CSCD01: Deliverable 2

Team Name: //TODO

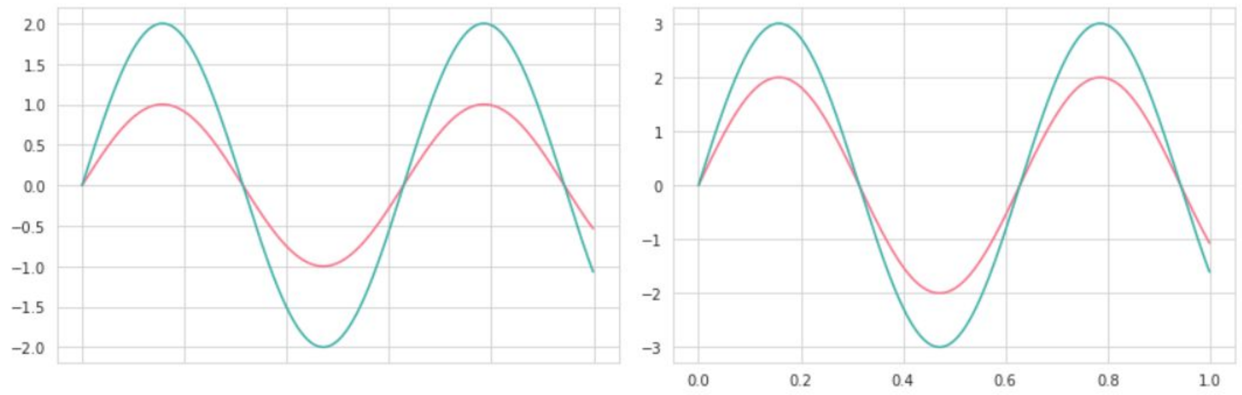
Table of Contents:

Explanation of 5 Issues:

Issue #15225:

The bug being outlined by this issue is the inability to consistently apply x-values, more commonly known as x-ticks, to the x-axis of a graph when it is created. There are some minor changes that can be made to make the x-ticks appear on the graph's x-axis, but these have been shown to only work in certain conditions. One such fix is to explicitly code that the bottom label of the graph should be displayed upon creation. However, even this fix can be overridden and the x-ticks are not displayed when the limit on the x-axis values becomes too large.

Image of Bug:



In the thread of comments on the issue, it is discussed that this issue may actually have to deal with the text associated with the x-axis itself and how the offset text for the axis is not being displayed. This issue is specifically found to be in the matplotlib/lib/matplotlib/axis.py file which directly deals with offsetText as one of its own parameters - this is affected when the user manually overrides "labelbottom" or "labeltop" to display x-ticks through the set_tick_params() function.

A potential solution is to let the default case to be not to show any text at all because the programmers will never be sure if the user wants to actually display that information or not; but this can also be split into specific cases: checking if 'labeltop' and 'labelbottom' are both false, then the ticks can be invisible. Otherwise if only one or neither is false, the x-ticks may still want to be visible - the user should get to make a decision of whether or not to display the information instead of outright defaulting to no text. As this issue is associated with the axis.py file in the project, this fix could be relatively simple as discussed in the comments by the other developers. A simple case-by-case evaluation presents itself as a logical solution, but the fact that no developers know what the expected functionality behind this should be is an issue. If there are no other unexpected challenges, this issue should take about a day - an hour or two to implement the case statements and a few more to thoroughly create tests to ensure the feature works as intended.

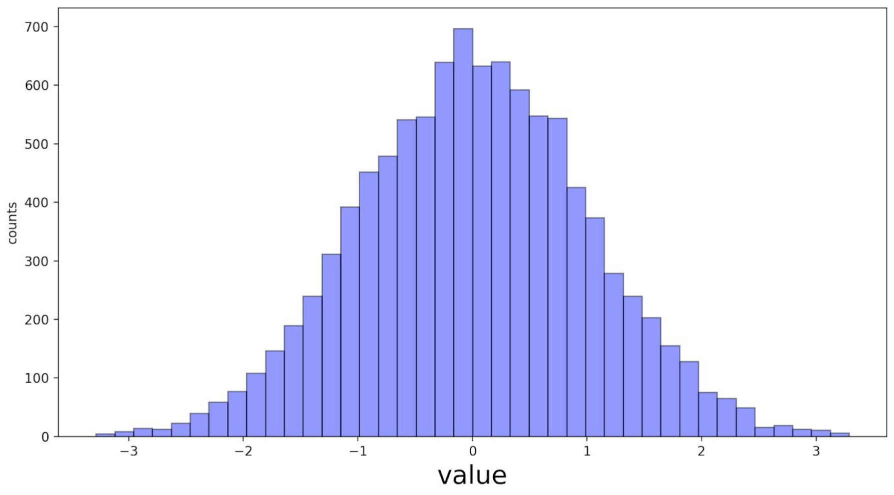
Issue #16389:

The issue described in this bug report is that if size is placed before fontproperties in the parameters of plt.xlabel or plt.ylabel, size gets ignored and the size of the label is thus left as the default size, as seen for the y-axis label in the image below. Furthermore, the bug report indicates that in as.text, ax.set_title and any other function related to showing characters on the figure, size is ignored if it is placed before fontproperties.

Code to Reproduce Bug:

```
import matplotlib.pyplot as plt
import numpy as np
data = np.random.randn(10000)
plt.hist(data, bins=40, facecolor=”blue”, edgecolor=”black”, alpha=0.5)
plt.xlabel(“value”, fontproperties=’SimHei’, size=20) # this will work
plt.ylabel(“counts”, size=20, fontproperties=’SimHei’) # this won’t work
plt.show()
```

Image of Bug:



Based on investigation, it seems that even though Text() holds fontproperties and size is handled via **kwargs, the method update() in Artist gets called with both values. Parameter order matters in update() since the parameters are processed sequentially, which is why having size before fontproperties becomes an issue here, even though parameter order isn’t supposed to matter. If fontproperties is used with a string, a new FontProperties instance is created with default values, which causes the given size value to be overwritten with a default value since size isn’t set within fontproperties. Therefore, the size value resets to default in the code for plt.ylabel because fontproperties is called after size in the update() method.

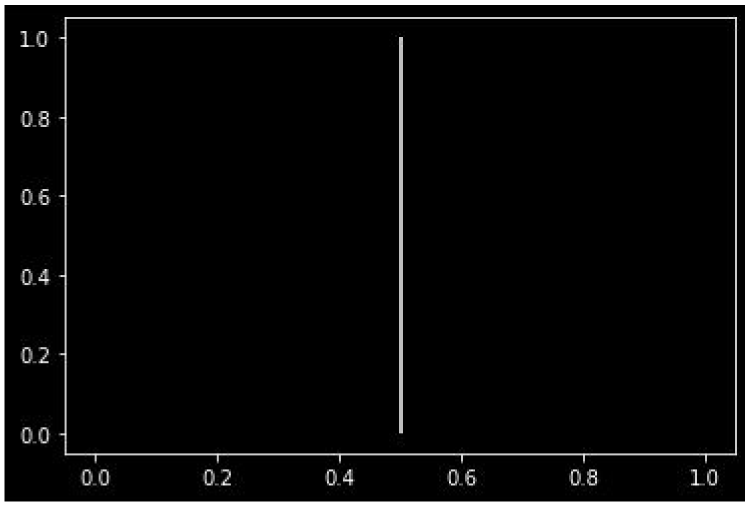
A possible solution for this bug is to ensure that fontproperties is called before size within the Artist update() method, so that size doesn't get overwritten by default values. To do this, we'd have to pop fontproperties from the props dictionary object first to make sure fontproperties is run before any other argument. By doing this, we'd ensure that fontproperties values won't overwrite any other values, which would solve the issue described above.

We believe this issue will take a day to complete thanks to one of the contributors assisting by locating exactly where the bug is in the code and by commenting that the bug is related to the order of the arguments being called in the Artist.update() function. This saves us time for investigation since we know what method to go to fix the issue. Therefore, one day will mainly be spent figuring out how to implement a solution and writing test cases.

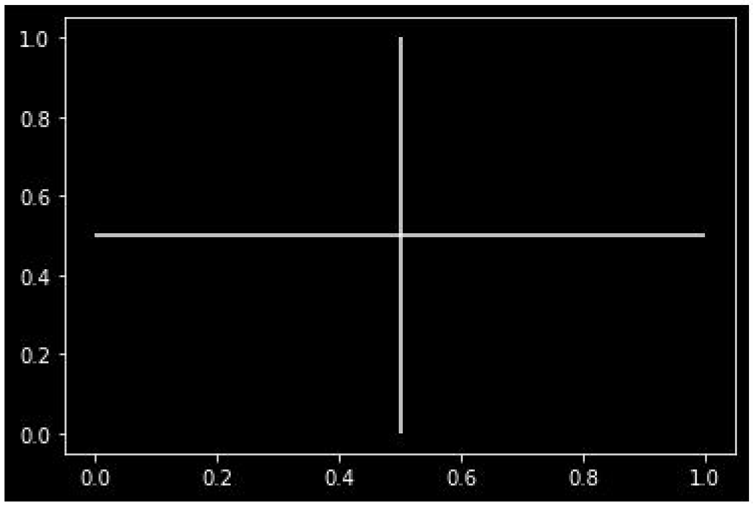
Issue #16482:

The hlines and vlines methods in the pyplot module do not use the lines.colour property stored in rcParams as the default value. Currently, the default value for colors in both the methods is black. This is a problem because not only is this inconsistent with the rest of the API but it also does not work as expected with black backgrounds. The two methods display the lines in black which cannot be seen when the background colour is also black. The methods work as expected (i.e. line color is set to the value in rcParams['lines.color']) when the value for the optional parameter 'colors' is set to None. However, when there is no value passed in for colors, the lines are displayed in the default colour black, even if the background colour is also black.

Actual outcome:



Expected outcome:



Through investigation we determined the following: first, the hlines method in pyplot is invoked. This method calls pyplot's gca method to get the current Axes instance on the current figure. The gca method calls pyplot's gcf method to get the current Figure instance and then calls the gca method in _axes.py to get the current Axes instance on the figure. Once the Axes object is obtained, pyplot's hlines method calls the hlines method in _axes.py passing all the arguments without modification. This method calls the init method of the LineCollection class in collections.py. The LineCollection class is used to plot multiple lines on the figure.

The main problem is that the default value is being set to 'k' instead of the value in rcParams['lines.color']. Therefore, the fix is modifying the default value for the optional parameter 'colors' to rcParams['lines.color']. This will ensure that the color of the horizontal line is white when the background is black since the value in lines.color is white when the background is black. To fix this issue we changed the default value of colors from 'k' to None. The default value was changed to None because in LineCollection, if 'colors' is None, the line color is set to rcParams['lines.color']. Therefore, assigning the default value of None will actually assign the value in rcParams['lines.color']. The vlines method works similarly. The only difference is that once the Axes object is obtained in pyplot, the vlines method _axes.py is called.

After further investigation it was discovered that the default value for colors must also be changed in hlines and vlines methods from _axes.py (these methods may be used if the user does not use pyplot to generate graphs). Therefore, the fix is in 4 methods: pyplot's hlines and vlines methods and _axes's hlines and vlines methods.

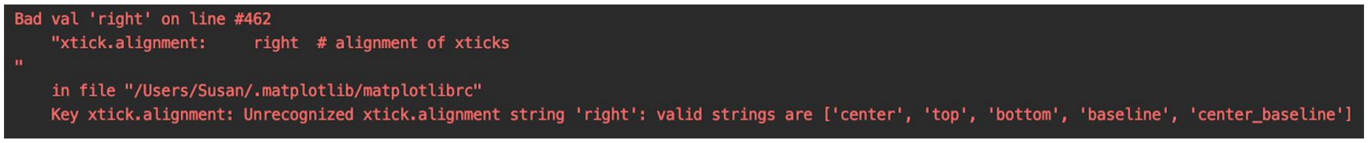
The estimated time to investigate and solve this bug is 2 days. 1 day for investigation and finding the root cause and 1 day for implementing changes, writing test cases and running the regression test suite.

Issue #16583:

In matplotlib, users can customize properties using a file called “matplotlibrc”. Example properties include figure size, line width, color, and axis. This issue details that customizing the value for the property ‘xtick.alignment’ creates errors due to inconsistent parameter checks.

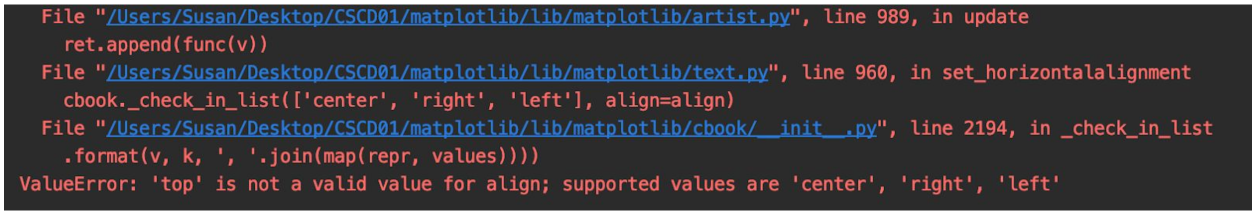
In order to reproduce the bug, the user will first need to find the active matplotlibrc file on their computer. Opening the python3 shell and typing matplotlib.matplotlib_fname() will output the path of the file. This matplotlibrc file will need to be in the matplotlib configuration directory in order to apply the properties. Typing matplotlib.get_configdir() will output the path of the configuration directory. After finding both paths, the user should place the matplotlibrc file into the config directory and comment out ‘xtick.alignment’ for this example.

In matplotlib, the horizontal alignment value can be set to {‘center’, ‘right’, ‘left’} as specified in “set_horizontalalignment”, line 952 in text.py. However, changing matplotlibrc to xtick.alignment: right or xtick.alignment: left will output the errors shown below:



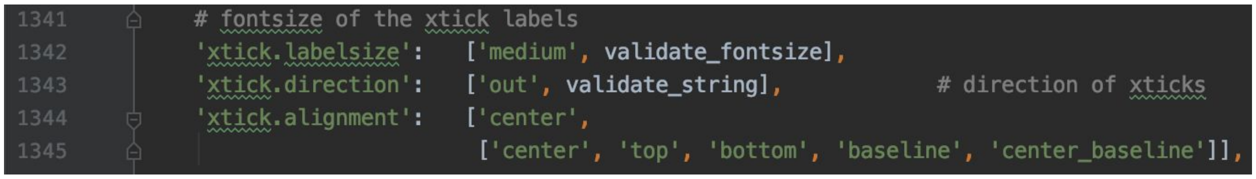
```
Bad val 'right' on line #462
    "xtick.alignment:     right  # alignment of xticks
"
    in file "/Users/Susan/.matplotlib/matplotlibrc"
    Key xtick.alignment: Unrecognized xtick.alignment string 'right': valid strings are ['center', 'top', 'bottom', 'baseline', 'center_baseline']
```

The valid strings the error outputs are different than what was originally defined for horizontal alignments which were {‘center’, ‘right’, ‘left’}. Testing the strings provided in the error message (above) ‘top’, ‘bottom’, ‘baseline’ or ‘center_baseline’ for xtick.alignment will result in a ValueError.



```
  File "/Users/Susan/Desktop/CSCD01/matplotlib/lib/matplotlib/artist.py", line 989, in update
    ret.append(func(v))
  File "/Users/Susan/Desktop/CSCD01/matplotlib/lib/matplotlib/text.py", line 960, in set_horizontalalignment
    cbook._check_in_list(['center', 'right', 'left'], align=align)
  File "/Users/Susan/Desktop/CSCD01/matplotlib/lib/matplotlib/cbook/__init__.py", line 2194, in _check_in_list
    .format(v, k, ', '.join(map(repr, values))))
ValueError: 'top' is not a valid value for align; supported values are 'center', 'right', 'left'
```

After examination, we can determine that the class ValidateStrings in rcsetup.py first checks the parameters in matplotlibrc to validate whether they are valid rc parameters. In set_horizontalalignment, it then calls cbook._check_in_list([‘center’, ‘right’, ‘left’], align=align) to check whether the parameter matches one of those listed. Setting up rc parameters is done in rcsetup.py. In line 1344 it defines the parameters for xtick.alignment.



```
1341        # fontsize of the xtick labels
1342        'xtick.labelsize':    ['medium', validate_fontsize],
1343        'xtick.direction':    ['out', validate_string],          # direction of xticks
1344        'xtick.alignment':    ['center',
1345                               ['center', 'top', 'bottom', 'baseline', 'center_baseline']],
```

This does not match the values specified in set_horizontalalignment. This is the reason the xtick.alignment values ‘top’, ‘bottom’, ‘baseline’ and ‘center_baseline’ gave no errors of unrecognized xtick.alignment strings in rcsetup.py, but got a ValueError in cbook.__init__.py line 2181 as it did not contain any of ‘center’, ‘right’, or ‘left’. Changing the value for ‘xtick.alignment’ to [‘center’, [‘center’, ‘right’, ‘left’]] in rcsetup.py line 1345 is a possible solution. Only a day is needed to implement and fully test this change.

Issue #14233:

This issue is a new feature request. In the rcParams global object, users can choose to display minor ticks on their graph axes; however, they cannot give a default value for the number minor ticks shown between each pair of major ticks. Instead the default values of 3 or 4 are used (either 3 or 4 will be used depending on the locations of the major ticks). User getup8 requested that users have the option of setting a default value for the number of minor ticks in rcParams.

This new feature deals with three files: rcsetup.py, matplotlibrc.template and ticker.py. rcsetup.py is used for storing the default values and validation code for customization using matplotlib’s rc settings. Each rc setting has a default value and a validation function that tests any attempted changes to the setting. The rcsetup file is also used to create and manipulate the rcParams global object. The rcParams object stores the setting values and is used throughout the matplotlib code. In order to add this feature two parameters need to be added to this file. The parameters should be given specific names indicating their purpose such as: xtick.minor.ndivs and ytick.minor.ndivs. New validate functions also need to be added. The purpose of the new validate functions is to ensure that any value given to the new parameters is of an acceptable type and a valid value. As discussed in the issue comments, acceptable values are non-negative ints and the string ‘auto’. If the rcParams is not set by the user it will use the default value ‘auto’. These changes need to be reflected in the matplotlibrc.template file found in the root directory on github; since the template file reflects the values given in rcsetup.py and must be kept consistent. In the template file two new entries need to be added under the TICKS section header indicating the name of the new parameters, their default values, and their purpose.

In the ticker.py file we find the classes and methods used to configure tick locating and formatting. In order to complete this feature we need to update AutoMinorLocator.__init__ to use our new parameters. The AutoMinorLocator class is used to determine the location of minor ticks when the axis is linear and the major ticks are evenly spaced. It contains a ndivs value that it uses to determine how many minor ticks should be placed between each pair of major ticks. The ndiv value represents the number of intervals created by the minor ticks. For example, if ndiv = 2 then there will be one minor tick between each pair of major ticks creating 2 intervals. Currently, AutoMinorLocator takes in two parameters: self, and n=None. If n is given then ndivs=n. If no value is given then ndivs is set to either 4 or 5 (creating either 3 or 4 minor ticks) depending on the location of the major ticks. To add the new feature we want the AutoMinorLocator to accept the following values for n: ‘auto’; a non-negative int; and None. The logic of the functionality should be: if n=’auto’ then use the current method where ndivs=4 or 5; if n=[some non-negative int] then ndivs=[the given int]; if n=None then use the rcParams value. Barring unexpected challenges or complications this issue should take approximately one day of work to implement and test.

The Two Issues Selected

The two issues we chose to implement are issues #16482 and #14233. Issue #16482 deals with the hlines and vlines method not using the lines.colour property stored in rcParams as the default value. Issue #14233 is a new feature where users will have the option to set a default value for the number of minor ticks in rcParams. Out of the five issues, these two were the best to implement because we had a clear idea of how to solve both issues and based on our estimates, they could be fully implemented and tested prior to the Deliverable 2 due date. The estimated time for each of the two issues was 1-2 days, which we thought was perfect for this deliverable. Additionally, both issues seemed interesting and non-trivial. We wanted to implement issues that would provide us with the opportunity to further explore the code base and familiarize ourselves with the contribution process. Knowing these issues required thorough investigation and touched multiple files while still being manageable within the time constraints, we were confident in our choices that these two issues would be perfect to implement.

To solve issue #16482, we have to modify the default value for the colors parameter. Originally the default value was ‘k’ i.e. black, but it must be changed to the value in lines.color from rcParams. This change must be made in 4 places: pyplot’s hlines and vlines methods and _axes.py’s hlines and vlines methods. The anticipated risk for the implementation of issue #16482 is the following. Since the default color has been changed from ‘k’ to rcParams[‘lines.color’] the user may assume that the functionality is not working as expected if they are not aware of this change. For example, originally, when the background was white, the color of the horizontal and vertical lines was black when no value was provided to ‘colors’. However, after implementing the fix, when the background is white, the color of the lines will be blue since rcParams[‘lines.color’] has the value ‘C0’ i.e. blue. Another risk was the existing test cases failing because of the modified code. To ensure that this does not happen, we ran the entire regression test suite with the modified code.

For issue #14233 we need to update the AutoMinorLocator.__call__ method to use the rcParams settings when no parameter is provided. Additional functionality also needs to be added to determine the behaviour of the AutoMinorLocator class depending on the given or stored value. For example, if the parameter given is ‘auto’ the current method of determining the number and locations of minor ticks would be used (resulting in 3-4 minor ticks per pair of major ticks). If no parameter is given the ‘__call__’ method will retrieve the values stored in rcParams (which may be ‘auto’ or a non-negative int). Lastly, if a non-negative integer is specified it will be used by the method. In addition to the reasons listed above we chose this issue because we were intrigued by the idea of implementing a new feature. The anticipated risk for the implementation of issue #14233 is that we are changing the public API by adding new parameters to the rcParams object. It is important that we carefully follow the coding guidelines provided by the matplotlib developers for editing the API and the rcParams settings in order to avoid breaking any of the existing functionality. Furthermore, by changing the functionality of the AutoMinorLocator class we need to check that we have not caused any of the existing tests for this class to fail. In short, our code must add the new feature without damaging any of the existing features.

Acceptance Test Suite

Instructions: Fork the repo https://github.com/CSCD01-team23/matplotlib and build matplotlib in a virtual environment as described in https://matplotlib.org/3.1.1/devel/contributing.html#installing-for-devs. The implementation for both issues are in their respective branches. Checkout branch 'bugfix-for-issue-16482' for testing issue #16482. Checkout branch '14233-sol' for testing issue #14233. Make sure to rebuild matplotlib before testing issue #14233.

Issue #16482

User Story: As a matplotlib user, I want to be able to plot vlines and hlines in different background styles so that I can see these lines.

ID	GIVEN	WHEN	THEN
01	-	The user wants to plot hline and vline without specifying 'colors' in the style 'dark_background'	The hline and vline will be drawn in white
02	-	The user wants to plot hline and vline in a different color ('C0') in the style 'dark_background'	The hline and vline will be drawn in 'C0' (blue)
03	There are no styles	The user wants to plot hline and vline without specifying 'colors'	The hline and vline will be drawn in 'C0'
04	There are no styles	The user wants to plot hline and vline in a different color ('k')	The hline and vline will be drawn in 'k' (black)

ID 01: Run this code in python3

```
import matplotlib.pyplot as plt
plt.style.use('dark_background')
plt.figure()
plt.hlines(0.5, 0, 1)
plt.vlines(0.5, 0, 1)
plt.show()
```

ID 02: Same code as ID 01 but change hlines and vlines to this

```
plt.hlines(0.5, 0, 1, colors='C0')
plt.vlines(0.5, 0, 1, colors='C0')
```

ID 03: Run this code in python3

```
import matplotlib.pyplot as plt
plt.figure()
plt.hlines(0.5, 0, 1)
plt.vlines(0.5, 0, 1)
plt.show()
```

ID 04: Same code as ID 03 but change hlines and vlines to this

```
plt.hlines(0.5, 0, 1, colors='k')
plt.vlines(0.5, 0, 1, colors='k')
```

Issue #14233

User Story: As a matplotlib user, I want the option to change the number of minor ticks between each pair of major ticks so that I have more flexibility than the default (3-4) ticks.

ID	WHEN	THEN
01	The user wants matplotlib to automatically determine the best number of minor ticks between each pair of major ticks for all plots.	matplotlib will choose to display either 3 or 4 minor ticks between major ticks depending on the locations of the major ticks.
02	The user wants matplotlib to default to displaying a specified number of minor ticks between each pair of major ticks.	matplotlib will display the number of minor ticks specified in rcParams
03	The user wants matplotlib to display a specified number of minor ticks between each pair of major ticks, but only for this plot.	matplotlib will display the number of minor ticks specified through the AutoMinorLocator class.
04	The user wants matplotlib to automatically determine the best number of minor ticks to display between each pair of major ticks, but only for this plot. For other plots the user wants to display a specified number of minor ticks.	matplotlib will choose to display either 3 or 4 minor ticks between major ticks depending on the locations of the major ticks for this plot, and will default to displaying the number of minor ticks specified in rcParams on other plots.

ID 01: Run this code in python3

```
import matplotlib.pyplot as plt
import matplotlib.ticker as tick
fig, ax = plt.subplots()
ax.plot()

ax.xaxis.set_minor_locator(tick.AutoMinorLocator())
ax.yaxis.set_minor_locator(tick.AutoMinorLocator())
plt.show()
```

ID 02: Run this code in python3

```
import matplotlib.pyplot as plt
import matplotlib.ticker as tick

plt.rc_context(rc={'xtick.minor.ndivs': 2})
plt.rc_context(rc={'ytick.minor.ndivs': 2})

fig, ax = plt.subplots()
ax.plot()

ax.xaxis.set_minor_locator(tick.AutoMinorLocator())
ax.yaxis.set_minor_locator(tick.AutoMinorLocator())
plt.show()
```

ID 03: Run this code in python3

```
import matplotlib.pyplot as plt
import matplotlib.ticker as tick
fig, ax = plt.subplots()
ax.plot()

ax.xaxis.set_minor_locator(tick.AutoMinorLocator(2))
ax.yaxis.set_minor_locator(tick.AutoMinorLocator(2))
plt.show()
```

ID 04: Run this code in python3

```
import matplotlib.pyplot as plt
import matplotlib.ticker as tick

plt.rc_context(rc={'xtick.minor.ndivs': 2})
plt.rc_context(rc={'ytick.minor.ndivs': 2})

fig, ax = plt.subplots()
ax.plot()

ax.xaxis.set_minor_locator(tick.AutoMinorLocator('auto'))
ax.yaxis.set_minor_locator(tick.AutoMinorLocator('auto'))
plt.show()
```

Brief Commentary

Issue #16482

Design changes

The only design change for this issue is the change in default value. Previously, the default value for the ‘colors’ parameter is the hlines and vlines methods in pyplot and _axes.py was ‘k’. After the fix, the default value for the ‘colors’ parameter in these methods will be the value in rcParams[‘lines.color’].

Files Changed

Code changes:

In pyplot.py: def hlines() in line 2496 and def vlines() in line 2867.
In _axes.py: def hlines() in line 1078 and def vlines() in line 1154.
All of the colors=’k’ have been changed to colors=None.

Test cases:

In test_pyplot.py: Added 4 methods called test_lines_dark_background_default, test_lines_dark_background, test_lines_white_background_default, and test_lines_white_background. These methods use assertion to check the color of the lines drawn using hlines and vlines methods from pyplot.py.

In test_axes.py: Add 2 methods called test_vlines_default and test_hlines_default. These methods use assertion to check the color of the lines drawn using hlines and vlines methods from _axes.py.

Documentation:

Added two sections for the methods changed in pyplot.py and _axes.py to *matplotlib/doc/api/next_api_changes/behaviour.rst* explaining that the default color is now set to None which causes the line color to be the color in rcParams[‘lines.color’].

Issue #14233

Design Changes

There are two main design changes made in the implementation for this feature. First, two new parameters were added to the rcParams function: xtick.minor.ndivs and ytick.minor.ndivs. These were accompanied by their own validator function. Since they accept both the string ‘auto’ and any non-negative int as valid values, the pre-existing general validator functions did not suffice and a custom validator function (validate_int_or_auto(n)) was added. This design change is reflected in the matplotlibrc.template file. Additionally, the functionality of the AutoMinorLocator class in the ticker.py file was changed. Previously, the AutoMinorLocator class, used to determine the location of minor ticks when the axes scale is linear and the major ticks are evenly separated, accepted only int values or no value for its parameter. The AutoMinorLocator class would create the given number of intervals by placing n-1 minor ticks in between each pair of major ticks, where n is the number given. When no value was given it would display 3-4 minor ticks between each pair of major ticks depending on the location of the major ticks. The new functionality accepts no value, the string ‘auto’, or non-negative ints for its parameter. Furthermore, the functionality of the class changes based on the parameter type. If the parameter is None (no value was given) the class’ __call__ method will retrieve the rcParam values stored in xtick.minor.ndivs and ytick.minor.ndivs and use them for determining the number and locations of the minor ticks on the x and y-axis respectively. If the parameter value is ‘auto’ the original functionality will be used (3-4 ticks based on major tick locations). If the parameter value is a non-negative int the plot also follows the original functionality by creating the given number of intervals for each pair of major ticks (displaying n - 1 minor ticks for each pair of major ticks, where n is the given number).

Files Changed

Code changes:

In rcsetup.py: Added a new function validate_int_or_auto(n) (lines 794-807) and added 2 new parameters ‘xtick.minor.ndivs’ and ‘ytick.minor.ndivs’ (lines 1354 and 1378 respectively) under # tick properties (line 1337).

In matplotlibrc.template: Under the TICKS header (line 441) added 2 keys ‘xtick.minor.ndivs’ and ‘ytick.minor.ndivs’ (lines 463 and 484 respectively) both set with the default value of ‘auto’.

In ticker.py: Added new case statements in the ‘__call__’ method of the AutoMinorLocator class (lines 2765-2797).

Test Cases:

In test_ticker.py four new test cases were added to the TestAutoMinorLocator class (lines 197-259): test_ndivs_rcParams_auto; test_ndivs_rcParams_int; test_ndivs_auto; test_ndivs_int. These test cases check that the new functionality works as expected by testing each parameter type.

In test_rcparams.py a new validator test was added (line 391-400) for the validate_int_or_auto(n) function. This test uses the validator function to ensure that attempts to change the rcParams settings xtick.minor.ndivs and ytick.minor.ndivs will only succeed on valid values and will raise the appropriate error if an invalid value is given.

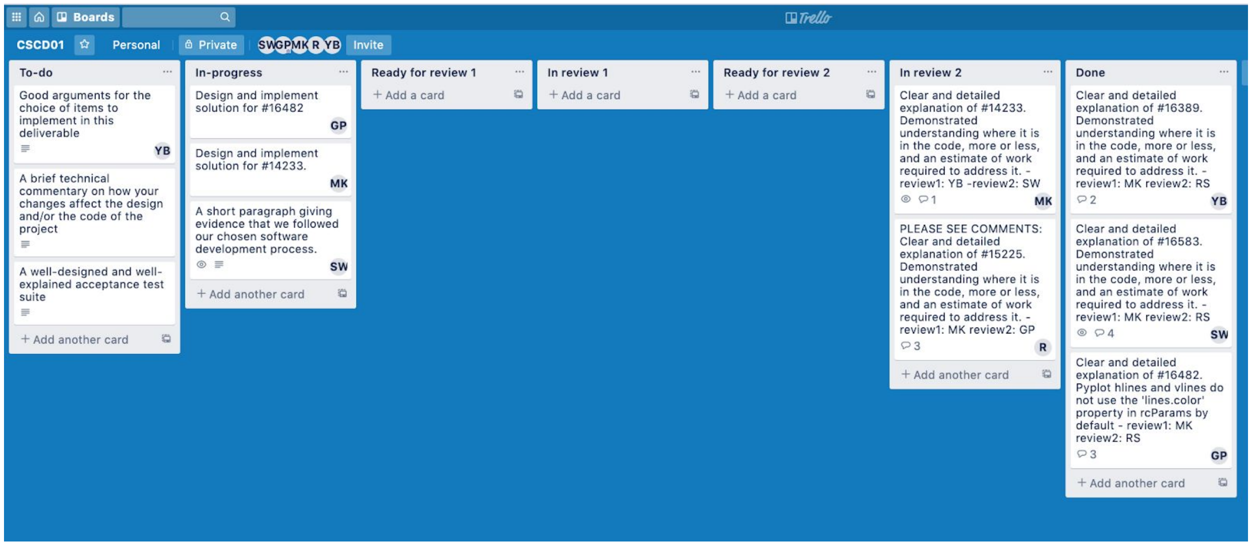
Documentation:

The AutoMinorLocator class docstring was updated to reflect the new functionality. In-line comments were added next to the new parameters in rcsetup.py to explain their function.

Software Development Process

As mentioned in deliverable 1, our team chose the Kanban development process. We have been actively using Trello as our Kanban board and this is the invite link: https://trello.com/invite/b/MP33R5lS/e426d41f8d3e392b40a85792ff93bafc/cscd01

A view of the Kanban board during a particular time can be seen below.



To-do

- Good arguments for the choice of items to implement in this deliverable — YB
- A brief technical commentary on how your changes affect the design and/or the code of the project
- A well-designed and well-explained acceptance test suite
- + Add another card

In-progress

- Design and implement solution for #16482 — GP
- Design and implement solution for #14233. — MK
- A short paragraph giving evidence that we followed our chosen software development process. — SW
- + Add another card

Ready for review 1

- + Add a card

In review 1

- + Add a card

Ready for review 2

- + Add a card

In review 2

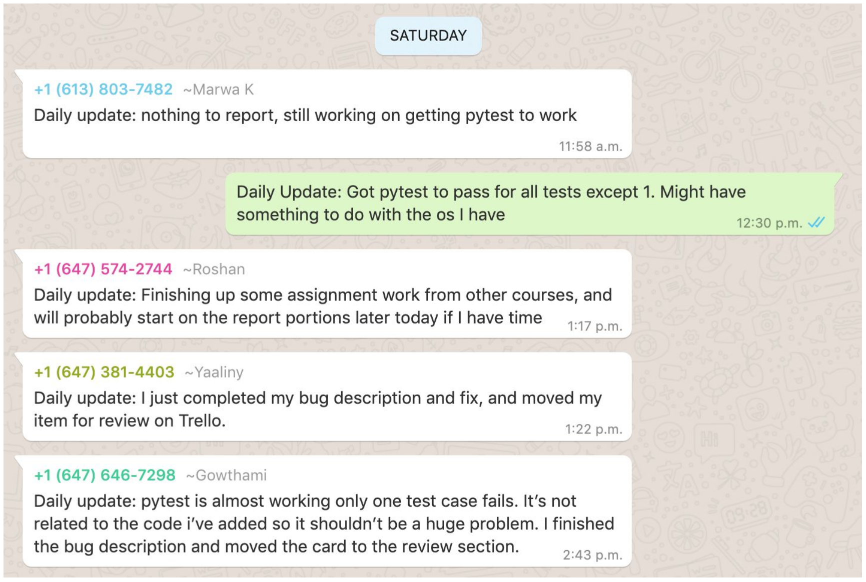
- Clear and detailed explanation of #14233. Demonstrated understanding where it is in the code, more or less, and an estimate of work required to address it. -review1: YB -review2: SW — 1 — MK
- PLEASE SEE COMMENTS: Clear and detailed explanation of #15225. Demonstrated understanding where it is in the code, more or less, and an estimate of work required to address it. -review1: MK review2: GP — 3 — R
- + Add another card

Done

- Clear and detailed explanation of #16389. Demonstrated understanding where it is in the code, more or less, and an estimate of work required to address it. -review1: MK review2: RS — 2 — YB
- Clear and detailed explanation of #16583. Demonstrated understanding where it is in the code, more or less, and an estimate of work required to address it. -review1: MK review2: RS — 4 — SW
- Clear and detailed explanation of #16482. Pyplot hlines and vlines do not use the 'lines.color' property in rcParams by default - review1: MK review2: RS — 3 — GP
- + Add another card

Our Kanban board contains seven columns that consist of To-do, In-progress, Ready for review 1, In review 1, Ready for review 2, In review 2 and Done. Items beneath the columns are assigned to a team member, as seen by the initials on the bottom right corner of each item, and when worked on, move to the In-progress column. Once they are completed, they move from the In-progress column to the Ready for review 1 column. As mentioned before, we require two team members to review a task before the task can be moved to the 'Done' column. The reviewers will write their initials in the title of the card they are reviewing and leave a comment on the card addressing issues or offering feedback for the task. The initials of the reviewers can be seen on the board (e.g. -review1: MK). At any given time, only 5 tasks can be present in 'In-progress'.

Daily updates are done asynchronously in WhatsApp in place of daily meetings. Daily updates are sent by each team member, and contain a summary of what they're working on. An example of a daily update is shown in the next page.



SATURDAY

+1 (613) 803-7482 ~Marwa K
Daily update: nothing to report, still working on getting pytest to work
11:58 a.m.

Daily Update: Got pytest to pass for all tests except 1. Might have something to do with the os I have
12:30 p.m.

+1 (647) 574-2744 ~Roshan
Daily update: Finishing up some assignment work from other courses, and will probably start on the report portions later today if I have time
1:17 p.m.

+1 (647) 381-4403 ~Yaaliny
Daily update: I just completed my bug description and fix, and moved my item for review on Trello.
1:22 p.m.

+1 (647) 646-7298 ~Gowthami
Daily update: pytest is almost working only one test case fails. It's not related to the code i've added so it shouldn't be a huge problem. I finished the bug description and moved the card to the review section.
2:43 p.m.

Our software process helped improve team efficiency by maintaining frequent communication and strong organization. The asynchronous daily updates improved communication and kept us up-to-date on what other team members were working on. This ensured that we knew when a team member was struggling and could help them sooner rather than later, keeping the development process moving smoothly and facilitating our teamwork efforts. The Trello board allowed us to visually keep track of our progress and clearly identified what work still needed to be done. This minimized confusion as it kept us organized and on-track. It was easy to see which items were posing challenges as they did not move across the board and got stuck in a single column. This let us know where we needed to put more effort. Overall, our software process kept us up-to-date on the status of the project and our team mates work in a clear and easy manner.