# NumPy Techniques

## Overview

In this lab you’ll continue working with the Bergen 2019 weather dataset. You’ll make use of universal functions, calculate aggregations, explore broadcasting, and dig deeper into the meaning of the data by using Boolean logic. If time permits, you’ll get a chance to try out fancy indexing, partitioning, and sorting.

## Source folders

Student folder : C:\AdvPythonDev\Student\10-NumpyTechniques

Solution folder: C:\AdvPythonDev\Solutions\10-NumpyTechniques

## Roadmap

There are 8 exercises in this lab, of which the last two exercises are "if time permits". Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Using universal operators and functions
2. Specifying the data type for elements in a NumPy array
3. Creating a multidimensional NumPy array
4. Working with a multidimensional NumPy array
5. Making use of NumPy broadcasting capabilities
6. Using Boolean operations and aggregation on an array
7. (If time permits) Using Boolean masks on an array
8. (If time permits) Additional suggestions

## Familiarization

In the *student* folder, open processBergenData.py in a text editor. This file loads the Bergen 2019 weather data from a CSV file into a Pandas DataFrame object, and then assigns the precipitation column into a NumPy array. The NumPy array has shape (365,).

## Exercise 1: Using universal operators and functions

Universal operators and functions are NumPy array methods that automatically execute on each element in the array, without the need for you to write explicit looping code. Universal functions are more concise than loops, and far more efficient.

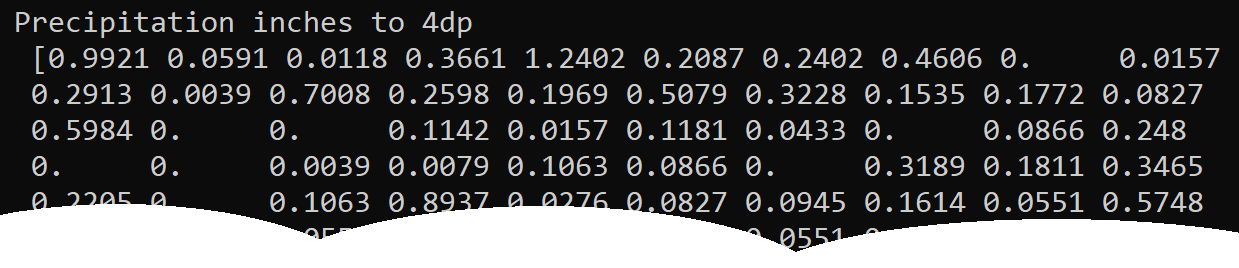
Using universal operators, convert all the precipitation values from millimetres into the following units. In each case, print out the new array values:

* Centimetres (to convert from mm to cm, divide by 10)
* Inches (to convert from cm to inches, divide by 2.54)

The inches values will be displayed to machine accuracy, e.g. 0.99212598. You don’t need this much accuracy; 4 decimal places is perfectly adequate. Think about how you can convert the values to 4 decimal places; if you’re thinking of using a loop, think again. You hardly ever use loops with NumPy arrays, because they are grossly inefficient. Instead, your first though should be “I wonder if NumPy has a universal function to do that…”. As it happens, NumPy has an around() universal function to round all values in an array to the specified number of decimal places. See here for details:

<https://docs.scipy.org/doc/numpy/reference/generated/numpy.around.html>

Use the around() function to round all the inches values to 4 decimal places, then print the array. It should look something like this:



When you’ve got all that working, combine your operations into a single statement that does all of the following:

* Converts the precipitation values from mm to inches, to 4 decimal places.

You can write arbitrarily complex statements using universal operators and functions 😊.

**Exercise 2: Calculating aggregations**

NumPy has a wealth of functions for calculating aggregation results. Aggregations are very important in data science, and you’ll use these functions a lot.

Using the precipitation data in inches (to 4dp), output the following aggregation information. Where indicated, round the result to 4dp via the round() standard Python function (why is it OK to use round() here, rather than the NumPy around() function?).

* Total annual precipitation, to 4dp (correct answer is 92.3893)
* Minimum precipitation on a day (correct answer is 0.0, as you might guess 😊)
* Maximum precipitation on a day (correct answer is 3.622)
* Mean daily precipitation, to 4dp (correct answer is 0.2531)
* Median daily precipitation, to 4dp (correct answer is 0.0669)
* Variance, to 4dp (correct answer is 0.1952)
* Standard deviation, to 4dp (correct answer is 0.4418)
* The quartiles (i.e. 25th percentile, 50th percentile, and 75th percentile). The correct answers are 0.0, 0.0669, and 0.315 respectively.

**Exercise 3:** **Creating a multidimensional NumPy array**

In Exercises 3 to 5, you’ll see how to create and use a multidimensional NumPy array. The array will have shape (365,2) and will contain the daily minimum temperatures (in column 0) and the daily maximum temperatures (in column 1).

At the moment, your code reads the data into a Pandas DataFrame and then plucks out the 'Precipitation' column into a NumPy array. The code looks like this:

precipitation = np.array(dataframe['Precipitation'])

It’s also possible to pluck out multiple columns from a Pandas DataFrame – you just specify an array of column names. You can then convert the multiple columns into a 2D NumPy array via the to\_numpy() function. For example, the following code plucks out the 'MinTemp' and 'MaxTemp' columns and converts them into a 2D NumPy array:

temps = dataframe[['MinTemp', 'MaxTemp']].to\_numpy()

Print out the shape of the temps array. It should be (365,2).

**Exercise 4:** **Working with a multidimensional NumPy array**

Create an array named diurnal\_ranges that contains the diurnal temperature range for each day (i.e. the maximum temperature minus the minimum temperature for each day). The diurnal\_ranges array will have shape (365,).

Using universal operators, get the following subarrays and print them:

* Diurnal ranges > 8 degrees Celsius
* Diurnal ranges < 4 degrees Celsius

**Exercise 5: Making use of NumPy broadcasting capabilities**

In this exercise you’ll use NumPy broadcasting capabilities to perform binary operations on arrays of different shape. You might like to take a moment to remind yourself about how broadcasting works before you tackle this exercise.

The first step in this exercise is to calculate the following values:

* The mean value in the “minimum temperature” column in the temps array
* The mean value in the “maximum temperature” column in the temps array

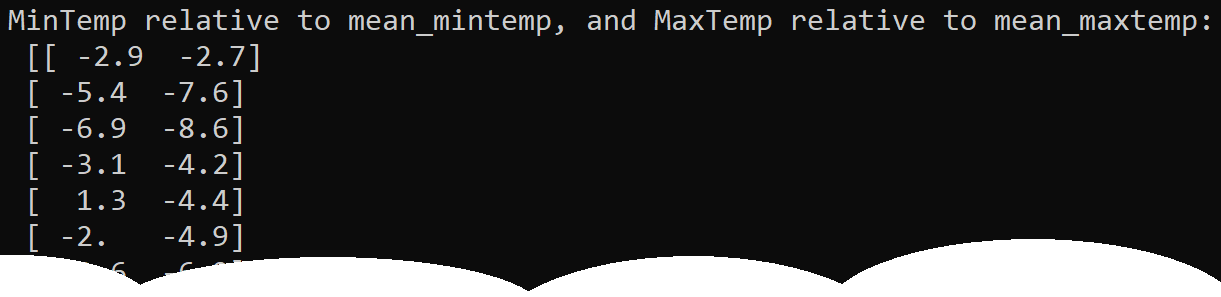
Now create a NumPy array containing these two values. Print the array values – they should be [6.0 12.4], i.e. the mean minimum temperature over the year was 6, and the mean maximum temperature over the year was 12.4. Also print the array shape – it should be (2,).

The next step is to see how the temperatures for each day compared to these means. For example, consider the temperatures for the first day of the year:

* For the first day of the year, the minimum temperature is 3.1. Compared to the mean minimum temperature of 6 for the whole year, this day is 2.9 cooler.
* For the first day of the year, the maximum temperature is 9.7. Compared to the mean maximum temperature of 12.4 for the whole year, this day is 2.7 cooler.
* Thus, the result we’d like to obtain for the first day is [-2.9 -2.7].

Now think how you can perform this calculation for every day of the year – i.e. take the temps array of shape (365,2) and subtract the array containing mean min and max temps of shape (2,). These arrays have different shape, but NumPy’s broadcasting rules allows you to do it.

Get this working and be sure you understand what’s happening. The results should look like this:



**Exercise 6: Using Boolean operations and aggregation on an array**

For the remainder of the lab, you’ll switch back to using the *precipitation* data, i.e. the 1D array of shape (365,).

Perform the following Boolean operations on the *precipitation* data. In each case, the result is a 1D array of shape (365,) containing True or False values for each day:

* Was there any precipitation on a day?
* Was there between 1 inch and 2 inches of precipitation on a day?

Now perform the following Boolean aggregation on the *precipitation* data:

* Were there any days with more than 2 inches of precipitation? The result should be a single Boolean, i.e. True of False, that represents the overall result for the whole year.
* How many days had more than 2 inches of precipitation during the whole year? The result should be an integer (the correct answer is 4 days).

**Exercise 7 (If time permits): Using Boolean masks on an array**

Use a Boolean mask on the *precipitation* data to get an array of days of precipitation, i.e. just the days where the precipitation is nonzero. Print out the following info (NumPy arrays have a size property you can use here):

* The number of days of precipitation (the correct answer is 258)
* The number of days of no precipitation (the correct answer is 107)

Now imagine a city has 365 inches of rain across the whole year, but bizarrely it all falls on a single day. The mean rainfall across the whole year is 1 inch per day, but the mean rainfall across just the rainy days is 365 inches. As you can see, eliminating zero-precipitation days can give a different insight into the data.

With this in mind, print the following information to see precipitation statistics across the whole year, compared to precipitation statistics on just the rainy days:

* Mean daily precipitation all year, compared to that of rainy days only.
* Median daily precipitation all year, compared to that of rainy days only.
* Variance all year, compared to that of rainy days only.
* Standard deviation all year, compared to that of rainy days only.

**Exercise 8 (If time permits): Additional suggestions**

* Use fancy indexing to pluck out the first and last days of precipitation.
* Partition the rainy days into the 30 least wet days, then the rest.
* Sort the rainy days.