we can compare the 2 graphs using the image comparison feature used in numerous Matplotlib tests. Additionally, we set our *showquartiles parameter* to False and ensure that no unnecessary lines are drawn on the plot. Lastly, we will need to test the functionality of other parameters such as *showmedian* and quartiles with the *showquartiles* set to true to see how they would all be displayed concurrently on the graph. Pre-existing test cases using the functions we modify should still pass, as well. Since horizontal and vertical violin plots are possible, we must test our feature against both orientations of a violin plot.



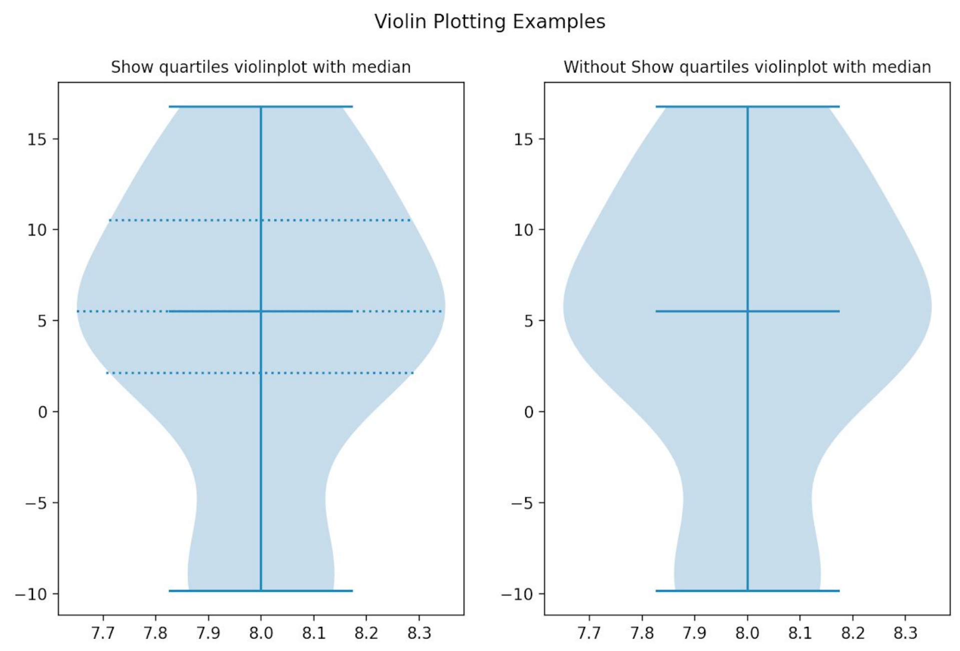
This figure shows that violin plot with show quartiles and without show quartiles along with the extremas.



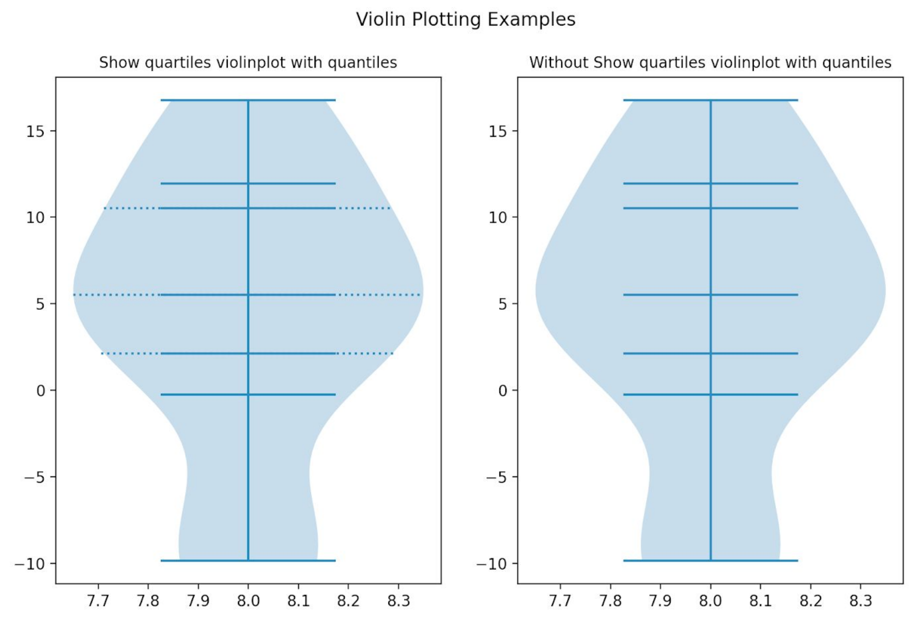
This example shows the violin plot with show quartiles enabled and another violin plot where show quartiles is false but the quantiles are set to [0.25,0.5,0.75].



This example shows the violin plot with show quartiles enabled in both horizontal and vertical orientation. The plot ensures that the quartile lines are drawn correctly in both types of violin plots.



This example shows the usability of showquartiles with showmedian. We can see that the previous functionality of median is preserved while the quartiles are also plotted on the graph in dotted lines that extend to the edge of the graph.



This last example shows the usability of quantiles with *showquartiles* set to true. The quantile array passed in the function is [0.2,0.25,0.5,0.75,0.8]. We can see that even with the same values of 0.25,0.5 and 0.75, the quartiles are distinct and extend to the edge of the violin.

## Unit Test Suite

Matplotlib has a robust suite of Unit Tests that cover every major component class in the library. We are adding a new functionality to the existing violin plot function found in the Axes component so we will be adding tests within the Axes test suite. Using the existing tests as a guideline, we must modify or add the following tests:

**MODIFY** baseline violinplot tests (vertical and horizontal):
`lib/matplotlib/tests/test_axes.py::test_vert_violinplot_baseline`
`lib/matplotlib/tests/test_axes.py::test_horiz_violinplot_baseline`
These are the baseline image comparison tests to verify the core violin plot function is rendering properly in both the vertical and horizontal orientation. We must explicitly specify that the showquartiles parameter is **false** in both tests.

**MODIFY** previous isolation tests (vertical and horizontal):
`lib/matplotlib/tests/test_axes.py::test_vert_violinplot_showmeans`
`lib/matplotlib/tests/test_axes.py::test_vert_violinplot_showextrema`
`lib/matplotlib/tests/test_axes.py::test_vert_violinplot_showmedians`
`lib/matplotlib/tests/test_axes.py::test_horiz_violinplot_showmedians`
`lib/matplotlib/tests/test_axes.py::test_horiz_violinplot_showmeans`
`lib/matplotlib/tests/test_axes.py::test_horiz_violinplot_showextrema`
These are the image comparison tests that verify specific components of violinplot are rendering properly in isolation. We must explicitly specify that the showquartiles parameter is **false** in all tests.

**MODIFY** showall tests (vertical and horizontal)
`lib/matplotlib/tests/test_axes.py::test_vert_violinplot_showall`
`lib/matplotlib/tests/test_axes.py::test_horiz_violinplot_showall`
These are the image comparison tests that verify that all components of violinplot are rendering properly when processed concurrently. We must generate new reference images that render all components including the quartile lines that we implemented. Moreover, we must modify the script that generates our test image by setting the showquartiles parameter to **True**.

**ADD** showquartiles tests (vertical and horizontal)
`lib/matplotlib/tests/test_axes.py::test_vert_violinplot_showquartiles`
`lib/matplotlib/tests/test_axes.py::test_horiz_violinplot_showquartiles`

These are the image comparison tests that verify the showquartiles feature is rendering properly in isolation. We must generate a new reference image given a randomly seeded dataset and write a script that generates the same image.