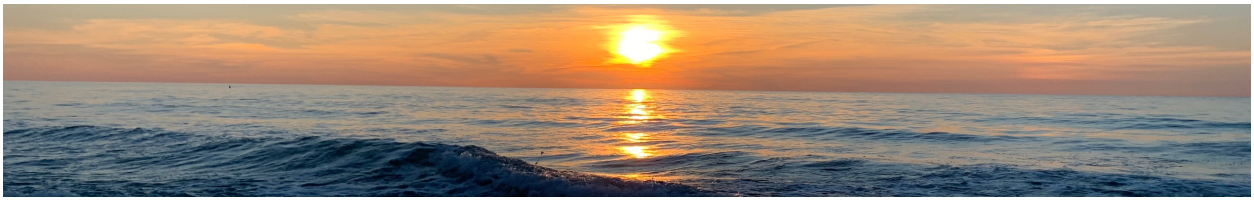


# Investigating sea-surface iodide concentrations throughout the global ocean

*For use by participants in the virtual offering of the Coastal Ocean Environment Summer School in Ghana (COESSING), August 3 – 8, 2020*

**Authors:** Madelyn K. Cook[1], mkcook@umich.edu
Eric Szymanski[1] for generation of python coding exercises
[1]PhD Candidate, Dept. of Earth and Environmental Sciences, University of Michigan, Ann Arbor

**Learning Objectives:**

1. Develop proficiency using helpful Google Sheets functions to import and do small-scale manipulations of a dataset
2. Evaluate available contributions of global sea surface iodide concentrations spatially and temporally using python in Google Colab

The dataset used in this lab exercise was compiled by Dr. Rosie Chance, University of York, and others (Chance et al., 2019). The use, sharing, adaption, and distribution of this dataset is permitted by the creative commons license linked below. No changes were made to alter the dataset from its original form prior to distribution. http://creativecommons.org/licenses/by/4.0/

Chance, R. J., Tinel, L., Sherwen, T., Baker, A. R., Bell, T., Brindle, J., et al. (2019). Global sea-surface iodide observations, 1967-2018. *Scientific Data*, *6*(1), 286. https://doi.org/10.1038/s41597-019-0288-y

## Part I. Tools for small scale dataset manipulation in Google sheets

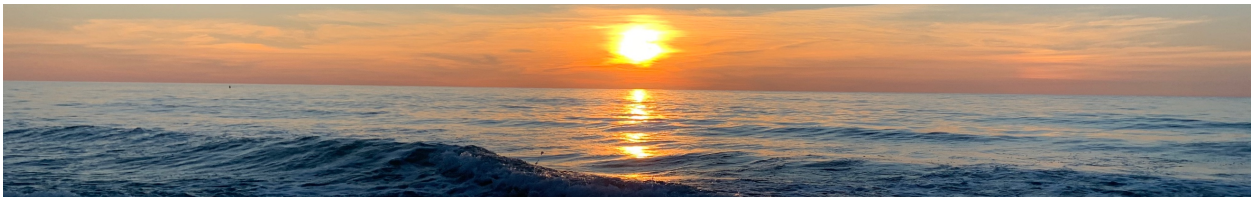
1. Open up google drive
2. New > file upload
3. Select "Global_Iodide_obs_surface.csv"
4. Once the file upload is complete, open with google sheets
5. File > Make available offline

Now that we have the dataset in Google drive, accessible offline, we can employ commonly used commands in Google Sheets (and Microsoft Excel) to manipulate the dataset. Our goal here is to gain some proficiency with logic we can use in sheets before importing it into a coding user interface, for example "Google Colab" or "Jupyter notebook."

### **Goal 1:** Determining the average sea surface iodide for all entries in the dataset

In column I, you will find the reported concentrations (in nM) of sea surface iodide for all entries in the dataset.

1. Using the **average** function, calculate the average for the whole dataset.



*Hint:* every time you write a function into a cell, you must first type "="
You should notice that formula suggestions will be displayed after you type the equals sign.
Example: **=average(A1:A100)** would generate the average of all values in column A, rows 1 to 100

## **Goal 2:** Determine the year during which the most sea surface iodide data was collected

1. Add a sheet to the workbook by clicking on the "+" in the lower left corner

2. Rename the sheet "data_collection_years"

3. In column A, figure out all of the years that data was collected in the dataset using the **unique** function

4. In column B, determine the number of data points in the year displayed in column A using the **countif** function
   *Hint:* the data range you are searching is F2:F1343 found on **Global_Iodide_obs_surface**, be sure to lock this range so you can drag formula down column B on the **data_collection_years** tab. You can lock the range in the formula using the "$" character (e.g. $A$1:$A$100)

5. Select columns A and B and right click, then select the option to "Sort from A to Z." You should now see the years and the corresponding number of data points sorted from earliest to latest.

6. Once you have sorted columns A and B, select them both again and create a "column graph" to visualize the number of observations per year.

7. Which year has the greatest number of datapoints?

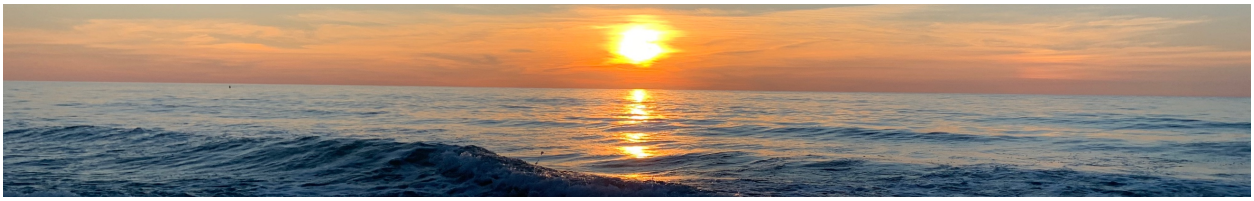
8. Which decade?

## **Goal 3:** Determining the average sea surface iodide in the vicinity of Accra, Ghana

1. Add a sheet to the workbook by clicking on the "+" in the lower left corner

2. Rename the sheet "Location_and_boundaries"

Within the **Location_and_boundaries** sheet, we are going to build a box around specific latitude and longitude coordinates. Setting up the location and boundaries box in this way



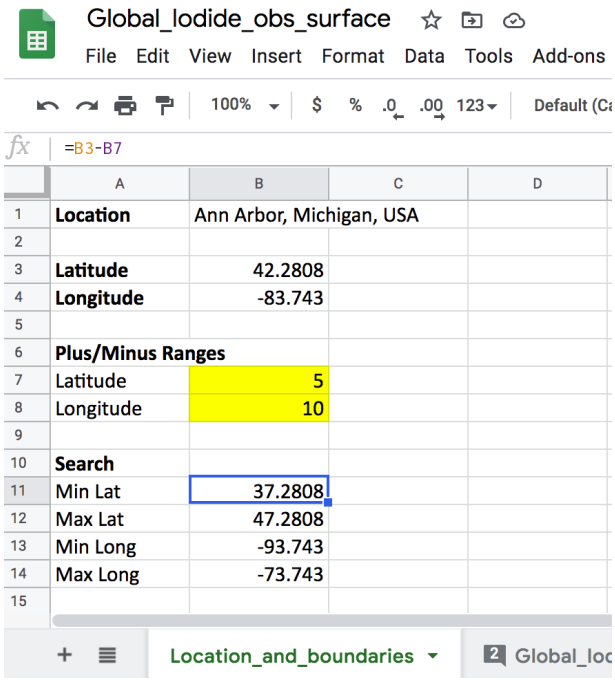
enables us to quickly and easily manipulate the range of latitudes and longitudes about a specific location in which we can search for iodide data. We will first build our box around Accra, with a range of latitude and longitudes to locate the presence of proximal data points for sea surface iodide concentrations.

3. In the **Location_and_boundaries** sheet, set up the text entries shown in Figure 1 (right)

4. Fill in the latitude and longitude values for Accra, Ghana in cells B3 and B4, respectively

5. Choose a range of latitude and longitude values you are interested in exploring beyond the coordinates of Accra (for example, ± 5° latitude and ± 15° longitude) and put the values in cells B7 and B8.

6. Using the **averageifs** function, determine the average sea surface iodide within our boundary box around Accra. If you do not capture any data points within your initial ranges, expand your range by 5° latitude and 5° longitude until you get a value.



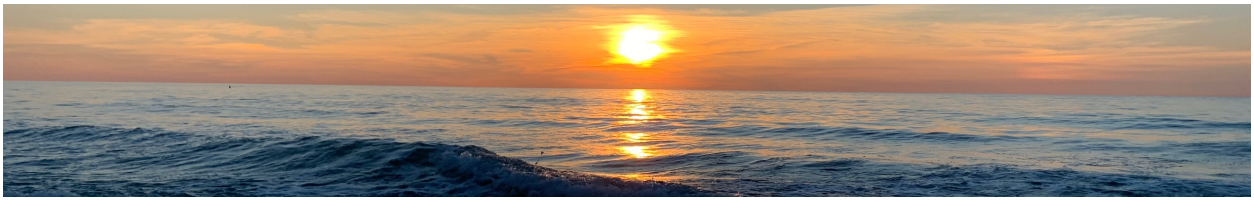

*Hint*: this function takes an average of the range you select if all of the conditions (criteria) that you specify are met. We care about meeting certain latitude and longitude criteria based on the box we built in the **Locations_and_boundaries** tab. To easily generate latitude and longitude boundaries, you can create a series of minimum and maximum lat/long values that call upon the central coordinates we are interested in investigating. An example of how to do this is shown in the figure above.

7. What is the average sea surface iodide concentration near Accra?

8. What were the boundaries of your latitude and longitude box? How much did you have to expand your ranges in order to get data, if at all?

9. How many data points contributed to this average?

   *Hint:* You can do this using the **countifs** function, with the same criteria as your **averageifs** function.



**Goal 4:** Determining the number of sea surface iodide measurements in the box bounding Accra, then determine the number of measurements near your home location (or near a location of choice, if Accra is your home)

Continue to focus on the coordinates of Accra (which we filled into cells B3 and B4on the **Location_and_boundaries** tab on the previous section. We are now going to use the **if** function and the **and** function to create a column where iodide concentration is only displayed in that row if the coordinates meet the criteria.

**if** is a conditional statement, which has 3 parts.

In **part 1** we describe a condition which will evaluate as either *true* or *false.* In our case, we are interested in knowing if the sea surface iodide value reported in a specific row falls within our boundary box around Accra.

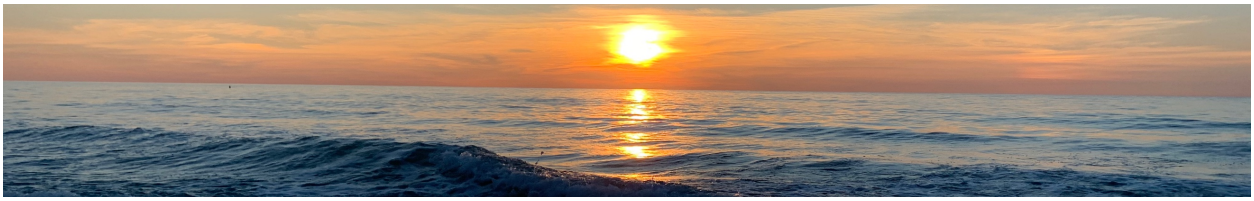
In **part 2** we assign a value to be returned if the first part of the statement is *true*. In our case, we would want the sea surface iodide concentration to be the returned value.

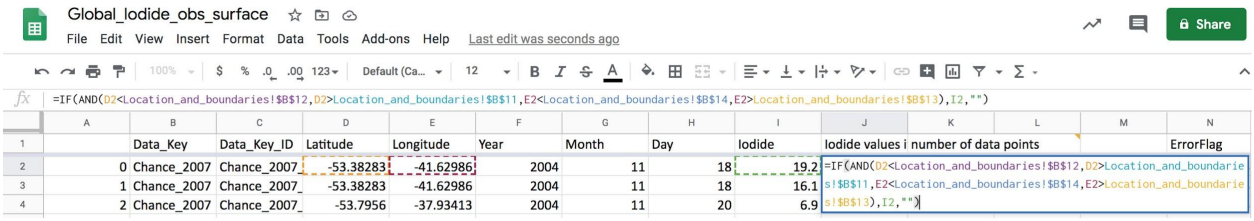
In **part 3** we assign a value if a condition described in the function is *false*. In our case, we do not want anything reported for false values, so we can assign a blank value, using double quotation marks **“”**

Because we need to test multiple values (latitude and longitude) to see if coordinates fall within our boundary box, we use the **and** function in part 1 of the **if** function.

**and** is the logical expression. Every statement within the **and**( ) needs to be true for the entire statement to evaluate as “true” otherwise it is false, and will return false.

A sample piece of code for evaluating whether or not a set of coordinates falls within the boundary box and returning the corresponding sea surface iodide concentration if true is below:






=IF(AND(D2<Location_and_boundaries!$B$12, Tests if latitude in each row (in this case, D2) is less than max

D2>Location_and_boundaries!$B$11, Tests if latitude in each row (in this case, D2) is greater than min

E2<Location_and_boundaries!$B$14, Tests if longitude in each row (in this case, E2) is less than max

E2>Location_and_boundaries!$B$13) Tests if longitude in each row (in this case, E2) is greater than min

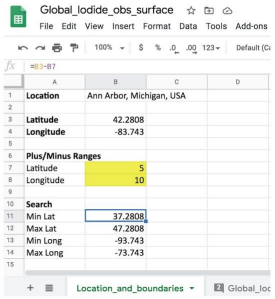
,I2,"")

Report the value in column I for that row if the *and* statement is true

Return a blank if the *and* statement is false

**After we make this formula in the first row, we can copy it down the column so it evaluates every data point (row).**

**For example, in row 3, the D2/E2 will become D3/E3. But, the max/min locations will stay the same because the $ characters are used.**




Now we are on the last step. Once you have generated the full list of values which fall within the boundary box using the equation above, you can use the **count** function to determine how many data points in the entire dataset were returned that meet the criteria specified above. This represents the number of sea surface iodide measurements present in our boundary box.

1. How many measurements in the dataset are within 20 degrees latitude and 30 degrees of longitude of Accra?

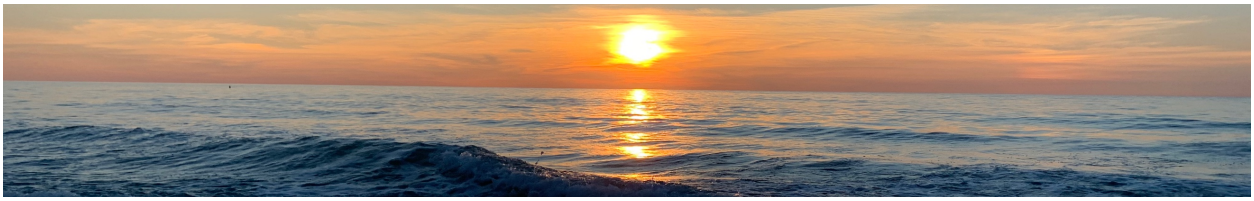
   *Hint*: if you set up your boundary box like the image above, you should list 10 and 15 in cells B7 and B8, respectively.

Continuing to use the **averageifs** formula in sheets, we can determine average sea surface iodide concentrations in the vicinity of coordinates we choose. Try to do this for your home coordinates! How wide do you need to make your boundary box to get data near home?

## Part II. Using python to visualize sea surface iodide concentrations

### **Goal 1:** Importing required modules into Google Colab

A module is a library of tools we can use carry out computing tasks within the Python environment. Standard Python come equipped with a large number of functions already, but as other needs have arisen many great people within the Python community have put great time and effort into creating other, more advanced, libraries.



In order to import these modules, we use the syntax "import module as alias". The 'as' statement is not required, but it allows us to give the module a shorthand so we don't have to type the full name over and over again. The module name and alias act as a sort of address; it tells the computer which library it should look in for the function name we are trying to use.

1. matplotlib is the standard Python plotting module and gets its name from its effort to emulate 'Mat'lab plot styles.

```
import matplotlib.pyplot as plt
```

2. Numerical Python, **numpy** for short, is another standard python module used in most scientific calculations and implements vectorized code.

```
import numpy as np
```

3. pandas is the standard Python data frame (data table) manager.

```
import pandas as pd
```

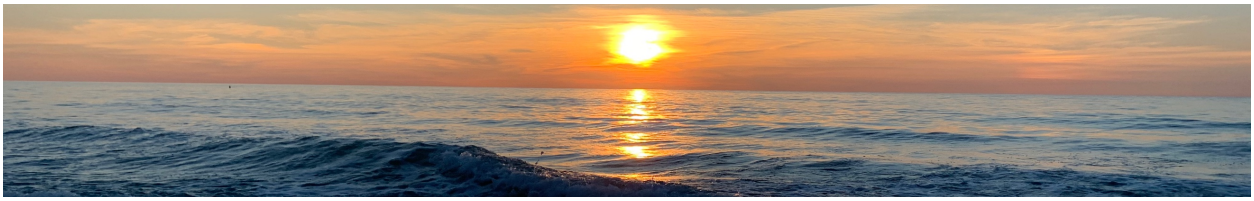
4. The following line of code is a google colab specific module for access to google drive. Here we are importing a single function from the module 'google' and its sub-library 'colab' with no alias.

```
from google.colab import drive
```

We ultimately plan to plot the sea surface iodide data superimposed on a global map, which requires importing cartopy. Below you will find the installation code for cartopy with some necessary steps for running in the google colab environment. The '!' is a way to execute the line in the terminal without leaving the colab environment.

5. This portion of code must be run EVERY TIME you start a new runtime, for example, whenever you close the colab notebook and reload it.

    *Hint*: If you include the installation code as a comment, as shown in the image below, you can uncomment to install and comment back out once installed. Be sure to remove any spaces at the front of the lines as well.



```
#_____________________uncomment below to install_____________________________
# !grep '^deb ' /etc/apt/sources.list | \
#   sed 's/^deb /deb-src /g' | \
#   tee /etc/apt/sources.list.d/deb-src.list
# !apt-get update

# !apt-get -qq build-dep python3-cartopy
# !apt-get -qq remove python-shapely python3-shapely

# !pip install --no-binary shapely shapely --force
# !pip install --no-binary cartopy cartopy==0.17.0
#_____________________________________________________________________________
```

6. Import cartopy (cartography in Python). This module is used for making coordinate transfomations, for example lon/lat to x/y and map projections (Mercator, etc). It is also used alongside matplotlib to render maps.

```
import cartopy.crs as ccrs
import cartopy.feature as cfeature
```

7. Seaborn is another Python module for plotting. I like it because it makes your plots very pretty by default using the simple command 'sns.set()' seen below.

```
import seaborn as sns
sns.set()
```

8. If you are using google colab, this line of code can be useful for accessing files in google drive. You should receive a prompt to enter a passcode and a url link. Click on the link and a new window showing the code should appear. Copy and paste the code into the empty cell below in the notebook and that should allow you to mount your google drive.

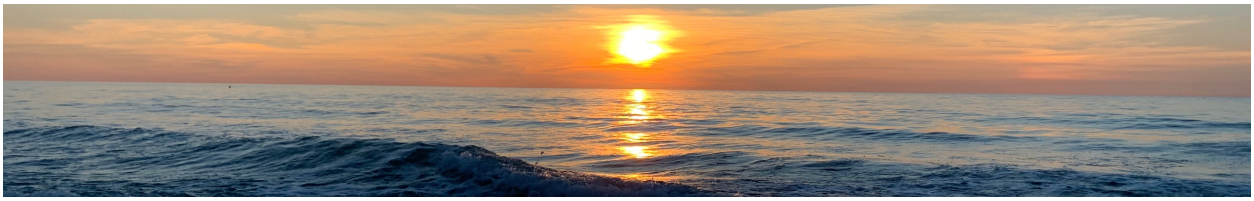
```
drive.mount('/content/drive')
```

   during installation of drive mount you should see activity below... don't freak out. ☺

## **Goal 2:** Uploading data file(s) and opening in Google Colab

1. First, be sure the file is in your google drive! Then use pandas to upload the data file from google drive (some parts of the file path may need to be changed).

   pd.read_csv can read any plain text file, not just .csv's (e.g., .txt, .dat)
   'sep=',' indicates the file is comma delimited, and 'skipinitialspace=True' means we remove any white space before the data.



```
DATA = pd.read_csv('/content/drive/My Drive/Global_Iodide_obs_surface.csv', sep=',', skipinitialspace=True)
```

*NOTE: variables labeled in all caps is coding convention for constants that should not be changed. Since this is the original data file, we keep it as is.

2. Now we will use the command: display(DATA.head(10)) to inspect the imported data frame.

   *Hint*: The method '.head(10)' allows us to look at only the first ten entries which should be sufficient to tell whether the data was imported properly or not.

3. Since we don't want to alter the original data, we drop two unwanted columns. 'Unnamed: 0','Œ¥Iodide' from the dataframe and create a new dataframe named 'fulldata'. 'axis=1' means we drop it as a column as opposed to a row (axis=0).

```
fulldata = DATA.drop(['Unnamed: 0','Œ¥Iodide'], axis=1)
```

4. Using the dataframe method '.loc' we can grab all data gathered in the year 2000 and beyond. This is done by saying "in the dataframe 'fulldata' grab all the rows where the column with the header 'Year' is greater than or equal to 2000"…

   This logic is all captured by the [fulldata.Year >= 2000].

```
past2000 = fulldata.loc[fulldata.Year >= 2000]
```

## **Goal 3:** Extracting Data from the data frames and generating plots

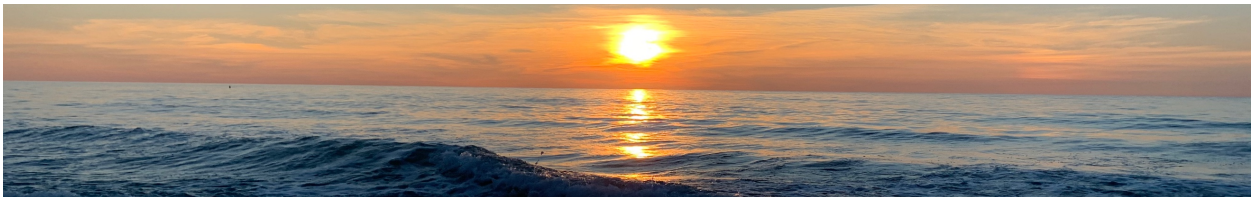
1. Here we convert the longitude and latitude information for 'fulldata' into two separate numpy arrays.

```
lon, lat = np.array(fulldata['Longitude']), np.array(DATA['Latitude'])
```

   We do this because arrays are typically handled better by many Python functions. Notice we can set two variables in one line as long as we have two pieces of information separated by commas on both sides of the equals sign. This can be extended to any number of variables, but don't make your lines too long!

2. Pull out the Iodide data

```
iodide = np.array(fulldata['Iodide'])
```



The line below takes the iodide array and sorts from least to greatest and then flips it from greatest to least. We are organizing our iodide data so we can create a scatter plot

```
sorted_iodide = np.flip(np.sort(iodide))
```

3. Set up x values for plotting the iodide information (sorted_iodide).
   *Hint*: Because we are only interested in looking at the magnitude of iodide concentrations recorded, not the years or locations, we create an array numbered from zero to the number of data points (remember Python indexing begins at 0).

```
x = np.arange(0,iodide.shape[0])
```

4. Using matplotlib.pyplot to generate a figure object with size [24,6]. The values in [] describe the dimensions of the figure.

```
fig = plt.figure(figsize=[24,6])
```

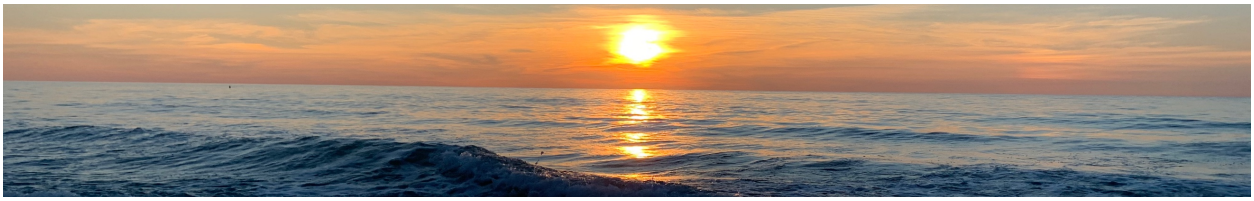
'suptitle' creates a label for entire figure

```
fig.suptitle('Iodide Data', size=36)
```

5. Use the .add_subplot() method to generate subplots and assign them to their respective 'axis'. In .add_subplot(###) the ### determine how the figures are layed out, i.e, rows/cols/plot#... see example code in figure below.

```
# Here 13# is read "make 1 row with 3 columns" and the last number is the
# the panel where that subplot will go from left to right and then row by row.
ax1, ax2, ax3 = fig.add_subplot(131), fig.add_subplot(132), fig.add_subplot(133)

## regular plot ##
# scatter plot
ax1.scatter(x, sorted_iodide)
# set label for x-axis
ax1.set_xlabel('number of samples',size=20)
# set label for y-axis
ax1.set_ylabel('Iodide (I$^{-}$)',size=20)
ax1.text(0.08,0.11, 'iodide data with no\n transformation',
         transform=ax1.transAxes, fontsize=16)
```



6. Follow the code outlined above, and make two more plots of iodide in the subplot, one following a $\log_{10}$ transformation of sea surface iodide concentrations, and another following a cubic root transformation. Sample code for these plots below:

```
ax2.scatter(x,np.log10(sorted_iodide))

ax3.scatter(x, sorted_iodide**(1/3))
```

## **Goal 4:** Plotting sea surface iodide in a global context – Equidistant Projections

This section uses cartopy functions ccrs, coastline, and add_feature.

1. Using matplotlib.pyplot to generate a figure object with size [20,16]. Refer to code above (Goal 3, question 4) for an example of how to initialize a figure.

2. Create a map projection, as shown below. The 1,1,1 here operates in the same manner as the add_subplot(###) does above...

```
ax = fig.add_subplot(1, 1, 1, projection=ccrs.PlateCarree())
```

   NOTE: Plate Carree is an equidistant or equirectangular projection. For more information see here.

3. Set a title for your map.

```
ax.set_title("Plate Carree Equidistant Projection", size=30)
```

4. Plot sea surface iodide data. Use the 'c=' option to determine what data dictates the color fill and the option 'cmap=' to set the colormap. A google search of "python colormaps" will give you plenty to choose from. In this graph, we will plot the $\log_{10}$ of sorted_iodide.

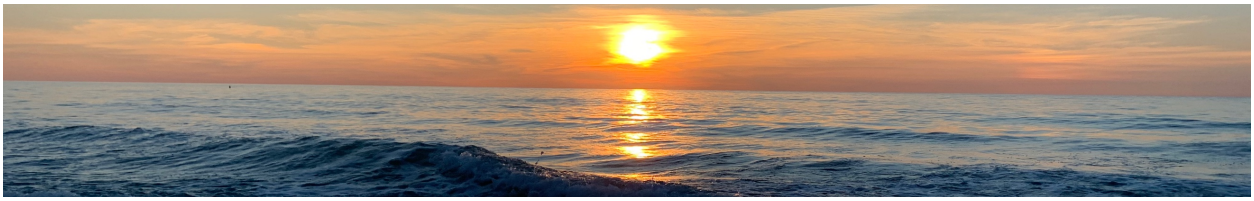
```
img = ax.scatter(lon, lat, c=np.log10(sorted_iodide), cmap='jet')
```

5. Use the following command to plot coastline polygons for the landmasses (continents).

```
ax.coastlines(resolution='110m')
```

6. Use the following commands to color in the landmass polygons.

```
ax.add_feature(cfeature.LAND, color='k')
ax.add_feature(cfeature.OCEAN, color='darkgrey')
```



```
ax.gridlines(draw_labels=True, color='black')
```

7. Use the following commands to adjust the colorbar axis.

```
cb_ax = fig.add_axes([0.93, 0.3, 0.02, 0.4])
```

use '.set_label' to change the title on the colorbar

```
fig.colorbar(img, cax=cb_ax).set_label('log$_{10}$(I$^{-}$)', size=20);
```

*NOTE: You may receive some download warnings...they can safely be ignored.

## **Goal 5:** Plotting sea surface iodide in a global context – Equal Area Projections

This section uses cartopy functions ccrs, coastline, and add_feature.

1. Using matplotlib.pyplot to generate a figure object with size [22,16].

2. Create a map projection, as shown below. The 1,1,1 here operates in the same manner as the add_subplot(###) does above...

```
ax = fig.add_subplot(1, 1, 1, projection=ccrs.EckertIV())
```

NOTE: Eckert IV is an equal area projection. For more information see here.

3. Set a title for your map.

```
ax.set_title("Eckert IV Equal-Area Projection", size=30)
```

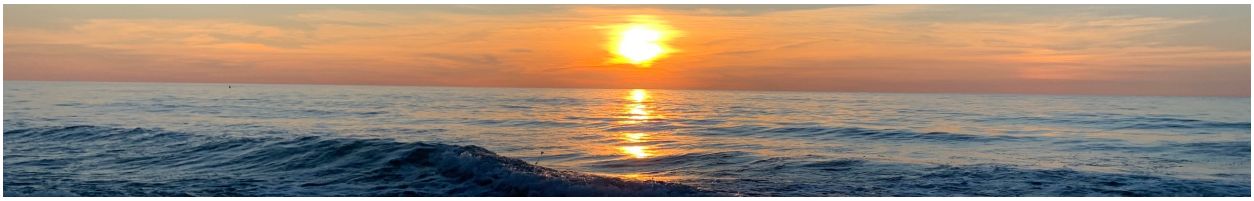
4. Plot sea surface iodide data. Use the 'c=' option to determine what data dictates the color fill and the option 'cmap=' to set the colormap. A google search of "python colormaps" will give you plenty to choose from.

```
img = ax.scatter(lon, lat, c=np.log10(sorted_iodide), cmap='jet',
                 transform=ccrs.Geodetic())
```

5. Use the following command to plot coastline polygons for the landmasses (continents).

```
ax.coastlines(resolution='110m')
```

6. Use the following commands to color in the landmass polygons.



```
ax.add_feature(cfeature.LAND, color='k')
ax.add_feature(cfeature.OCEAN, color='darkgrey')
ax.gridlines(draw_labels=True, color='black')
```

7. Use the following commands to adjust the colorbar axis.

```
cb_ax = fig.add_axes([0.93, 0.3, 0.02, 0.4])
```

use '.set_label' to change the title on the colorbar

```
fig.colorbar(img, cax=cb_ax).set_label('log$_{10}$(I$^{-}$', size=20);
```

*NOTE: You may receive some download warnings...they can safely be ignored.

8. Compare the two projections you made (equidistant and equal area). How are they similar, and how are they different?

9. Assess the global distribution of sea surface iodide data. How would you describe its spread? Where do you see the most data points, where do you see the fewest?

10. Comment on any general trends you see in location (latitude, longitude, coastal, open ocean) in sea surface iodide concentration.