Arctic Sea Ice Extent: Data Visualization
ISTA 131 Hw6, Due 2/29/2024 at 11:59 pm

Introduction. This homework is the second in a three-assignment arc intended to introduce you to dealing with data that is stored on disk in csv format using `pandas`. In this assignment, you will reproduce figures from the web created by the National Snow and Ice Data Center (NSIDC) and Japan's version, the Arctic Data Archive System (ADS). The NSIDC plot is the way they used to do it, better than their current version. The ADS site will be changing soon, so who knows how long that link will work. Plus, they made their plots worse. We'll recreate the old style.

Instructions. Create a module named `hw6.py`. Below is the spec for seven functions. Implement them and upload your module to D2L Assignments.

Testing. Download `hw6_test.py` and auxiliary testing files and put them in the same folder as your `hw6.py` module. Each of the first three functions is worth 20% of your total score. You do not directly receive points for `main`, but it has to work correctly or you will get no points for two of your other functions. You can examine the test module in a text editor to understand better what your code should do. The test module is part of the spec. The test file we will use to grade your program will be different and may uncover failings in your work not evident upon testing with the provided file. Add any necessary tests to make sure your code works in all cases.

Documentation. Your module must contain a header docstring containing your name, your section leader's name, the date, `ISTA 131 Hw6`, and a brief summary of the module. Each function must contain a docstring. Each docstring should include a description of the function's purpose, the name, type, and purpose of each parameter, and the type and meaning of the function's return value.

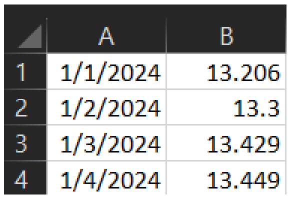
Grading. Your module will be graded on correctness, documentation, and coding style. Code should be clear and concise. You will only lose style points if your code is a real mess. Include inline comments to explain tricky lines and summarize sections of code.

Collaboration. Collaboration is allowed. You are responsible for your learning. Depending too much on others will hurt you on the tests. "Helping" others too much harms them in reality. Cite any sources/collaborators in your header docstring. Leaving this out is dishonest.

Resources.
http://nsidc.org/arcticseaicenews/
https://ads.nipr.ac.jp/vishop/#/extent
http://pandas.pydata.org/pandas-docs/stable/api.html
https://docs.python.org/3/tutorial/datastructures.html
http://matplotlib.org/users/mathtext.html
http://matplotlib.org/api/axes_api.html

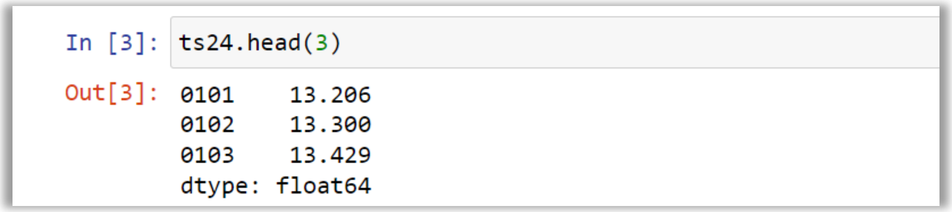
`get_column_labels`: Cut-and-paste this one from hw5, if you want to use it.

`get_2024`: This function reads the data from `data_2024.csv`, which looks like this:



	A	B
1	1/1/2024	13.206
2	1/2/2024	13.3
3	1/3/2024	13.429
4	1/4/2024	13.449

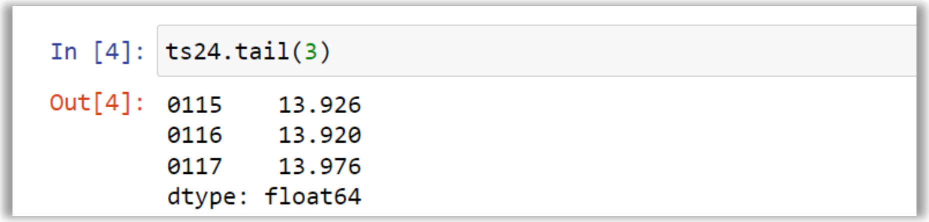
and returns a `Series` that looks like this:



```
In [3]: ts24.head(3)

Out[3]: 0101    13.206
        0102    13.300
        0103    13.429
        dtype: float64
```

⋮

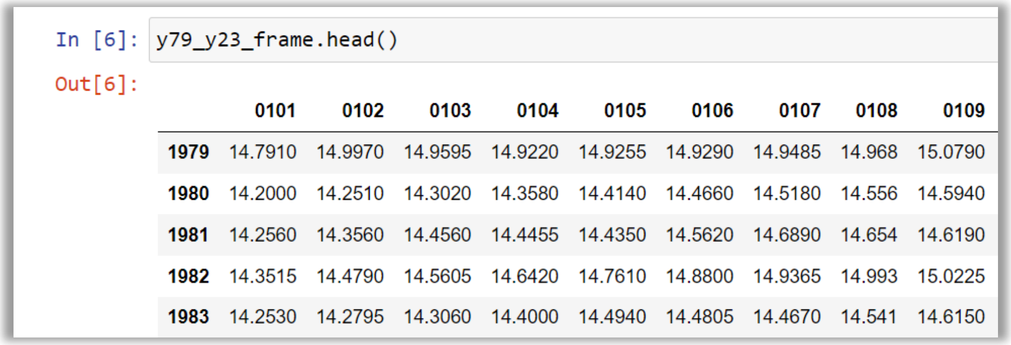


```
In [4]: ts24.tail(3)

Out[4]: 0115    13.926
        0116    13.920
        0117    13.976
        dtype: float64
```

I used `get_column_labels` as the first step in constructing the `index` for my `Series`, but you may do it however you see fit.

`extract_fig_1_frame`: This function takes the `DataFrame` you created in hw5 (which you get how?), which looks like this:

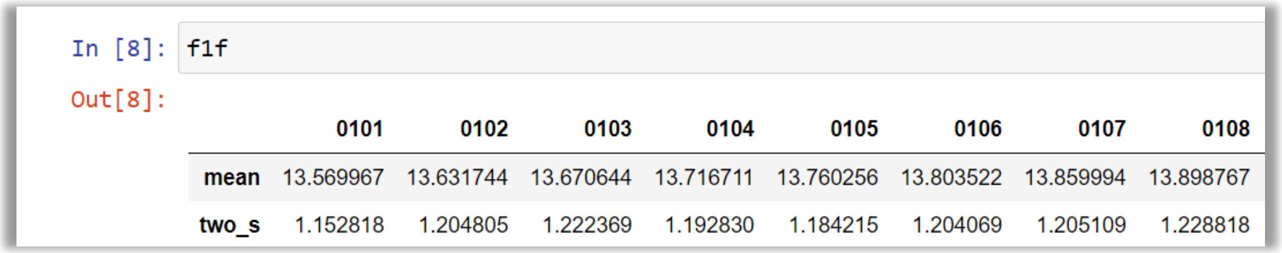


```
In [6]: y79_y23_frame.head()

Out[6]:
```

	0101	0102	0103	0104	0105	0106	0107	0108	0109
1979	14.7910	14.9970	14.9595	14.9220	14.9255	14.9290	14.9485	14.968	15.0790
1980	14.2000	14.2510	14.3020	14.3580	14.4140	14.4660	14.5180	14.556	14.5940
1981	14.2560	14.3560	14.4560	14.4455	14.4350	14.5620	14.6890	14.654	14.6190
1982	14.3515	14.4790	14.5605	14.6420	14.7610	14.8800	14.9365	14.993	15.0225
1983	14.2530	14.2795	14.3060	14.4000	14.4940	14.4805	14.4670	14.541	14.6150

It contains data for each day of the year from 1979 through 2023. Return a `DataFrame` that looks like this:

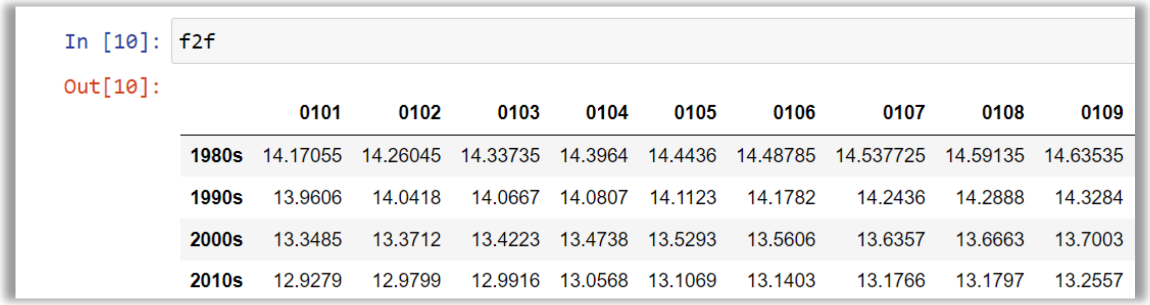


```
In [8]: f1f
Out[8]:
```

	0101	0102	0103	0104	0105	0106	0107	0108
mean	13.569967	13.631744	13.670644	13.716711	13.760256	13.803522	13.859994	13.898767
two_s	1.152818	1.204805	1.222369	1.192830	1.184215	1.204069	1.205109	1.228818

The `mean` row contains the mean of the data in a given column, the `two_s` row contains 2 × the standard deviation of the column. Use the `std` method passing in the keyword argument `ddof=1`.

`extract_fig_2_frame`: This function takes the `DataFrame` you created in hw5 (illustrated at the top of the page) and returns a frame that looks like this:



```
In [10]: f2f
Out[10]:
```

	0101	0102	0103	0104	0105	0106	0107	0108	0109
1980s	14.17055	14.26045	14.33735	14.3964	14.4436	14.48785	14.537725	14.59135	14.63535
1990s	13.9606	14.0418	14.0667	14.0807	14.1123	14.1782	14.2436	14.2888	14.3284
2000s	13.3485	13.3712	13.4223	13.4738	13.5293	13.5606	13.6357	13.6663	13.7003
2010s	12.9279	12.9799	12.9916	13.0568	13.1069	13.1403	13.1766	13.1797	13.2557

The values are the decadal means for the given day of the year. We only have 4 years so far for the 2020's, so we're not doing that decade, yet.

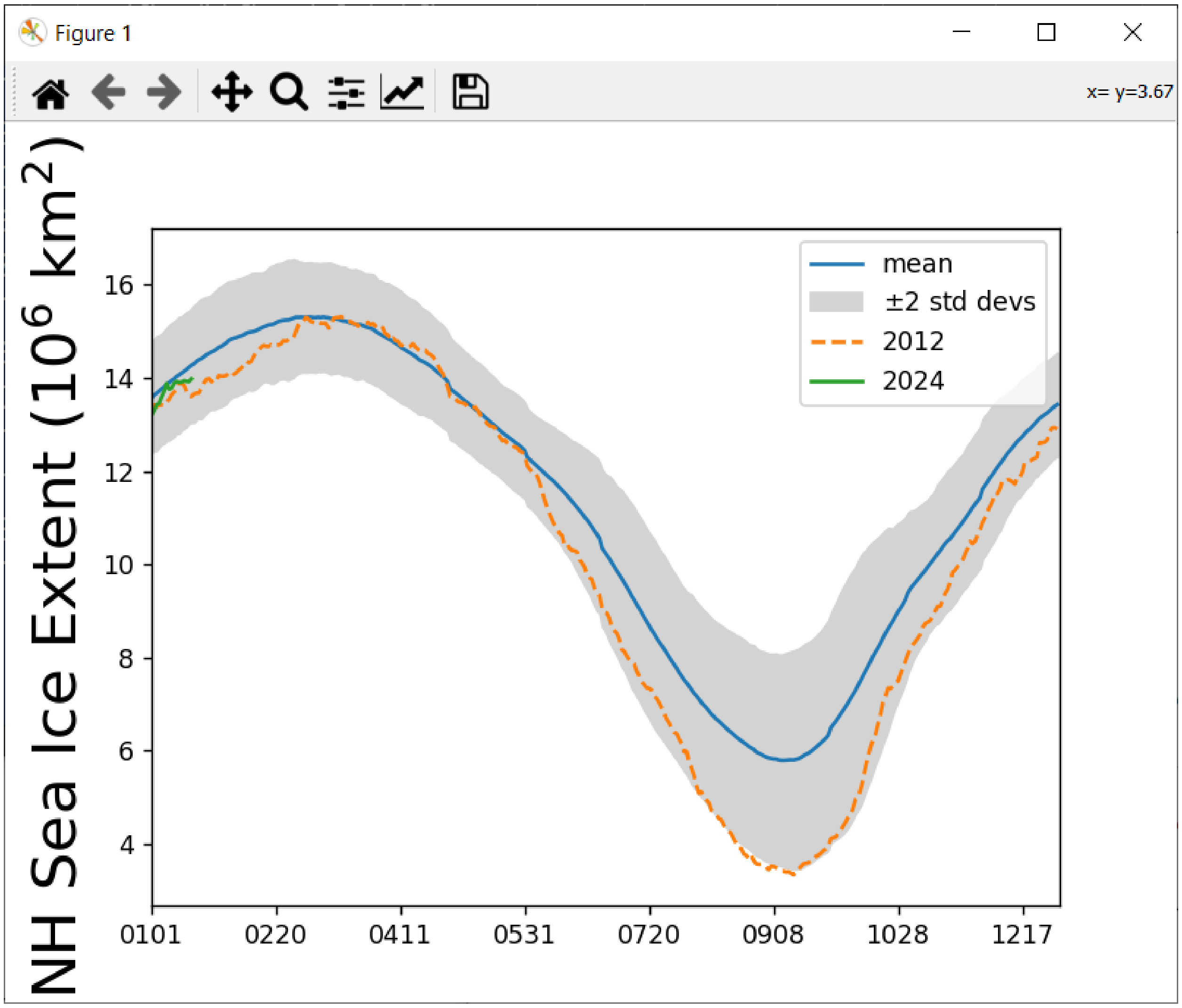
`make_fig_1`: This function takes a figure 1 frame and a hw5 frame, and creates a figure that looks like the image on the following page. The easiest way to match the color scheme is to plot the current year last. But that will screw up the `xticklabels`. Use this code to defeat that problem:

```
ax = plt.gca()
xtl = [tick_label.get_text() for tick_label in ax.get_xticklabels()]
get_2024().plot(label='2024')
ax.set_xticklabels(xtl) # could pass a list of string literals
```

Also, the gray area is the between the `mean + 2 standard deviations` and the `mean − 2 standard deviations`. You will need the `fill_between` function/method (see the sea ice notebook). If necessary, you can fix your x-axis limits with the `pyplot xlim` function.

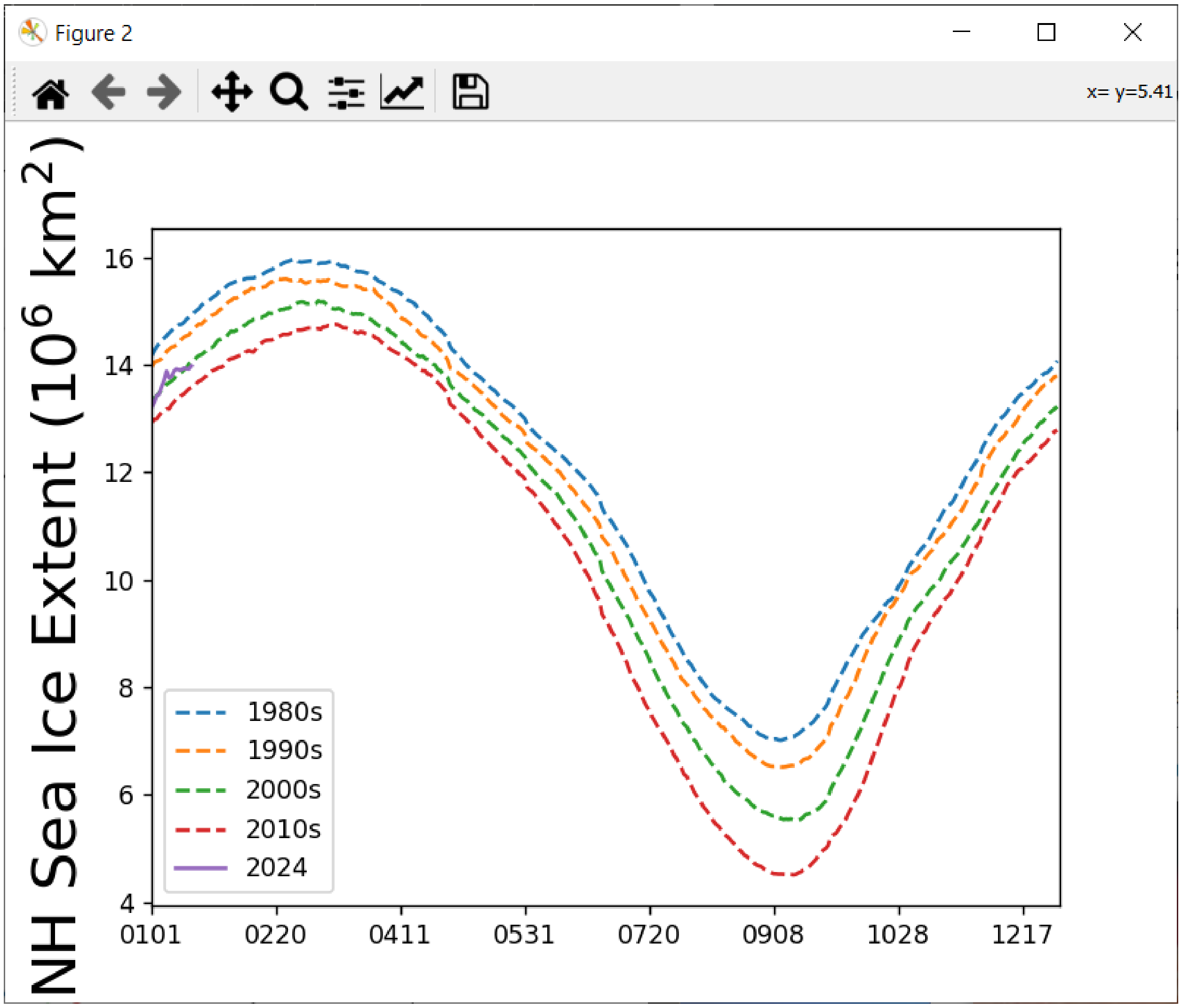
The rubric for this figure:

- +4: the gray area looks like the image. It must not be bordered with visible lines.
- +2 each: the three curves look like the image.
- +4: the y-axis title looks like the image. Superscripts required for any title points.
- +2: correct x-axis tick labels.
- +4: legend looks like the image. Plus/minus symbol required for any legend points.



`make_fig_2`: This function takes a figure 2 frame and creates a figure that looks like the image on the following page. The rubric for this figure:

- +2 each: the five curves look like the image.
- +4: the y-axis title looks like the image. Superscripts required for any title points.
- +2: correct x-axis tick labels.
- +4: legend looks like the image.



main: Read the data in the file `data_79_23.csv` into a frame, recreating the hw5 frame. Make a figure 1 frame and a figure 2 frame. Make the figures (don't forget to call `plt.figure()` in between) and call `plt.show()` (only once at the end). Grading notes:

- -5 figures show up more than once each.
- -2 call `plt.show` twice.