Doctoral Academy Python **Worksheet 1**
12th June 2018

Overview

This lab is an introduction to using Python for analysing and visualising datasets. This lab sheet has been adapted from one written by Dr Matthew Williams for the CMT206 Human Centric Computing module.

1 A simple example

We'll now try some basic coding in Python.

We need to import some of the modules we'll be using:

```
import matplotlib.pyplot as plt
import numpy as np
```

We won't use any real-world data yet. Let's instead create 100 random values and then print them out:

```
data = np.random.rand(100)
print(data)
```

Now let's do a visual inspection of the data. For the moment we'll use a simple scatter plot. We'll use our random points as y-axis values. The x axis will be integer indexes from 0 onwards.

```
x = np.arange(100)
fig = plt.figure()
plt.scatter(x, data)
plt.show()
```

Our x-axis values are 0, 1, ..., 99 (see `x = np.arange(100)`). You should now see a scatter plot showing the random values we created earlier. You can save this for use elsewhere by clicking the 'save' icon in the window that appears. Save the plot now.

Exercise 1.1. Re-run the code from above. Is the plot different to the one you saved earlier? If so, why?

Exercise 1.2. Edit the visualisation to plot 250 random values rather than 100.

2 Exploring the effectiveness of antibiotics

We'll now move on to exploring a real dataset.

Dataset

For the rest of the exercises you will need to have downloaded the course resources from https://github.com/martinjc/doctoral-academy-python-advanced. Make sure you have unzipped this into the same directory as you are currently working in in Python.

> You can use the `os` module to find out which directory you are working in, and to change directories

Make sure you have the file 'antibiotics.csv' in your working directory.

Open the file in a text editor. The data is in **comma separated values** (CSV) format. This is a textual representation of a spreadsheet. Each line represents a row in the spreadsheet. The values (or columns) on a particular line are separated by a comma.

This dataset was collected in 1951 by Will Burtin, who was investigating the effectiveness of three antibiotics (penicilin, streptomycin, and neomycin) on 16 species of bacteria. The first row in the file gives the headings of the four columns: *Bacteria*, *Penicillin*, *Streptomycin*, and *Neomycin*. The Bacteria column (the values before the first comma) are the names of the species of bacteria. Below the remaining headings are 'MIC' values. The MIC gives an indication of the effectiveness of a particular antibiotic at preventing the growth of a particular bacteria. A small MIC indicates that the antibiotic is more effective at combating a particular bacteria.

Analysing the data

Let's plot a bar chart to show how well penicillin combats each of the 16 species of bacteria.

We'll do our usual set up:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

This time we've also imported *pandas*, as we'll be using it to load the CSV file into Python.

To load and print the data:

```
data = pd.read_csv('antibiotics.csv')
print(data)
```

We can also look at specific columns and cells from the table:

```
print(data['Bacteria'])  # list all the species

print(data['Bacteria'][0] ) # show the first species

print(data['Bacteria'][0:5])  # list the first five species

print(data['Penicilin'])  # list the MICs for Penicillin
```

pandas provides some plotting functionality; for example, to plot a bar chart from all columns:

```
data.plot(kind='bar')
plt.show()
```

However, *pandas*'s features are too restrictive for some visualisations we will be building. For the rest of this lab we'll be focusing on *matplotlib*'s plotting library.

If you're interested, you may want to have a look at panda*'s plotting examples to see what it offers.*

To make our analysis easier we're going to pull out the bacteria column and the penicillin column and give these data their own variables:

```
bacteria = data['Bacteria']
penicilin_MICs = data['Penicilin']
```

We'll also find out how many rows of data there were (this should be 16 – one for each species) and give these indexes from 0:

```
n = len(penicilin_MICs)  # the number of values we're plotting
indexes = np.arange(n)
```

We can then build a basic bar chart of penicillin MIC values with the code in the following cell:

```
fig = plt.figure()  # create a new canvas to plot onto
plt.bar(indexes, penicilin_MICs, align='center')
plt.show()
```

There are a few issues with this chart: the two axes have no labels, the chart doesn't have a title, and the names of the bacteria are not given.

We'll begin by fixing the labels and title. We can do this through an *axes* object, which is a matplotlib object that represents a particular plot. Note the difference between an *axes* object and a *figure* object. An *axes* object represents the area on which a single plot is placed. A *figure* object

is the canvas on which multiple *axes* (i.e., plot areas) can be placed. So far we've only placed one plot on a figure, but in future we'll see how we can add many to create composite visualisations.

We'll use the `add_axes` method to explicitly create a new axes on the current figure. This allows us to modify the axes object to add labels and a title.

```
fig = plt.figure()  # create a new canvas to plot onto
ax = fig.add_axes([0.1, 0.1, 0.8, 0.8])

ax.bar(indexes, penicilin_MICs, align='center')
ax.set_xlabel('Species')
ax.set_ylabel('Penicilin MIC')
ax.set_title('Effectiveness of Penicilin against Bacteria')
plt.show()
```

The argument passed to `add_axes` is a list of four values: `[left, bottom, width, height]`. *left* and *bottom* specify the gap from the left and bottom where the plot should begin. *width* and *height* specify the width and height of the plot area. These are all measured as fractions of the figure's overall width and height. We chose `[0.1, 0.1, 0.8, 0.8]` in this example because our single plot is going to take up most of the figure, but still leaving a little space on each side; in particular, it'll leave 0.1 (10%) of the figure on each side.

Finally, we need to swap the integer x-axis values with the names of the bacteria:

```
fig = plt.figure()  # create a new canvas to plot onto
ax = fig.add_axes([0.1, 0.1, 0.8, 0.8])

ax.bar(indexes, penicilin_MICs, align='center')
ax.set_xlabel('Species')
ax.set_ylabel('Penicilin MIC')
ax.set_title('Effectiveness of Penicilin against Bacteria')

ax.set_xticks(indexes)  # force all x-axis ticks to appear
ax.set_xticklabels(bacteria, rotation='vertical')
    # use bacteria names instead of integers
plt.show()
```

Exercise 2.1. Update the bar chart so that the bacteria are sorted by MIC.
Hint: The easiest way to do this is to use *pandas* to sort the rows before you pull out the individual columns. See: sort in the pandas documentation.

Exercise 2.2. Produce the same bar chart of MICs for streptomycin. How does its antibiotic performance compare to penicilin?

The following snippet shows how two axes can be placed side-by-side:

```
fig = plt.figure()

ax1 = fig.add_axes([0.1, 0.1, 0.2, 0.8])
ax2 = fig.add_axes([0.4, 0.1, 0.2, 0.8])

ax1.scatter([1, 2, 3], [1, 2, 3])
ax2.scatter([3, 2, 1], [1, 2, 3])
```

Exercise 2.3. Produce a new visualisation with the bar charts for streptomycin and penicilin placed side-by-side.

Exercise 2.4. Investigate the `figsize` parameter of *matplotlib*'s figure function. Use it to increase the overall width of the canvas for your side-by-side plot. Save your figure to disk and view it in an image viewer.

3 Coffee sales in the US

We'll now try some more-complex data processing.

Dataset

Make sure you have the file 'coffee.csv' from the course resources in your working directory. This is a fabricated dataset of coffee sales for a hypothetical national-wide coffee company. This covers one day of sales. The sales are aggregated on a per-state basis. Open the file in a text editor to get an idea of what the data look like. Each row in the file gives the sales for a particular type of drink in a particular state.

An example row is as follows:

```
sales  profit   region   state  category  type      caffeination
219    94       Central  CO     Coffee    Amaretto  Regular
```

This represents the sales and profit (both in $1000s) of the 'Amaretto' drink in the company's shops across Colorado (`CO`). `region` is additional information stating the broader geographic region the state belongs to. `category` and `caffeination` provide further information about this particular drink.

Analysing the data

To begin we import the relevant modules and read the CSV file:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

data = pd.read_csv('coffee.csv')
```

In a moment we'll compare the nation-wide sales of two drinks: Columbian and Caffe Mocha. First, let's look at how *pandas* represents data...

Exercise 3.1. In *pandas*, the data are read into a *DataFrame* object. This provides many tools for performing data filtering, sorting, and more.
Investigate the list of methods that can be called on a *DataFrame* by looking at the *DataFrame* documentation. Try the following methods on the coffee data and print the result to see what they do.

(a) List the column headers:
`data.keys()`
Extra question: how would you turn this into a Python list object?

(b) Get the unique values contained in a column:
`data['state'].unique()`

(c) Delete a column:
`data.drop('category', 1)`

(d) Build a new DataFrame that contains a subset of the columns:
`data.reindex(columns=['state', 'sales'])`

Exercise 3.2. As we've seen before, we can pull out the values of a particular column; for example, to get all the sales values:

```
sales = data['sales']
```

or only the first few values:

```
sales = data['sales'][:5]
```

In both cases the sequence (or subsequence) of values is represented by a *Series* object. Investigate the methods of *Series* in the documentation. Try out the following methods on the coffee data:

(a) Addition and multiplication:
`sales.add(55)`
`sales.mul(2)`

(b) Sum, average, and standard deviation:
`sales.sum()`
`sales.mean()`
`sales.std()`

(c) Minimum and maximum:
`sales.min()`
`sales.max()`

To compare the two drinks we've chosen we're going to need to pull out their particular rows from the data using *pandas*'s row select syntax:

```
columbian_rows = data[ data['type'] == 'Columbian' ]
mocha_rows = data[ data['type'] == 'Caffe Mocha' ]
```

Print these two new variables to check that each only contains rows for its corresponding drink. We should also check the states where these products were sold, as this might affect our analysis:

```
print(columbian_rows['state'].unique())
print(mocha_rows['state'].unique())
```

To compare the performance of these two products we're going to create a **box plot**. This is a visual way of describing the distribution of data. In particular, it's a compact means of showing the median, interquartile range, and maximum and minimum values.

Currently, we're only interested in the profit columns, so we'll filter the other ones out:

```
columbian_profits = columbian_rows.reindex(columns=['profit'])
mocha_profits = mocha_rows.reindex(columns=['profit'])
```

Before we start our visual analysis, let's have a look at the two drinks' median sales across all states:

```
print(columbian_profits.median())
print(mocha_profits.median())
```

According to median profit, Columbian outperforms Caffe Mocha by $8,000. However, this doesn't tell us anything about the variation in the drinks' sales across the states. We can do this through visual analysis by producing a box plot for each drink.

matplotlib's `boxplot` method handles this for us. To use it, we just need to provide a list of *pandas* `Series` objects; e.g.,

```
ax.boxplot([columbian, mocha])
```

Here is a full example:

```
fig = plt.figure()
ax = fig.add_axes([0.1, 0.1, 0.8, 0.8])

ax.boxplot([columbian_profits, mocha_profits])

ax.set_xlabel('Drink')
ax.set_ylabel('Profit (\$1,000s)')
ax.set_title('Profit Comparison Across States')

# Replace the integer tick labels with the names of the drinks:
ax.set_xticks([1, 2])
ax.set_xticklabels( ['Columbian','Caffe Mocha'] )
plt.show()
```

Note: unlike `bar`, `boxplot` *uses x-axis labels from 1 rather than 0.*

How does the visual analysis influence your interpretation of the data?

Exercise 3.3. Investigate and try out the following `axes` methods that allow you to further customise the appearance of a figure (see the axes documentation):

(a) Specify the font size of various text elements:

```
ax.xaxis.label.set_fontsize('x-small')
ax.yaxis.label.set_fontsize('xx-small')
ax.title.set_fontsize('small')
ax.tick_params(axis='x', labelsize='x-large')
ax.tick_params(axis='y', labelsize='xx-large')
```

What other font size specifiers are available?

(b) Completely hide the x- and y-axis:
`ax.set_axis_off()`

(c) Hide the x-axis only:
`ax.get_xaxis().set_visible(False)`

(d) Set the range of values that the y-axis displays (also called the axis *limits*):
`ax.set_ylim(bottom=-300)`
`ax.set_ylim(top=1000)`

(e) Add a text annotation:
`ax.annotate("Here's an outlier", xy=[1,400])`
Note: `xy` *specifies the coordinates of the point that's being annotated.*

(f) Add a text annotation with an arrow:

```
ax.annotate("Here's an outlier", xy=[1,400],
  xytext=[1.5,400],
  arrowprops={'arrowstyle':"->", 'connectionstyle':"arc3"} )
```

Note: `xy` specifies the coordinates of the point that the arrow will point to; `xytext` specifies where the text will be placed, which is also where the arrow will start.

Exercise 3.4. Update your visualisation to include a third drink product, 'Amaretto', in the comparison.

Exercise 3.5. Going back to your original box plot figure, extend it to include box plots for all drink types. *Hint: Do **not** do this by laboriously copying and editing the same code for each box plot. Think about how you can elegantly code this in as few lines as possible. Consider using dictionaries, lists, loops, and/or defining your own functions.*

Exercise 3.6. Produce a new figure that compares the profits from caffeinated versus de-caffeinated drinks across states. Your figure should contain two box plots, one for caffeinated profit and one for de-caffeinated profit. Don't forget the dataset includes multiple types of caffeinated (and de-caffeinated) drinks for the same state. You'll need to aggregate these on a per-state basis so that you have the total caffeinated (and de-caffeinated) profit for each state.