# Computer Lab 5: Atlantic Tropical Storms

Climate Data Analysis, ATS 301, Fall 2018

Primary objectives for today:

- Plot (in different ways) information about tropical storms, hurricanes, and major hurricanes.

## Reading in the data (possibly Pre-Lab 5)

As before, we start by specifing that we want plots to be displayed inside the Jupyter Notebook, and load the modules we'll need.

In [ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
```

Read in the data, again using the `np.genfromtxt` function. The file is located at

`/data/ATS_301/Data/tropical_storm_2017.csv`

In [ ]:
```
# Put the correct options here, based on the file.
storm_num=np.genfromtxt("/data/ATS_301/Data/tropical_storm_2017.csv",????)
```

In [ ]:
```
# ↔
```

**Record the information about the source of the data in a Markdown cell.**

Verify that the size of the array is what you'd expect.

In [ ]:
```
# Put the function to show the size of the array here.
```

In [ ]:
```
#↔
```

Look at file itself. Which column corresponds to the years? Verify this by printing out just that column (i.e., print out the years). (Remember that the first index is 0, and ":" means everything in that dimension.) What are the first and last years of this data set?

In [ ]:
```
# Replace XXXXXXX with correct indices to show the years.
storm_num[XXXXXX,XXXXXX]
```

In [ ]:
```
#↔
```

Print out the data for total number of "named storms (revised)" to see what sort of numbers you get.

In [ ]:
```
# Replace with the correct index.
storm_num[XXXXXX,XXXXXX]
```

In [ ]:
```
#↔
```

Plot the three columns that correspond to the (revised) number of named storms, hurricanes, and major hurricanes as a function of year.

This is basically the same thing as what you did with the temperature data; you're just using a different dataset. Copying your code from the previous assignment is fine here. (Reusing your own code is usually a great place to start.)

Note that you cannot copy cells directly from one notebook to another. Instead, highlight the code you want to copy in a cell, press ⌘C (for Copy), go the the notebook you want to copy it to, and press ⌘V (for Paste).

In [ ]:
```
###### your plotting code goes here. Be sure in include axis labels and and a lege
nd.
# Play around with the plotting options (markers, colors, lines, etc.) to make the
data
# as easy to read as possible.
```

In [ ]:
```
#↔
```

### End of Pre-Lab

## Basic Statistics

Calculate basic statistics for these three (revised) variables. To do this, use the the numpy commands `np.median`, `np.mean`, and `np.std` on just **one** column of data.

**Note**: if you input more than a single row or column to these commands, you might get an answer other than you'd expect. We'll talk about this later, but, for now, just stick to a single column at a time.

As an example, here's how to print out the median, mean, and std for (revised) named storms.

In [ ]:
```
print(np.median(storm_num[:,2]))
print(np.mean(storm_num[:,2]))
print(np.std(storm_num[:,2]))
```

You can also look at the maximum and minimum values. If you specify a subset or slice of the array, thes `max` and `min` functions will find the max or min of just that subset.

In [ ]:
```
print(max(storm_num[:,2]))
print(min(storm_num[:,2]))
```

Now calculate the median, mean, std, minimum, and maximum of the number of hurricanes.

In [ ]:
```
# Your code goes here↔
```

Printing out all of these numbers works OK when you don't have many numbers, but it can get difficult to keep track of what is what.

So, we add some text to the `print` functions. You can print multiple quantities separated by commas in one `print` function.

In [ ]:
```
print("Median named storms: ",np.median(storm_num[:,2]))
print("Mean named storms: ",np.mean(storm_num[:,2]))
print("Standard deviation of named storms: ",np.std(storm_num[:,2]))
```

# Tables

To make it even easier to look at the data, we'll organize it into tables.

As we did last week, we will be using the `format` function to specify how many digits to display. Here's an example of a row in the table:

In [ ]:
```
print('            Named    Hurr.  Str Hurr   U.S.')
print('Median:     {0:.2f}          {1:.2f}     {2:.2f}     {3:.2f}'.format\
                                    (np.median(storm_num[:,2]), \
                                     np.median(storm_num[:,4]), \
                                     np.median(storm_num[:,6]), \
                                     np.median(storm_num[:,10])))
```

**Note what happens when you type the backslash and press enter.** The backslash indicates that the command continues on the next line. You can do this to avoid having the line of code get too wide for the window.

Now add lines for the mean, standard deviation, maximum, and minimum.

In [ ]:
```
#### Put your table here↔
```

## Comparison with original data

Next, try calculating the median for the original named storm data.

In [ ]:
```
np.median(storm_num[:,1])
```

What do you think this means? Take a look at the data in this column:

In [ ]:
```
storm_num[:,1]
```

To ignore the nan's and calculate the statistical properties for the remaining values, use `np.nanmedian` , `np.nanmean` , `np.nanstd` , `np.nanmax` , and `np.nanmin` instead.

Note that these functions are part of the `numpy` module, so you have to include the `np.` prefix here.

In [ ]:
```
print(np.nanmedian(storm_num[:,1]))
print(np.nanmean(storm_num[:,1]))
print(np.nanstd(storm_num[:,1]))
print(np.nanmin(storm_num[:,1]))
print(np.nanmax(storm_num[:,1]))
```

Now you can create a table with the statistical properties for the original data. **Note that this data includes fewer years**, so we have to be careful when comparing to the revised data, in case the hurricane properties were actually different for the years with missing original data.

Finally, include a header on the table to indicate this is original data. You'll want to add a header to the revised table you made earlier.

In [ ]:
```
print('Original data')
print('-------------')
print('            Named    Hurr.  Str Hurr   U.S.')
print('Median:    {0:.2f}     {1:.2f}     {2:.2f}     {3:.2f}'.format\
                                                (np.nanmedian(storm_num[:,1]), \
                                                 np.nanmedian(storm_num[:,3]), \
                                                 np.nanmedian(storm_num[:,5]), \
                                                 np.nanmedian(storm_num[:,9])))
```

In [ ]:
```
# Finish the table↔
```

## Histograms

Now, create a histogram with the named storm data, using the `plt.hist` function.

In [ ]:
```
help(plt.hist)
```

In [ ]:
```
plt.hist(storm_num[:,2])
```

I like to clearly see the edges of the different bins sometimes. To do this, use the `ec` (short for `edgecolor`) optional argument, with a different color (`k` - black, in this case).

In [ ]:
```
plt.hist(storm_num[:,2],ec="k")
```

By default, the `plt.hist` uses 10 bins.

- The first output array shows the number of data points (years, in this case) that have values within each bin.
- The second output array shows the beginnings (inclusive) and ends (exclusive, except for the final bin) for each bin. [The range of the final bin is inclusive. Yes, this is confusing.]
- For example, there are 2 years with between 1 and 3.7 tropical storms.

You can specify a different number of bins using the `bins` option. Play around with this value to see what the histogram looks like with different numbers of bins.

In [ ]:
```
plt.hist(storm_num[:,2],bins=20,ec="k")
```

You can also use tell python to use an algorithm to determine the optimal number of bins by using `bins='auto'` :

In [ ]:
```
plt.hist(storm_num[:,2],bins='auto',ec="k")
```

## Specifying the bins exactly

For this dataset, we know we have only integers (except for ACE). It doesn't really make sense to have ranges that have floats, rather than integers, for their ranges. So, it would be good if the ranges included the same number of integers within each. The best way to do this is to specify the ranges of the bins exactly. To do this, we need to create a 1-D array.

What range of values do we need to include in the array? Use the `min` and `max` functions to determine this.

In [ ]:
```
max(storm_num[:,2])
```

In [ ]:
```
min(storm_num[:,2])
```

Our max value is 28. Even though the minimum value here is 1, we'll soon want to compare this histogram to one for strong hurricanes, which do have some values of 0. So we'll create an array that goes from 0 to 29. (The 29 is because we want the last bin to *inclusive* on both ends, and we don't want include 27 and 28 in the same bin.

**One quick way to make a 1-D array with integers increasing from 0 to some number, in increments of 1, is the `np.arange` function.**

We want an array of size 30 (since arrays start with an index of 0).

Start by printing out the array to make sure it's what you expect.

In [ ]:
```
np.arange(30)
```

Now we can use this array for the `bin` input variable in the `plt.hist` command.

By default, `plt.hist` centers the bars in the middle of the bin ranges. However, we have created the bin array such that the first edge of the array corresponds to the number of storms for that bin. The `align="left"` options centers the bars around the left sides of the ranges. This is a little subtle in this case, because we have so many bins. It will be a little easier to see when we look at the hurricane data.

In [ ]:
```
plt.hist(storm_num[:,2],bins=np.arange(30),ec="k")
```

In [ ]:
```
plt.hist(storm_num[:,2],bins=np.arange(30),align="left",ec="k")
```

Now make a histogram of the number of *hurricanes*.

In [ ]:
```
# Your code here.↔
```

To better compare the numbers, we can put all three histograms on the same plot.

**Use the same bins for all three histograms** so that they line up. This is why we're specifying the bins exactly.

In [ ]:
```
plt.hist(storm_num[:,2],bins=np.arange(30),align="left",ec="k")
plt.hist(storm_num[:,4],bins=np.arange(30),align="left",ec="k")
plt.hist(storm_num[:,6],bins=np.arange(30),align="left",ec="k")
```

Some of the bins bars overlap, so we can use the `alpha` option to specify the opacity, with 0 entirely transparent and 1 entirely opaque (the default).

In [ ]:
```
plt.hist(storm_num[:,2],bins=np.arange(30),align="left",ec="k",alpha=0.4)
plt.hist(storm_num[:,4],bins=np.arange(30),align="left",ec="k",alpha=0.5)
plt.hist(storm_num[:,6],bins=np.arange(30),align="left",ec="k",alpha=0.4)
```

You can get rid of the individual lines by dropping the `ec` argument.

In [ ]:
```
plt.hist(storm_num[:,2],bins=np.arange(30),align="left",alpha=0.4)
plt.hist(storm_num[:,4],bins=np.arange(30),align="left",alpha=0.5)
plt.hist(storm_num[:,6],bins=np.arange(30),align="left",alpha=0.4)
```

Alternately, we can use a `histtype='step'` to just show the lines around the edges. Also, I'm increasing the `linewidth` to 2, to make the lines easier to see.

In [ ]:
```
plt.hist(storm_num[:,2],bins=np.arange(30),align="left",histtype='step',linewidth=
2)
plt.hist(storm_num[:,4],bins=np.arange(30),align="left",histtype='step',linewidth=
2)
plt.hist(storm_num[:,6],bins=np.arange(30),align="left",histtype='step',linewidth=
2)
```

`histype='stepfilled'` draws a line around the edges of each histogram and fills the inside with the specified color. The `color` and the `ec` should be different, so you can see the outline. (In this case, we are not specifying the `color`, just using the default values.)

In [ ]:
```
plt.hist(storm_num[:,2],bins=np.arange(30),align="left",histtype='stepfilled',line
width=2,ec="k",alpha=0.5)
plt.hist(storm_num[:,4],bins=np.arange(30),align="left",histtype='stepfilled',line
width=2,ec="k",alpha=0.5)
plt.hist(storm_num[:,6],bins=np.arange(30),align="left",histtype='stepfilled',line
width=2,ec="k",alpha=0.5)
```

Yet another way to view the data is to plot the individual histogram bars next to each other. You can do this by providing all three arrays to the `plt.hist` function at once. Note that they are all within a set of `[]` and separated by commas. We are effectively passing a *list* of the 3 arrays we want plotted.

In [ ]:
```
plt.hist([storm_num[:,2],storm_num[:,4],storm_num[:,6]],bins=np.arange(30),align="
left")
```

Play around with the different options until you find a style you like. (Note that you can mix and match all of the different fill colors, line colors, opacities, histtypes, etc. if you plot using the three separate `plt.hist` functions.)

**Add the labels and legend** to finish the plot. You can also use the `plt.axis` command (which we'll talk about later) to get rid of the wasted space on the left when you do your homework.

## Bar plots

Bar plots use vertical bars, rather than markers or lines. Depending on your data, they can sometimes be clearer than line plots. We'll start by making a bar plot for the named storms.

The `plt.bar` function works similarly to the `plt.plot` function. The x values goes first, followed by the y values. You can specify things like color, as well:

In [ ]:
```
plt.bar(storm_num[:,0],storm_num[:,2],color='y')
plt.ylabel("Number of systems")
plt.xlabel("Year")
plt.title("Atlantic basin storm count")
```

This is sort of difficult to read.

By default, python picks the ranges for the x and y axis. However, you can specify these yourself using the `plt.axis` command.

In [ ]:
```
help(plt.axis)
```

In [ ]:
```
## fill in the values for the plt.axis function

plt.bar(storm_num[:,0],storm_num[:,2],color='y')
plt.axis([???, ???, ???, ???])
plt.ylabel("Number of systems")
plt.xlabel("Year")
plt.title("Atlantic basin storm count")
```

In [ ]:
```
#↔
```

That's somewhat better.

Next, let's increase the size of the figure using the `plt.figure` command. Note that this will also alter the aspect ratio of the plot, which we want in this case, because we would like to stretch out the x axis. The first number corresponds to the x axis length, and the second to the y axis.

**Note: rather than re-typing all of this code, it's probably easiest to copy the cell and paste it as you go from step to step.**

In [ ]:
```
# Play around with the figsize values
plt.figure(figsize=(XXX, XXXX))
# Put rest of the commands here.
```

In [ ]:
```
#↔
```

We can plot multiple data sets on the same bar plot. Depending on the options specified, the bars will appear on top of each other or next to each other. By default, they will appear on top of any previous bar plots.

In [ ]:
```
# Add the appropriate additional commands to make this plot pretty.

plt.bar(storm_num[:,0],storm_num[:,2],color='y',label="Named storms",snap=False)
plt.bar(storm_num[:,0],storm_num[:,4],color='r',label="Hurricanes",snap=False)
plt.bar(storm_num[:,0],storm_num[:,6],color='purple',label="Major hurricanes",snap
=False)
```

In [ ]:
```
#↔
```

Try re-arranging the ordering of the data. What happens?

In [ ]:
```
plt.bar(storm_num[:,0],storm_num[:,6],color='purple',label="Major hurricanes",snap
=False)
plt.bar(storm_num[:,0],storm_num[:,2],color='y',label="Named storms",snap=False)
plt.bar(storm_num[:,0],storm_num[:,4],color='r',label="Hurricanes",snap=False)
```

Return to the original ordering of the `plt.bar` commands so you can see all the data again.

## Plotting ACE

Next, we want to plot ACE, similar to how it is done here:

https://www.epa.gov/climate-indicators/climate-change-indicators-tropical-cyclone-activity (https://www.epa.gov/climate-indicators/climate-change-indicators-tropical-cyclone-activity)

What sort of plot is this?

1. Specify the correct column in the array.
2. To make the bars somewhat separate, we can specify their `width` . The default is 0.8. Try a few different values.
3. Also, notice that you can specify some colors by a name, rather than a single letter. This Web page lists the named colors:

http://matplotlib.org/examples/color/named_colors.html (http://matplotlib.org/examples/color/named_colors.html)

In [ ]:
```
plt.figure(figsize=(???, ???))
plt.bar(storm_num[???,???],storm_num[???,???],color='???',width=???,snap=False)
plt.ylabel("ACE")
plt.xlabel("Year")
plt.title("Atlantic basin Accumulated Cyclone Energy Index")
```

In [ ]:
```
#↔
```

One handy thing you can use the `plt.axis` command for is to limit the plot to a smaller portion of the data. You don't have to specify the range within the `plt.bar` or `plt.plot` commands themselves.

For example, to enlarge part of the earlier bar chart, we just change the x axis range. This works particularly well for years, because then you don't need to figure out which index corresponds to a particular year.

In [ ]:
```
plt.figure(figsize=(8, 4))
plt.bar(storm_num[:,0],storm_num[:,2],color='y',label="Named storms")
plt.bar(storm_num[:,0],storm_num[:,4],color='r',label="Hurricanes")
plt.bar(storm_num[:,0],storm_num[:,6],color='purple',label="Major hurricanes")
plt.axis([1970, 2018, 0, 30])
plt.legend(loc=2)
plt.ylabel("Number of systems")
plt.xlabel("Year")
plt.title("Atlantic basin storm count")
plt.show()
```

Try this with the ACE data, to match the plot on the website.

In [ ]:
```
##### Your plot goes here.↔
```

Finally, we can add horizontal lines to the plot using the `plt.axhline` command with the `y=` option.

In this case, we want lines at 120 and 71.4. Add the following lines to your plot. **They must be before the `plt.show` command**.

```
plt.axhline(y=120)

plt.axhline(y=71.4)
```

In [ ]:
```
##### Your plot goes here.↔
```

## Fraction of hurricanes that become strong hurricanes

Let's start by just trying to plot this fraction. We can limit it to the years shown in the plot in the lecture:

In [ ]:
```
plt.plot(storm_num[:,0],(storm_num[:,6]/storm_num[:,4]))
plt.axis([1970,2012,0,1])
```

It worked, but what does that error message mean? Print out the fraction to see what it looks like:

In [ ]:
```
storm_num[:,6]/storm_num[:,4]
```

We've got some nan's in there. Why do we get these?

In [ ]:
```
storm_num[:,4]
```

However, these nan's are all before the time period we're interested in. So we can ignore this error message this time. In general, though, we need to be very careful when doing division where the denominator may be zero.

Now, **add the new years of data** to the plot for comparison with the year range on the UCS web site (which is now out of date by a few years).

In [ ]:
```
# Your code goes here.↔
```

Zoom back out to the entire record. What is happening to the variability over time? Do you think the variability is really changing?

In [ ]:
```
plt.plot(storm_num[:,0],(storm_num[:,6]/storm_num[:,4]))
```

Prettify the plot for your homework.

In [ ]:
```

```