

## Arctic Sea Ice Extent: Data Analysis and Visualization

ISTA 131 Hw7, Due 10/25/2018 at 11:59 pm

**Introduction.** This homework is the last 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 analyze the Arctic sea ice extent data and use your analysis to predict the date the ice will disappear altogether. You will do this by fitting lines to the March and September data. This is called simple linear regression, calculated using ordinary least squares. You will also recalculate the data as the anomaly. The anomaly is the difference between a given data point and the mean. In this case, we are interested in the maximum ice extent, which occurs in March but varies from year to year as to the exact day, and the minimum ice extent, which happens in September. Therefore, we are going to look at those months as single entities, i.e. the March and September means for each year. In other words, instead of looking at daily data, we will be looking at monthly data (the mean of the daily data for the month). To get the anomaly for the monthly data, we will calculate the difference between the monthly mean for a given year and the mean of the means for all of the years. For example, you will be calculating the difference between the mean of the March data for 1981 and the mean of the March means for all of the years. Looking at the anomaly instead of the absolute values will allow you to create a better visualization that compares how ice behavior is changing in March with how ice behavior is changing in September.

**Instructions.** Create a module named `hw7.py`. Below is the spec for six functions. Implement them and upload your module to D2L Assignments.

**Testing.** Download `hw7_test.py` and auxiliary testing files and put them in the same folder as your `hw7.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 Hw7, 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.

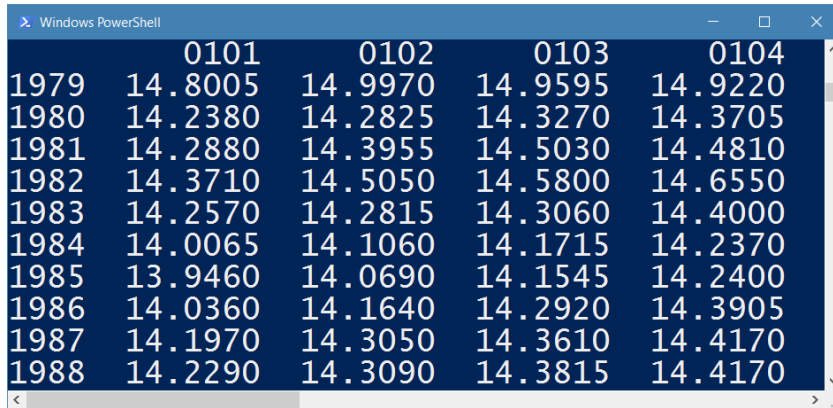
### Resources.

<http://statsmodels.sourceforge.net/devel/examples/notebooks/generated/ols.html>

[http://statsmodels.sourceforge.net/devel/generated/statsmodels.regression.linear\\_model.RegressionResults.html?highlight=regressionresults](http://statsmodels.sourceforge.net/devel/generated/statsmodels.regression.linear_model.RegressionResults.html?highlight=regressionresults)

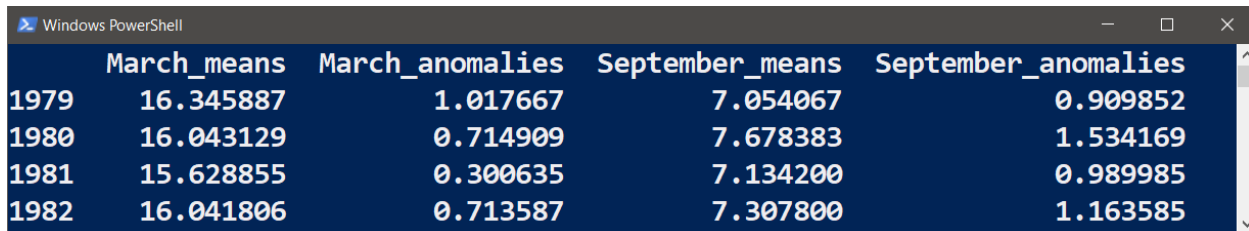
[http://statsmodels.sourceforge.net/devel/example\\_formulas.html](http://statsmodels.sourceforge.net/devel/example_formulas.html)  
<https://docs.python.org/3/library/math.html>

`get_Mar_Sept_frame`: This function takes no arguments. Read `data_79_17.csv` into a `hw4` frame. Recall that it looks like this:



	0101	0102	0103	0104
1979	14.8005	14.9970	14.9595	14.9220
1980	14.2380	14.2825	14.3270	14.3705
1981	14.2880	14.3955	14.5030	14.4810
1982	14.3710	14.5050	14.5800	14.6550
1983	14.2570	14.2815	14.3060	14.4000
1984	14.0065	14.1060	14.1715	14.2370
1985	13.9460	14.0690	14.1545	14.2400
1986	14.0360	14.1640	14.2920	14.3905
1987	14.1970	14.3050	14.3610	14.4170
1988	14.2290	14.3090	14.3815	14.4170

Use it to create and return a frame that looks like this:



	March_means	March_anomalies	September_means	September_anomalies
1979	16.345887	1.017667	7.054067	0.909852
1980	16.043129	0.714909	7.678383	1.534169
1981	15.628855	0.300635	7.134200	0.989985
1982	16.041806	0.713587	7.307800	1.163585

If you find yourself creating an empty `DataFrame` and filling in positions one at a time (it can be avoided), make sure that your new elements are of type `np.float64` by passing the optional argument by keyword `dtype=np.float64` when you call the `DataFrame` constructor. You will need to include the statement `import numpy as np` at the top of your module.

`get_ols_parameters`: This function takes a `Series`, fits a line to it, and returns the slope, intercept,  $R^2$ , and the p-value in a list.

`make_prediction`: The signature for this function should look exactly like this:

```
def make_prediction(params, description='x-intercept:', x_name='x',  
                    y_name='y', ceiling=False):
```

The *default arguments* provided with for each of the last four parameters makes passing corresponding arguments optional. You can see calls to this function without corresponding arguments in the test, if you wish.

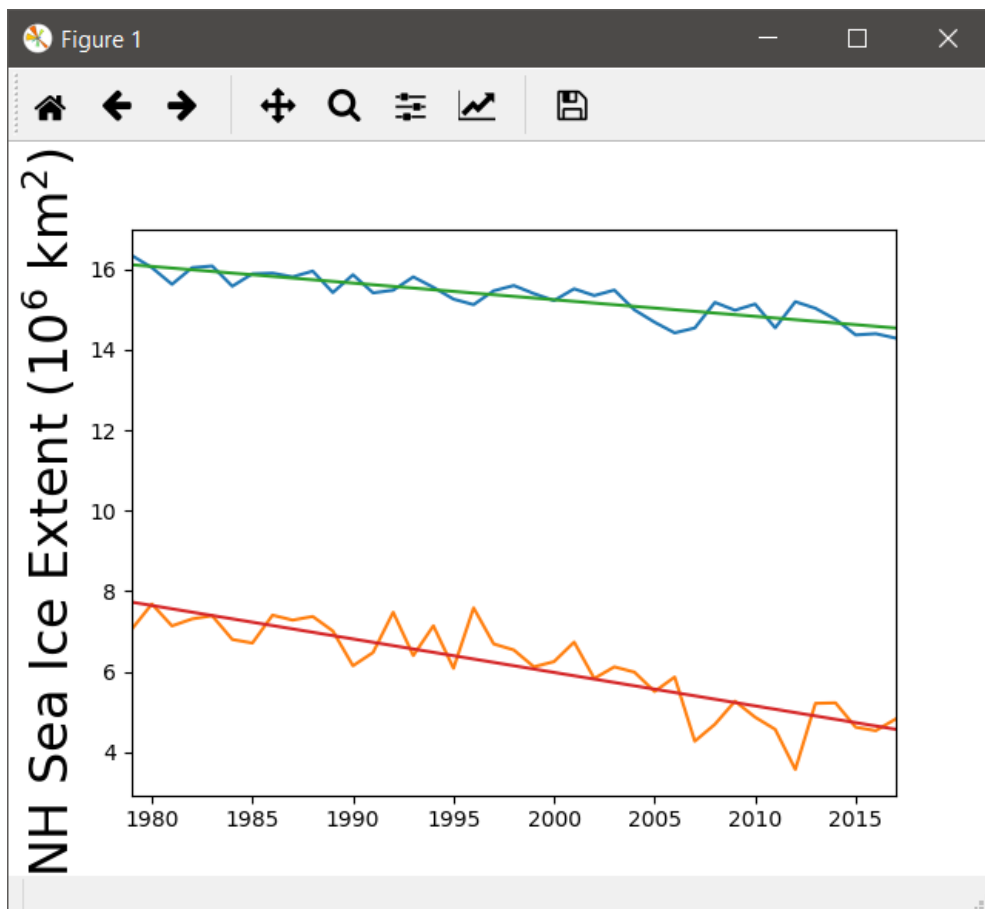
Calculate the x-intercept (the value of  $x$  for which  $y = 0$ ) using the OLS parameters contained in the first argument. If the last argument is `True`, round the intercept up to the nearest integer  $\geq$  x-intercept (hint: `math.ceil`). Print your result in the following format:

```
Windows PowerShell
Ice-free March in year 2369
77.0% of variation in Arctic sea ice accounted for by time (linear model)
Significance level of results: 0.0%
This result is statistically significant.
```

The first line is the description parameter followed by the x-intercept. The number at the beginning of the second line (77 in this example) is calculated using  $R^2$ . Round it using Python's built-in `round` function with one argument. You will also have to use the `y_name` and `x_name` parameters to construct the second sentence. `round(x)` returns an integer unless `x` is type `np.float64`, in which case you get an `np.float64`. Hence the `'.0'` in this example, but not in the test examples. Round the significance level to 1 decimal place. The message should say "This result is not statistically significant." if the p-value is  $> 0.05$ .

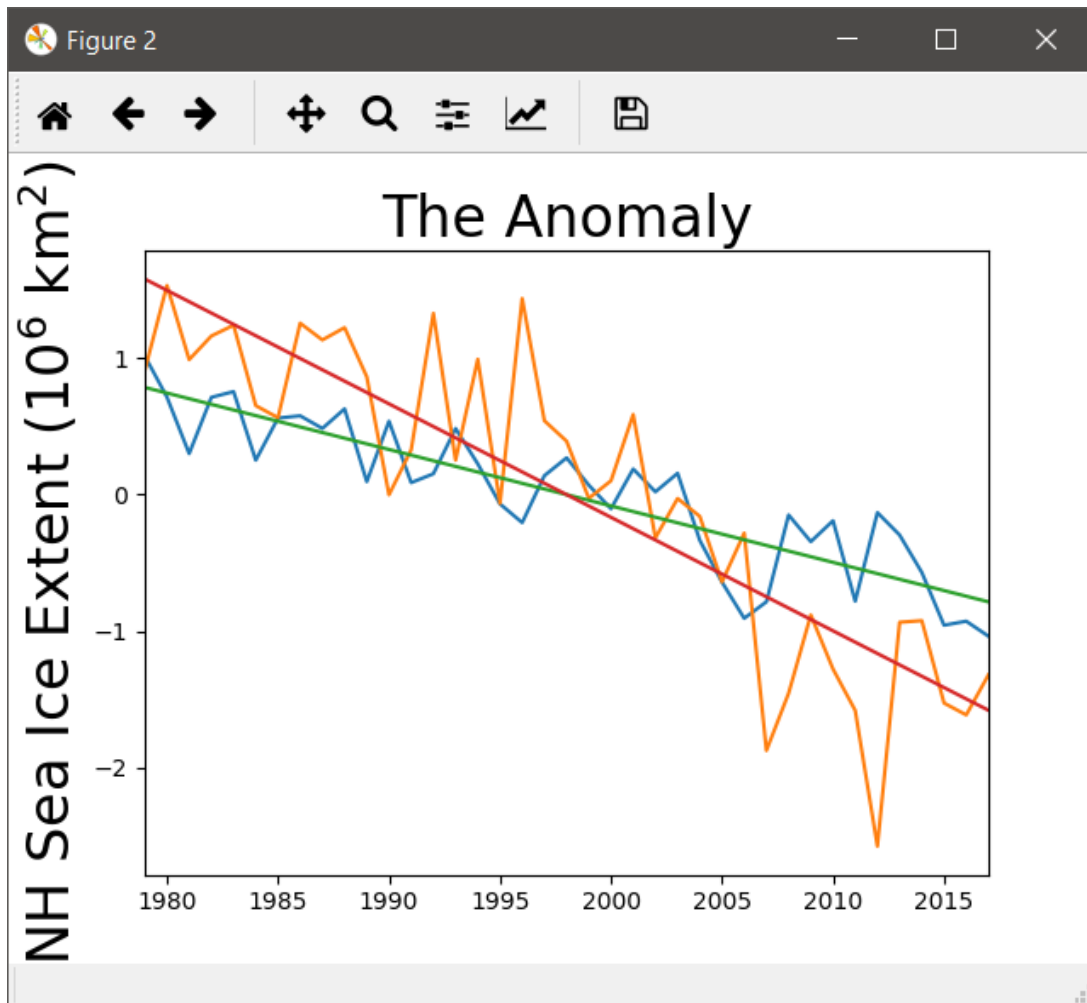
`make_fig_1`: This function takes a March-September frame (the return value from the first function), and creates a figure that looks like the image below. The rubric for this figure:

- +1 each: x and y tick labels are correct.
- +3 each: the two curves look like the image.
- +4: the y-axis title looks like the image. Superscripts required for any title points.
- +4 each: the two lines look like the image.



make\_fig\_2: This function takes a March-September frame, and creates a figure that looks like the image below. The rubric for this figure:

- +1 each: x and y tick labels are correct.
- +3 each: the two curves look like the image.
- +2: the y-axis title looks like the image. Superscripts required for any title points.
- +2: the title looks like the image.
- +4 each: the two lines look like the image.



main: Get the March-September frame. Get your OLS parameters for the four curves. Make your predictions for winter and summer, printing a blank line between them. Make your figures and call `plt.show()` to draw them. If the predictions don't print before the plots show up when main is run, there is a 20-point penalty.