Matplotlib

Deliverable 3: Bug Fixes Report

By: Team Legend (Team 4)

Table of Contents

Bug-Fix for Issue 8059

Issue URL: https://github.com/matplotlib/matplotlib/issues/8059

Source of Problem:

In reviewing commentary about issue #8059 it became clear that the issue is related to changes made for issue #5757 that sets the value of the kwargs argument fill=False for all inputs to **`BboxConnectorPatch`** and **`BboxConnector`**, (fill=False is set in **`BboxConnector`** which is called when **`BboxConnectorPatch`** is initialized, see: inset_locator.py, Line 315).

Further discussion in #8075 made it clear that the desired behaviour for **`BboxConnectorPatch`** is: if any input is given for the color or facecolor kwargs arguments (or one of their aliases) the value of the fill kwargs argument should be set to True. It was also suggest by members of the matplotlib team that the cbook function **`normalize_kwargs`** should be used to match aliases with the normalized argument names.

Finally after reviewing the documentation for **`BboxConnectorPatch`** it became clear that fill is also a valid kwargs argument, and the current solution to #5757 would also cause a **`TypeError`** to be thrown if a value for the fill argument was given, specifically:

```
TypeError: __init__() got multiple values for keyword argument 'fill'
```

because multiple values would have been given for the fill kwargs argument.

While examining inset_locator.py it also became clear that the solution to #5757 also caused the same problem in the **`mark_inset`** function, which as you can see from the documentation accepts the same inputs as kwargs arguments, and suffers from the setting of **`fill=False`** with no logic to handle conflicting kwargs arguments, see inset_locator.py, Line 585. The same solution (with minor modification) can be applied to this function as well.

Suggested Fix:

The suggested fix is to add the following logic to **`BboxConnector`** and **`mark_inset`**:

normalize the kwargs using the cbook function normalize_kwargs, then

```
if ('fill' in kwargs):
    Pass normalized kwargs unchanged to constructor (Patch or BboxPatch).
else if ('facecolor' in kwargs) or ('color' in kwargs)
    Set fill=True and pass normalized kwargs to constructor (Patch or BboxPatch).
else:
    Set fill=False and pass normalized kwargs  to constructor (Patch or BboxPatch).
```

Note: In looking at the **`Patch`** class in `patches.py`, it was discovered that there is also a definition for **`_patch_alias_map`** which defines all possible kwargs aliases for Patch and its descendants in one place; however having the private identifier, there was some hesitation in accessing it. To determine how best to handle this, we looked for other uses in the matplotlib code and found a very similar usage in matplotlib/axes/_axes.py, Line 1971. We decided to follow this format for our addition to make maintenance easier in future (i.e. any update to the patch aliases will automatically carry over to our fix; no magic words).

Files Modified: matplotlib/lib/mpl_toolkits/axes_grid1/inset_locator.py

These changes required the additional imports:

```
from matplotlib.cbook import normalize_kwargs
import matplotlib.patches as Patches
```

For implemented fix in BboxConnectorPatch/BboxConnector see:
matplotlib/lib/mpl_toolkits/axes_grid1/inset_locator.py, Line 318 to Line 324

Screenshots:

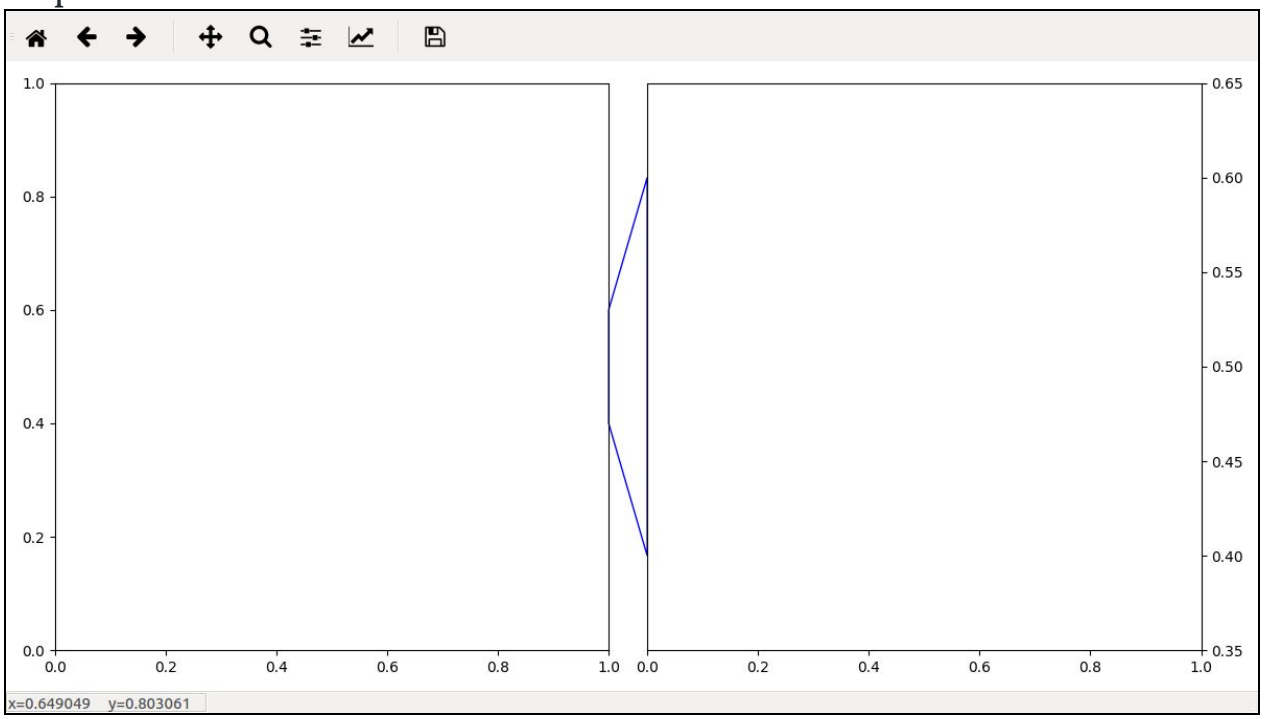
Before:

```
311         if "transform" in kwargs:
312             raise ValueError("transform should not be set")
313
314         kwargs["transform"] = IdentityTransform()
315         Patch.__init__(self, fill=False, **kwargs)
316         self.bbox1 = bbox1
317         self.bbox2 = bbox2
318         self.loc1 = loc1
319         self.loc2 = loc2
```

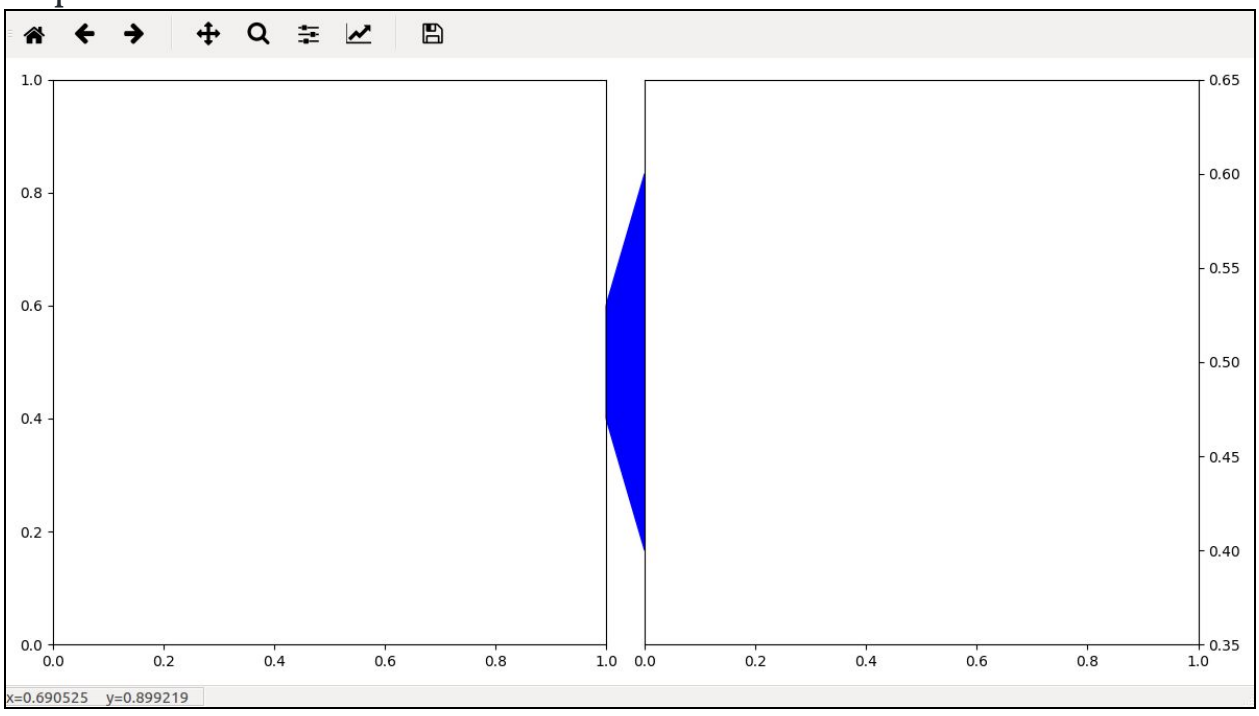
After:

```
313         if "transform" in kwargs:
314             raise ValueError("transform should not be set")
315
316         kwargs["transform"] = IdentityTransform()
317
318         kwargs = normalize_kwargs(kwargs, Patches._patch_alias_map)
319         if 'fill' in kwargs:
320             Patch.__init__(self, **kwargs)
321         elif ('facecolor' in kwargs) or ('color' in kwargs):
322             Patch.__init__(self, fill=True, **kwargs)
323         else:
324             Patch.__init__(self, fill=False, **kwargs)
325         self.bbox1 = bbox1
326         self.bbox2 = bbox2
327         self.loc1 = loc1
328         self.loc2 = loc2
```

Output Before:



Output After:



For implemented fix in mark_inset see:
matplotlib/lib/mpl_toolkits/axes_grid1/inset_locator.py, Line 594 to Line 600

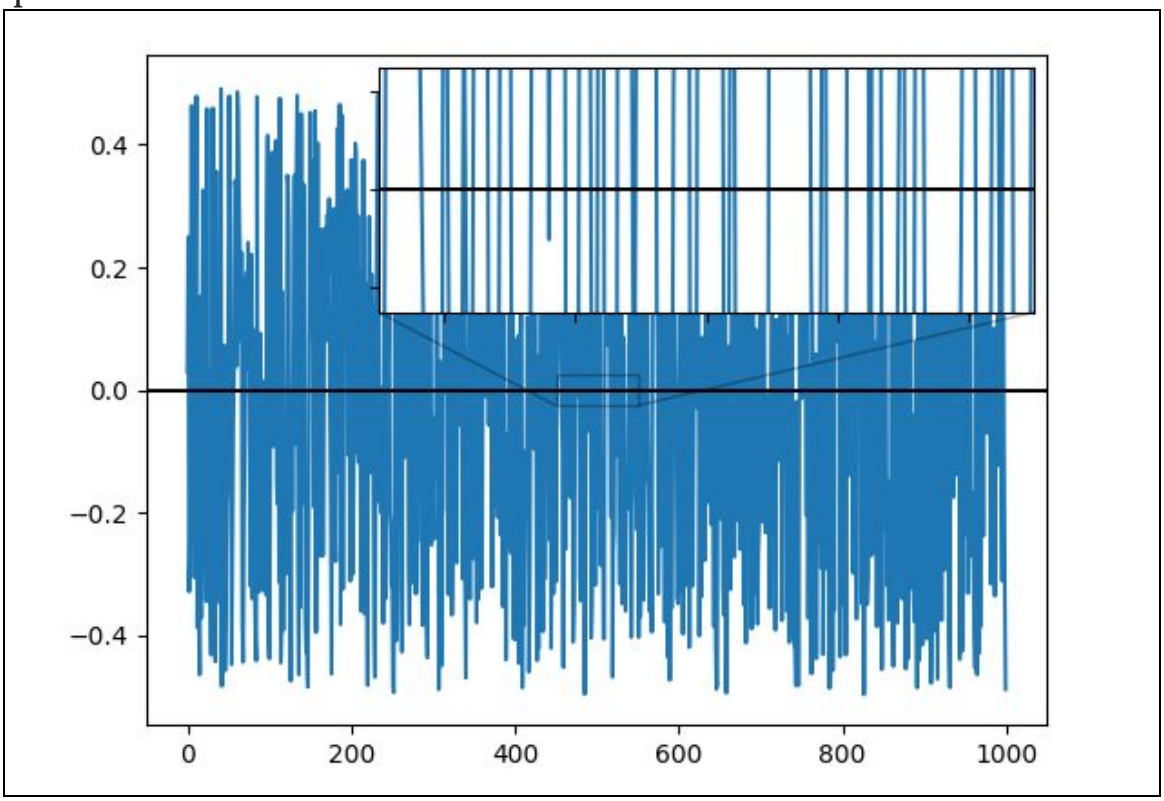
Before:

```
583     rect = TransformedBbox(inset_axes.viewLim, parent_axes.transData)
584
585     pp = BboxPatch(rect, fill=False, **kwargs)
586     parent_axes.add_patch(pp)
587
588     p1 = BboxConnector(inset_axes.bbox, rect, loc1=loc1, **kwargs)
589     inset_axes.add_patch(p1)
590     p1.set_clip_on(False)
591     p2 = BboxConnector(inset_axes.bbox, rect, loc1=loc2, **kwargs)
592     inset_axes.add_patch(p2)
593     p2.set_clip_on(False)
594
595     return pp, p1, p2
```

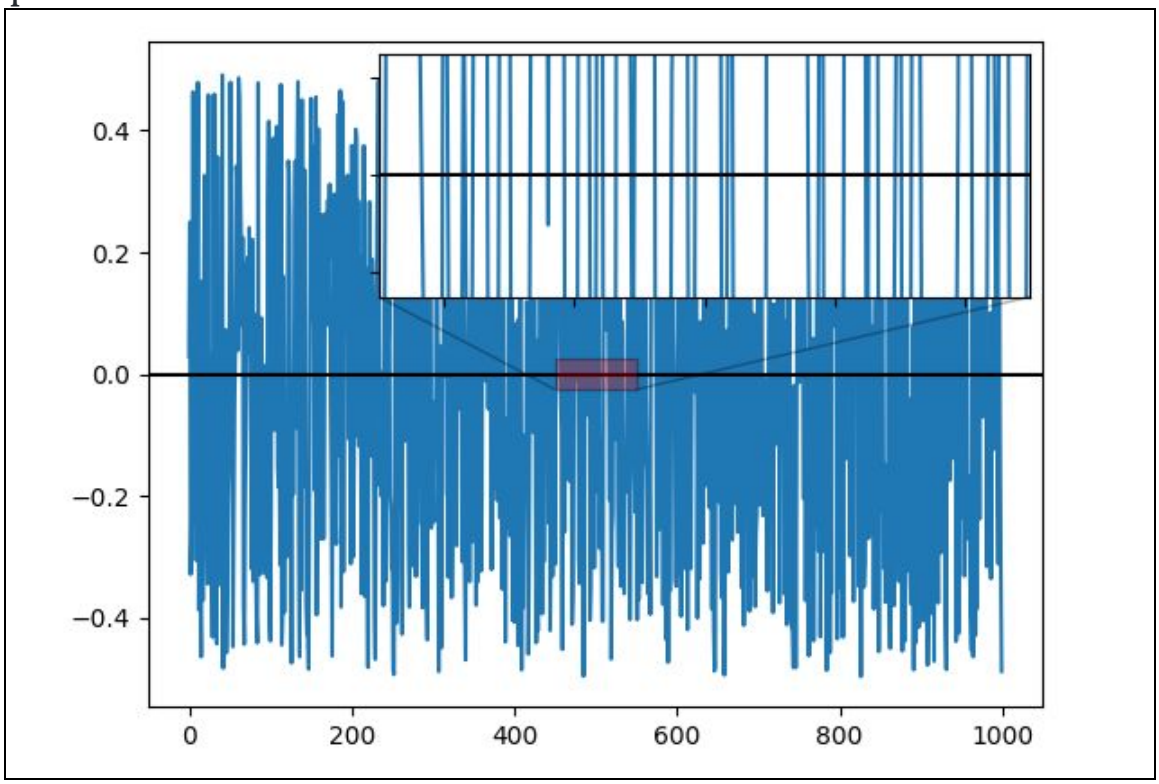
After:

```
592     rect = TransformedBbox(inset_axes.viewLim, parent_axes.transData)\
593
594     kwargs = normalize_kwargs(kwargs, Patches._patch_alias_map)
595     if 'fill' in kwargs:
596         pp = BboxPatch(rect, **kwargs)
597     elif ('facecolor' in kwargs) or ('color' in kwargs):
598         pp = BboxPatch(rect, fill=True, **kwargs)
599     else:
600         pp = BboxPatch(rect, fill=False, **kwargs)
601     parent_axes.add_patch(pp)
602
603     p1 = BboxConnector(inset_axes.bbox, rect, loc1=loc1, **kwargs)
604     inset_axes.add_patch(p1)
605     p1.set_clip_on(False)
606     p2 = BboxConnector(inset_axes.bbox, rect, loc1=loc2, **kwargs)
607     inset_axes.add_patch(p2)
608     p2.set_clip_on(False)
609
610     return pp, p1, p2
```

Output Before:



Output After:



Acceptance Testing:

The acceptance test for this problem involves creating an BboxConnectorPatch:

- Between two subplots
- Between and overlapping the subplots

Each test case will generate a plot, the facecolor of each BboxConnectorPatch can be visually checked. The test case also verifies that facecolor is shown via the line "assert patch.get_fill()" where patch.getfill() will be True if facecolor is shown.

```
def test_face_colour_fill_between_plots():
       fig, ax = pyplot.subplots(1, 2)
       data = ax[0].transData
       data1 = ax[1].transData
       TransformedBbox1 = TransformedBbox(Bbox.from_extents(0, 0.1, 1, 0.2), \
                                          blended_transform_factory(data, data))
       TransformedBbox2 = TransformedBbox(Bbox.from_extents(0, 0.1, 1, 0.2), \
                                          blended_transform_factory(data1, data1))
       patch = BboxConnectorPatch(TransformedBbox1, TransformedBbox2, loc1a=1, \
                                  loc2a=2, loc1b=4, loc2b=3, ec = "r", fc = "r")
       patch.set_clip_on(False)
       ax[0].add_patch(patch)
       pyplot.show()
       assert patch.get_fill()

def test_face_colour_fill_between_and_on_plots():
       fig, ax = pyplot.subplots(1, 2)
       data = ax[0].transData
       data1 = ax[1].transData
       TransformedBbox1 = TransformedBbox(Bbox.from_extents(0.5, 0.1, 0.5, 0.2), \
                                          blended_transform_factory(data, data))
       TransformedBbox2 = TransformedBbox(Bbox.from_extents(0.5, 0.1, 0.5, 0.2), \
                                          blended_transform_factory(data1, data1))
       patch = BboxConnectorPatch(TransformedBbox1, TransformedBbox2, loc1a=4, \
                                  loc2a=3, loc1b=2, loc2b=1, ec = "r", fc = "r")
       patch.set_clip_on(False)
       ax[0].add_patch(patch)
       pyplot.show()
       assert patch.get_fill()
```

In the issue_8059_bug-fix branch go to Deliverable 3/acceptanceTests and open test_BboxConnectorPatch.py.

Bug-Fix for issue 7460

Issue URL: https://github.com/matplotlib/matplotlib/issues/7460

Source of Problem:

Reviewing issue 7640 revealed that the issue demands to raise an error whenever invalid input such as nan, string etc are passed to xlim function. **`pyplot.xlim`** is a function used to find x limit of a plot. Currently, a call to **`pyplot.xlim`** with invalid input show incorrect x limit, which is (-0.001, 0.001). The issue on GitHub further showed a suggestion to using numpy's built in function called **`isFinite(x)`** as check for the parameter of xlim. **`numpy.isFinite(x)`** return true iff only x is finite function.

Files Modified:
matplotlib/lib/matplotlib/axes/_base.py

Screenshots:

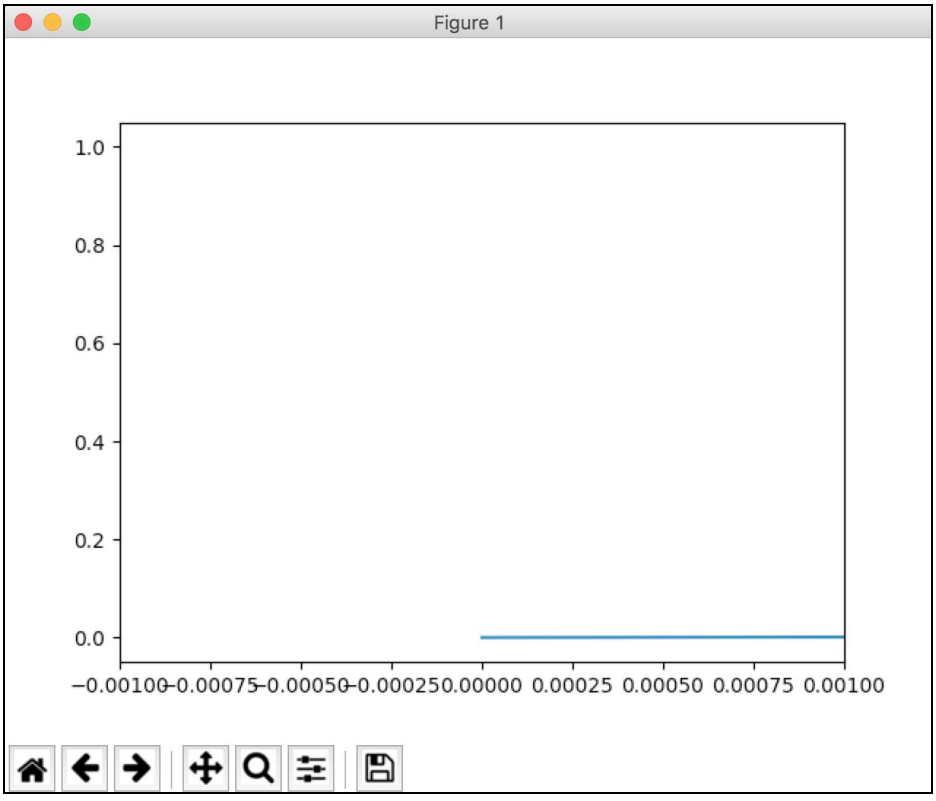
Code Before:

```
def set_xlim(self, left=None, right=None, emit=True, auto=False, **kw):
    """Set the data limits for the x-axis..."""
    if 'xmin' in kw:
        left = kw.pop('xmin')
    if 'xmax' in kw:
        right = kw.pop('xmax')
    if kw:
        raise ValueError("unrecognized kwargs: %s" % list(kw))

    if right is None and iterable(left):
        left, right = left

    self._process_unit_info(xdata=(left, right))
    if left is not None:
        left = self.convert_xunits(left)
    if right is not None:
        right = self.convert_xunits(right)

    old_left, old_right = self.get_xlim()
    if left is None:
        left = old_left
    if right is None:
        right = old_right

    if left == right:
        warnings.warn(
            ('...') % (left, right))
    left, right = mtransforms.nonsingular(left, right, increasing=False)

    if self.get_xscale() == 'log' and (left <= 0.0 or right <= 0.0):...
    left, right = self.xaxis.limit_range_for_scale(left, right)

    self.viewLim.intervalx = (left, right)
    if auto is not None:
        self._autoscaleXon = bool(auto)

    if emit:
```

Code After:

```
    def set_xlim(self, left=None, right=None, emit=True, auto=False, **kw):
        """Set the data limits for the x-axis..."""
        if 'xmin' in kw:
            left = kw.pop('xmin')
        if 'xmax' in kw:
            right = kw.pop('xmax')
        if kw:
            raise ValueError("unrecognized kwargs: %s" % list(kw))

        if right is None and iterable(left):
            left, right = left

        self._process_unit_info(xdata=(left, right))
        if left is not None:
            left = self.convert_xunits(left)
            if not (np.isfinite(left)):
                raise ValueError("%s is invalid input" % (left))
        if right is not None:
            right = self.convert_xunits(right)
            if not (np.isfinite(right)):
                raise ValueError("%s is invalid input" % (right))

        old_left, old_right = self.get_xlim()
        if left is None:
            left = old_left
        if right is None:
            right = old_right

        if left == right:
            warnings.warn(
                ('...') % (left, right))
        left, right = mtransforms.nonsingular(left, right, increasing=False)

        if self.get_xscale() == 'log' and (left <= 0.0 or right <= 0.0):...
        left, right = self.xaxis.limit_range_for_scale(left, right)

        self.viewLim.intervalx = (left, right)
```

Output Before:



Output After (Desired Output):

Acceptance Testing:

To test that the fix is working correctly, you will need to test that it outputs the desired error when an invalid input is given. Such as an infinite, String or nan (not a number). Next, need to test that it still functions correctly when given a valid input.

In the issue_7460_bug-fix branch go to Deliverable 3/acceptanceTests and open tests_xlim_ylim.py

There are sixteen tests you to run by uncommenting the wanted line. The latter eight lines for ylim and xlim could be tested concurrently:

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as pyplot

pyplot.plot([-100, 100])

#Below any of the 8 lines should cause an invalid type error.

#pyplot.xlim(np.nan)
#pyplot.xlim(np.inf)
#pyplot.xlim(np.NINF)
#pyplot.xlim("x is from 0 to 10")

#pyplot.ylim(np.nan)
#pyplot.ylim(np.inf)
#pyplot.ylim(np.NINF)
#pyplot.ylim("x is from 0 to 10")

#Below should work as expected, and show the chart with proper x and y limits.

#pyplot.xlim(-10,10)
#pyplot.xlim([3.3, 4.4])
#pyplot.xlim(np.array([1, 5]))
#pyplot.xlim(None)

#pyplot.ylim(-10,10)
#pyplot.ylim([3.3, 4.4])
#pyplot.ylim(np.array([1, 5]))
#pyplot.ylim(None)

pyplot.show()
```

The first four cases should return an error. The latter eight should produce a graph with the proper x and y limits.

Design

Both of the mentioned bugs do not require significant redesign of the existing code and thus do not have a significant impact on the overall architecture/design of matplotlib. However, we ran into one design decision while fixing bug #8059: the bug fix requires processing alias names for given kwargs stored in **`_patch_alias_map`** for the **`Patch`** class in `patches.py` is needed to fix the bug, but has a private identifier. This brings us to the decision of whether to use the map to simplify our solution to the bug (and thus make use of the private set of a different class, which is typically discouraged), or to have a more complicated and harder to maintain solution and refrain from using the set. We decided to look for other uses of the map in matplotlib to see how the current developers of matplotlib handled this design decision. We noticed a very similar use in matplotlib/axes/_axes.py, Line 1971, and decided that it would make more sense to follow their approach, as it would certainly be a negative to have conflicting solutions to the same problem in matplotlib, and this solution allows for considerably easier maintenance and less potential conflicts if the aliases for pathes.py are updated in future.