HOMEWORK #5
MSBA 201

This homework is intended to familiarize you with the NumPy and Pandas libraries in Python.

This assignment will be different from other assignments in that it is more exploratory than it is an attempt to make a functioning program given specified principles. We are attempting to use these libraries to explore the datasets. NumPy and Pandas can be used both for exploratory code as well as self-contained programs.

You may wish to use Jupyter Notebook, due to its interactive nature. If you use Jupyter Notebook:

- Split up each part of each problem into its own cell.
- Label each cell with code in it. You can do this by making a cell that does nothing but hold a label. This is called a markdown cell. You can make a markdown cell by adding a cell using the Insert menu option that is appropriate, select the cell you just made, and then going to the Cell menu, selecting Cell Type, and choosing Markdown.
- Make certain that ALL CODE EXECUTES IN ORDER before you submit it. You can make sure of this by going to the Kernel menu, and then clicking Restart and Run All. Note that this will clear out all of your output from previous cell execution, and will also wipe out all variables, so your notebook will work as though all cells are running for the very first time.
- Remember that in Jupyter Notebook, variables stick around during a session. So if you were to write code into a cell such as ‘total += 5’, ***<u>depending on the rest of the code</u>***, every time you ran that cell total would increase by 5. Keep this in mind if you run into problems. Don’t be afraid of using the Restart and Run All option mentioned earlier to check for this problem.
- When you are asked to print out something you may either:
  a) use the print() function, or
  b) use the variable name or function call as the last line of the cell, which will output what is requested. If you do this, don’t forget to assign results to variables when you are asked to do so.
- Submit your Jupyter Notebook file. Do not submit a text file with the code in it.

You can also treat the assignment like a regular program, as we have been doing for the last several assignments. If you write a regular program, you must use print() whenever you are asked to print out something. You must also split up the parts of each problem by printing out which part the current output belongs to.

Be certain to save any output that is reused to a variable, as described in the instructions.

PLEASE NOTE: Because we are trying to exercise how to use NumPy and Pandas, many things we do will be printed out and never used afterwards, so you can expect this to show up at least a few times.

PROBLEM 1. USING NumPy:

i) Connect the code to the NumPy library, as I have demonstrated in class or as in the book. After this, paste the following code:

```
np.set_printoptions(suppress=True, precision=3)
sourceList = [2.2587, 295, 1.3422, 183, 0.6159, 50, 0.4887, 14,
1.1474, 90, 7.5492, 2969, 2.0532, 374, 0.6363, 67, 3.1504, 378,
10.3136, 1886]
```

This will give you the basic data that we are going to use in the form of a list, and will also alter the output so that it will not use scientific notation when you print out results.

ii) Create an array named myData using the sourceList data. Print out the array.

iii) Shape the array. MAKE SURE THE METHOD THAT YOU USE CHANGES myArray. Don't create a view! Print out myData after the resizing is complete. There should be 10 rows and 2 columns.

iv) Print out the number of dimensions and the dimensions (shape) of the array, using the attributes on myData. DO NOT simply insert these into the document; you must EXCTRACT the data from the array object, using the myData variable. If you do not know what I am talking about, review your slides or the book. You may want to just use print() for this step, even if you use Jupyter Notebook, since you have to print two things out.

v) Using a nested loop, print out every element in the array, along with its coordinates within the array. You should use print() for this even if you are in Jupyter Notebook. There are a few ways to do this. If you do not use numbered for loops, you may want to use separate counter variables to keep track of where you are in the array.

vi) Using data slicing, extract the second, third and fourth rows of data from myArray. Put this in an array named 'extractedArray'. Print this array.

vii) Transpose this extraction, and bind the resulting object to the 'transposeExtract' variable. Print transposeExtract.

viii) Multiply transposeExtract by 10 using broadcasting, and bind the results in the variable mulTransposeExtract. Print mulTransposeExtract.

ix) Add together transposeExtract and mulTransposeExtract using the appropriate operator, and bind the result in alteredTransposeExtract. Print alteredTransposeExtract.

x) Use stacking to combine transposeExtract and alteredTransposeExtract into one array, such that transposeExtract is on top and alteredTransposeExtract is on the bottom. Bind the result to a variable named bigArray. Print bigArray.

xi) Use the negative ufunc to make all the values in bigArray negative. Bind the result to a new variable, negBigArray. Print negBigArray.

xii) Obtain the standard deviation OF THE POPULATION (consult the slides – this requires an additional parameter) of myArray for EACH COLUMN. This should result in a 2x1 matrix. Bind it to the variable 'arrayStdDev' and print it out.

PROBLEM 2: USING NumPy AND Pandas:

i) Write code to connect your code to both the NumPy and Pandas libraries.

ii) Have Pandas access the data from the following source:
https://vincentarelbundock.github.io/Rdatasets/csv/AER/EuroEnergy.csv

Note that you SHOULD NOT download this, at least not in your final code. Furthermore, you need to add a parameter to indicate that the first column of data consists of row labels (indexes). Bind the result into the variable EuroEnergy. You should now have a dataframe bound into the EuroEnergy variable. Print EuroEnergy.

iii) Iterate through the index and column attributes, printing out each one. This will require a loop. You should use print() for this, eve if you are using Jupyter Notebook.

iv) Using slicing, extract the column "gdp" and bind the result to a variable named gdpData. USE THE COLUMN NAME. Use the method recommended in the book and slides to avoid using more memory. Print gdpData.

v) Using slicing, extract the column "energy" and bind the result to a variable named energyData. USE THE COLUMN NUMBER. Use the method recommended in the book and slides to avoid using more memory. Print energyData.

vi) Get the descriptive statistics (using a single method call) from gdpData, and print it.

vii) Get the descriptive statistics (using a single method call) from energyData, and print it.

viii) Get the SAMPLE standard deviations for the columns in EuroEnergy. Print these out.

ix) Using data slicing, extract the data in the following rows: Sweden, Italy, Switzerland, Denmark, UK. Bind the result to a variable named euroExtract, and print out euroExtract.

x) Sort euroExtract,, WITHOUT making a copy, using the value in the GDP column. The order should be descending (largest first). Print euroExtract again.

xi) Sort euroExtract, returning a copy, using the value in the Energy column. The order should be ascending (smallest first). Bind the copy to a variable named sortedByEnergy. Print sortedByEnergy.

xii) Use slicing to extract the top half of the GDP column from EuroData. Note that you should not sort, just extract the top ten elements in the order that it is presently in. Bind the result of this extraction to a variable, topGDP. Then bind the bottom half of the GDP column into a variable, bottomGDP. Extract the underling NumPy arrays, and compare the array of topGDP to the array of bottomGDP using the > operator. Note that attempting to compare the extractions using Pandas data types will result in an error as the two extractions don't have identical labels. Bind the result of the comparison to a variable, topBottomCompare, and print it out. Note that this will be a NumPy data structure.

xiii) Get the average (mean) of the energy in energyData. Store it in a variable named averageEnergy. Then use Boolean indexing to select the rows in EuroData with a value for the energy column that are greater than or equal to averageEnergy. Name the output of this highEnergyConsumers, and print out highEnergyConsumers. HINT: Try using 'EuroData.energy' as part of the comparison, since the data is multi-dimensional, unlike the single-dimensional book example.