# Getting Started with NumPy

## Overview

In this lab you’ll load data into a NumPy array from a CSV file (and optionally an Excel spreadsheet). The dataset contains temperature and precipitation measurements for every day in 2019 for Bergen. You’ll use various NumPy array capabilities to slice and dice this data in your Python code.

## Source folders

Student folder : PythonDS/Student/01-NumpyGettingStarted

Solution folder: PythonDS/Solutions/01-NumpyGettingStarted

## Roadmap

1. Loading data into a NumPy array
2. Specifying the data type for elements in a NumPy array
3. Indexing and slicing an array
4. Reshaping an array
5. (If time permits) Splitting an array
6. (If time permits) Combining arrays
7. (If time permits) Plotting arrays

## Familiarization

In the *student* folder, open BergenWeather2019.csv in a text editor. This file contains real temperature and precipitation measurements for every day in Bergen in 2019. **All weather measurements are from Yr, delivered by the Norwegian Meteorological Institute and NRK**.

The CSV file has 366 rows – the first row is the column headings, and the other 365 rows contain the data for every day from 1 January to 31 December. Each row has 4 values:

* The DayOfMonth column indicates the day number in a month (notice how this value wraps from 31 back to 1 as we move from January to February, for example).
* The MinTemp column contains the minimum temperature for a day, in degrees Celsius.
* The MaxTemp column contains the maximum temperature for a day, in degrees Celsius.
* The Precipitation column contains the precipitation for a day, in mm.

The *student* folder also has an Excel spreadsheet named BergenWeather2019.xlsx, which contains exactly the same data but in Excel format.

**Exercise 1: Loading data into a NumPy array**

In this exercise you’ll write a Python script to load data from the CSV file (and optionally the Excel spreadsheet) into NumPy. Note that NumPy doesn’t have any functionality to load data from CSV/Excel files – instead, you need to use Pandas functions such as read\_csv() and read\_excel()…

In the *student* folder, create a new file named processBergenData.py in a text editor. Add code to load the data from the CSV file into a Pandas DataFrame. Print the shape of the DataFrame – this should be (365,4), i.e. it contains 365 real rows of data, and there are 4 columns per row.

The DataFrame contains all the rows and columns from the CSV file. We’ll just focus on the precipitation column for now, so extract this column in a NumPy array.

Print the following information about the NumPy array of precipitation data:

* The shape of the array – this should be (365,)
* The data type of elements in the array – this should be float64
* The data itself – this should display as a big array of 365 values

If you have Excel loaded on your machine, you can easily tweak your code to read data from the Excel spreadsheet rather than from the CSV file:

* Call read\_excel() rather than read\_csv()
* Specify the Excel filename, rather than the CSV filename (obviously!)

**Exercise 2:** **Specifying the data type for elements in a NumPy array**

All the elements in a NumPy array must be the same data type. By default, NumPy automatically determines the appropriate data type when it loads the data. In the case of the Bergen precipitation array, NumPy decides float64 is the best data type because the values are fractional.

Sometimes you want to force NumPy to use a particular data type. For example, in some cases it might be perfectly fine to work with integral data rather than floating point data (integers are smaller and faster than floats).

Experiment with specifying various data types when you load the Bergen precipitation array, by specifying a dtype parameter in the call to read\_csv() or read\_excel(). Note there are two ways to specify a NumPy data type:

* As a string literal, e.g. 'float64'
* As a NumPy type, e.g. np.float64

**Exercise 3: Indexing and slicing an array**

Add code to print the following information about Bergen precipitation:

* Precipitation on 1 January
* Precipitation on 2 January
* Precipitation on 31 December
* Precipitation on 30 December
* Precipitation on every day in January
* Precipitation on every day in December
* Precipitation on every day in November
* Precipitation on every 10th day

**Exercise 4: Reshaping an array**

A common requirement when working with datasets is that you want to reshape a linear 1D array into multiple dimensions.

For example, the Bergen precipitation data is a 1D array of 365 values, but you might want to process it as 52 weeks x 7 days. This would make it easier to perform tasks such as comparing rainfall values for every Saturday, calculating the ratio of rainfall during weekdays compared to weekends, etc.

When you reshape an array, you have to make sure the number of elements in the original array is compatible with the shape you’re aiming at. For example, you can’t reshape a (365,) array into a (52,7) array because 52 x 7 = 364, not 365…

Thus, the first step is to create a NumPy array that contains just the first 364 values of data (for the purposes of this exercice). Once you have this, reshape it into a 2D array containing 52 rows and 7 columns per row. Print out the 2D array and its shape.

Then print the following info:

* Precipitation for the first week.
* Precipitation for the last week.
* Precipitation for the first weekend. (For simplicity, pretend 1 Jan 2019 was a Monday, which means the Saturday and Sunday in any week would be days 5 and 6. In actual fact, 1 Jan 2019 was a Tuesday not a Monday, but let’s pretend this is fake news!)
* Precipitation for the weekdays in the final week (again, pretend Monday is day 0 in a week, Tuesday is day 1 in a week, etc.)

**Exercise 5 (If time permits): Splitting an array**

Another common requirement when working with datasets is to split it into smaller chunks, for simpler algorithms and faster execution.

With this in mind, split the weekly precipitation array vertically into 3 subarrays as follows. Print the shape of each of these subarrays, plus the data:

* Quarter One (i.e. weeks 0 to 12 inclusive). The shape should be (13,7).
* Quarter Two (i.e. weeks 13 to 25 inclusive). The shape should also be (13,7).
* Quarters Three and Four combined. The shape should be (26,7).

Now have a go at splitting the weekly precipitation horizontally into 2 subarrays as follows. Print the shape of each of these subarrays, plus the data:

* Weekdays (i.e. columns 0 to 4 inclusive, using our (fake!) assumption that day 0 in every week is a Monday). The shape should be (52,5).
* Weekends. The shape should be (52,2).

**Exercise 6 (If time permits): Combining arrays**

The opposite of splitting an array is to combine arrays together. NumPy has three functions for combining arrays:

* concatenate() – if the arrays have the same shape
* vstack() – vertically stack, if the arrays have the same number of columns
* hstack() – horizontally stack, if the arrays have the same number of rows

Using these functions as appropriate, combine the following subarrays together to create a (52,7) array again. Print the resulting array, plus its shape, to verify it’s all correct:

* Quarter One
* Quarter Two
* Quarters Three and Four combined

Also combine the following subarrays together to create a (52,7) array, and print its shape and data:

* Weekdays
* Weekends

**Exercise 7 (If time permits): Plotting arrays**

Using MatPlotLib, plot the precipitation in 3 different ways:

* As a graph (use the plot() function in MatPlotLib)
* As a bar-chart (use the bar() function in MatPlotLib)
* As a histogram (use the hist() function in MatPlotLib)

For more information about how to use these functions, see online documentation here:

<https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html>

The plots should look something like this (note we’ve set x/y labels and a fancy title in the *solution* code):





