Aston University
Machine Learning

Getting Started with Python

Learning Outcomes:
In this module, you will be expected to engage in practical work using Python and to submit the majority of your assessed portfolio work in Jupyter notebook format. This worksheet is not designed to be a comprehensive guide to Python – and there is material on Blackboard which will give you a fuller introduction – but is designed to get you started with some of the key components of the language which we will use in this module.

Instructions:
Download the file lab1data.csv from Blackboard. It contains a set of 100 pairs of points. We are going to use python to plot a graph of those points with a line of best fit through them.

If you are using your own machine and haven't done so already, install Python (see the Python material on Blackboard for recommendations on how to do so).

Task 1 – Opening a Jupyter notebook:
Create a folder to contain the work for this lab and add lab1data.csv to it. Open the Jupyter notebook app (e.g. through the start menu or, in the Anaconda prompt by using the command `jupyter notebook`) and, in the interface, navigate to the folder you just created. Create a new Python 3 notebook.

In the Jupyter notebook environment, we have a series of cells into which we can type python code. We can then click on a "Run" button to run this code and see its output. To check that you can do this, try and print the text "Hello, World!" using your first cell.

```
print("Hello, world!")
```

If you have done this correctly, you will see the text printed out below the cell and then a new cell created below it.

Task 2: Importing modules:
For many of the tasks in this module, you will need to use functionality from various Python modules. We will import a few here. Enter the following into your next cell and run it:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

Note that we are importing code in three different ways above. The first line (`import pandas as pd`) imports all the code in the module `pandas`, but allows us to reference it using the name `pd` (we will see an example of doing so in the next task). The second line (`import matplotlib.pyplot as plt`) imports a submodule `pyplot` of the module `matplotlib` and then allows us to reference it using the identifier `plt`.

The third line (`from sklearn.linear_model import LinearRegression`) does something slightly different. As in the previous example, `sklearn.linear_model` references `linear_model`, a submodule of the `sklearn` module. The `linear_model` module contains a large number of classes and functions. These classes/functions are not modules and so cannot be imported in the same was as a module
(e.g. `import sklearn.linear_model.LinearRegression`). The syntax we see on the third line allows us to import a specific named class or function from a module.

Task 3: Importing and accessing data
In machine learning, we train models on data. This means that we need to be able to access the data! Here, we will use the pandas library to import the data from the csv file we downloaded from Blackboard. In a new cell, enter and run the following:

```
df=pd.read_csv('lab1data.csv')
df
```

Let's break this down.

- Firstly, we are using the `read_csv` function from the pandas module (which we can access using identifier `pd` because of how we wrote the import statement in the previous task).
- We are pointing it at our data file using that file's name. This will only work if our Jupyter notebook is in the same directory as our file. If it isn't we will have to specify the relative path from our notebook to our file.
- `df=...` – `read_csv` returns a pandas dataframe. We are storing it here using name `df` for later use.
- `df` – the last line isn't doing any real work. However, if we have a variable on the last line of a cell in Jupyter notebook, it will be printed out in the line below the cell. This should allow you to inspect the structure that we have loaded.

We can access data in this data frame using the column headings (e.g. `df['x']`) or row ranges (e.g. `df[0:5]`). Play around with these yourself, printing out the results, until you are confident of how they work.

Task 4: Training a model using the data
Let us say that we want to predict the values of the 'y' column in our dataset using the values in our first, 'x', column. It is common practice to pull our dependent (the variable we are trying to predict) and independent variables (the variables we are basing our prediction on) into different python variables – traditionally given names `y` and `X` respectively.

We should already know how to do this for the dependent variable:

```
y = df['y']
```

However, for our independent variable, the story is not quite as simple. In this example, we are trying to predict `y` from a single independent variable but, often, we will want to base our predictions on multiple independent variables. The class we will use to train our model is built to accommodate multiple independent variables, so requires us to input our `X` values as a 2D array. As in many other programming languages, a 2D array in Python is just an array in which

every element is an array. If we were to try and construct our `X` variable in a similar way to above:

```
X = df['x']
```

then we would have an array in which every element is a number, rather than an array. Instead, we can do as follows:

```
X = df['x'].values.reshape(-1,1)
```

where `values` pulls the data out of the pandas format and the `reshape` method reshapes the data into a 2D array. Try printing out the results of the two statements above to see the difference between them.

Now that we have data in the required format, creating and training our model is straightforward:

```
reg = LinearRegression()
reg.fit(X, y)
y_pred = reg.predict(X)
```

where the first line creates an instance of a linear regression model and the second line trains it on the data. Once trained, we can use the model to predict values, given some inputs. We do this on the third line.

Task 5: Plotting the results
We can plot our data points as well as the line derived from our linear model as follows:

```
plt.scatter(X, y,  color='red')
plt.plot(X, y_pred, color='blue', linewidth=1)
```

The first line should be self explanatory: it plots a scatter graph of `X` against `y` with red markers. The second line of code plots a line graph on the same axes. Here we plot `X` against `y_pred` – the predictions from our linear model – as these will form a straight line.

Task 6: Commenting on our results
In our assessed tasks, you will be asked to demonstrate that you both understand and can apply the techniques introduced in the course. Writing code is sufficient for the latter but, to demonstrate your understanding, you will also need to write some text. There are a couple of ways we can do this within a Jupyter notebook. If you want to add a short snippet of text to your code, then you can do so with a comment:

```
#In python, any text on a line following the "#" symbol is a comment.
```

However, this isn't a great way to include longer pieces of textual content. To do that you can change a cell's type. Click on an empty cell in your notebook and, in the menu at the top of the page, select Cell > Cell Type > Markdown. This changes the cell from a code cell to a markdown cell. If you know how to write markdown you can write it here, but the simplest effect of this change is that if you write plain text into this cell and then press "Run", the text will be rendered to the page. If you double click on the rendered text, you can change it. Try it for yourself!