NPA Data Science:

Computer Programming 3

michaelferrie@edinburghcollege.ac.uk

Part 1: Installing Modules

1.1 The base installation of Python is very lightweight, this means it does not take up a lot of space on your computer's hard disk or use a lot of resources to run. Sometimes we need to add some extra capabilities onto the basic installation of python, we can do this by installing modules.

The first module we need to install is called csv, to do this add the import keyword to the top of the python script and then the name of the module:

```
import csv
```

Sometimes additional modules need to be downloaded from the internet, modules can be installed in the same way as Jupyterlab was installed using pip (pip3 if you are on a Mac/Linux) or conda if you are using anaconda. To install a module from the internet you should run one of the following 3 package manager names then the name of the module:

```
{pip / pip3 / conda} install module_name
```

From this point forward if we say `pip install module_name` - we expect you to understand that you may have to write pip3 or conda, depending on your own system. One library we are going to make use of in this course is called pyplot which is part of a larger library called matplotlib, install this using:

```
pip install matplotlib
```

Open this week's Jupyter notebook and start running the code as you read these notes.

1.2 Let us say that we want to use a part of matplotlib called pyplot, so the part of the library we want is actually called matplotlib.pyplot, in order to import this we need to run:

```
import matplotlib.pyplot
```

Each time we want to use some functionality from this library we would have to type it out in full, and this would become quite an effort in a very large script, so we can use an alias to refer to the library. The alias could be any word but there are some conventions used. It is very common to `import matplotlib.pyplot as plt`, now we can refer to it using the alias name.

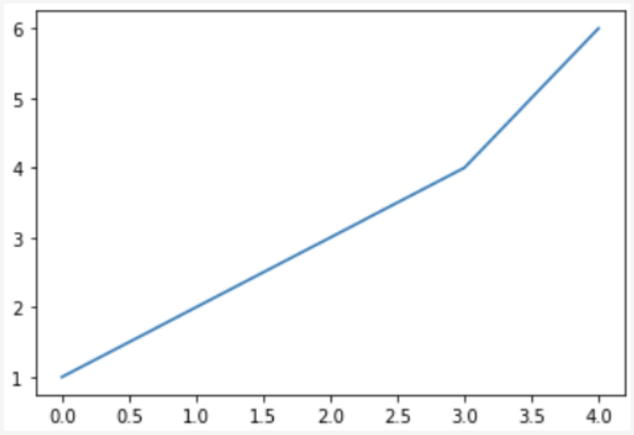
```
import matplotlib.pyplot as plt
```

Part 2: Creating the first plots

2.1 In last week's notes we learned that the list is a fundamental compound data structure in python, pyplot and lists work well together, run the code cell to create the following lists.

Consider the following example, we call pyplot (using the plt alias), then we want to use the plot function of the module, and then we want to send a list into this function as an argument. This is the most simple example of a piece of code that creates a plot.

```
plt.plot([1,2,3,6])
```



2.2 Hopefully you can see that the library takes care of the plot type, the graph size, the axes and the labels all by default. Obviously, we can customise all of these but this is the most simple example we can start with.

2.3 The plot function actually accepts 3 formal parameters in the following order: **(x, y, format)**. With x and y being the data to place on each axis of the plot and the format being the plot type. We could send in x and y as hard coded lists directly to the function as shown in 2.1, but with very large datasets this may not be practical, x and y can just be list variables and the lists can be specified elsewhere in the program.

This leaves only format to discuss, format is a shorthand combination of **{color}{marker}{line}**. There are many abbreviations of the shorthand, however the details of the matplotlib.pyplot.plot format shorthand are beyond the scope of this course, documentation is available to cover this in detail. Here is an example of some of the abbreviations available to us, this is called **'single-character-shorthand-notation'**.

Examples of single character shorthand notation for shades of colors:

'b' as blue

'g' as green

'r' as red

'c' as cyan

'm' as magenta

'y' as yellow

'k' as black

'w' as white

2.4 Create a plot and specify the X and Y arguments and provide a shorthand format notation to the plot function. First define two lists:

```
exam_score_maths = [1,2,3,3]

exam_score_english = [5,6,7,7]
```

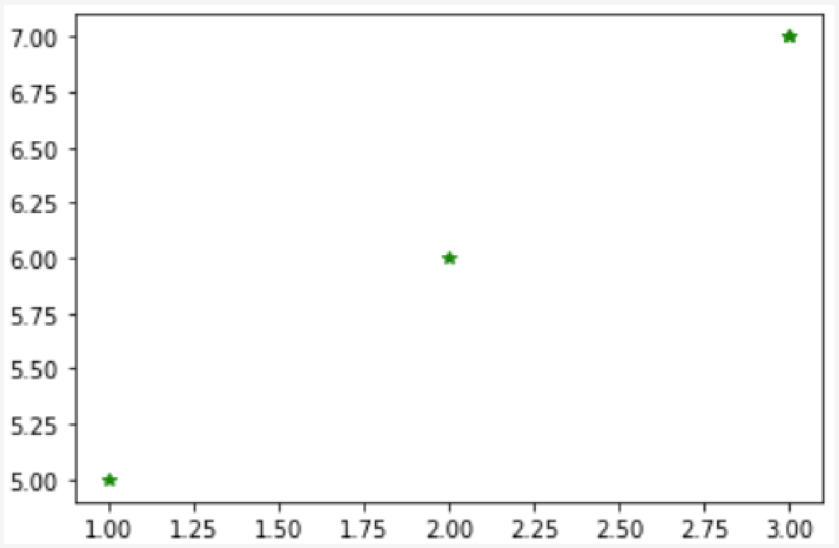
2.5 Next we want to call the plot function to plot these lists:

```
plt.plot(exam_score_maths, exam_score_english)
```

By default this will just return a basic scatter graph, which isn't much use considering we have two lists here, we need to tell the plot function which format we want it to plot:

```
# call the plot function and add some formatting

plt.plot(exam_score_maths, exam_score_english, 'g*')
```

2.6 In the 2.5 example we have specified ‘g*’ as the format, this gives us little green stars.



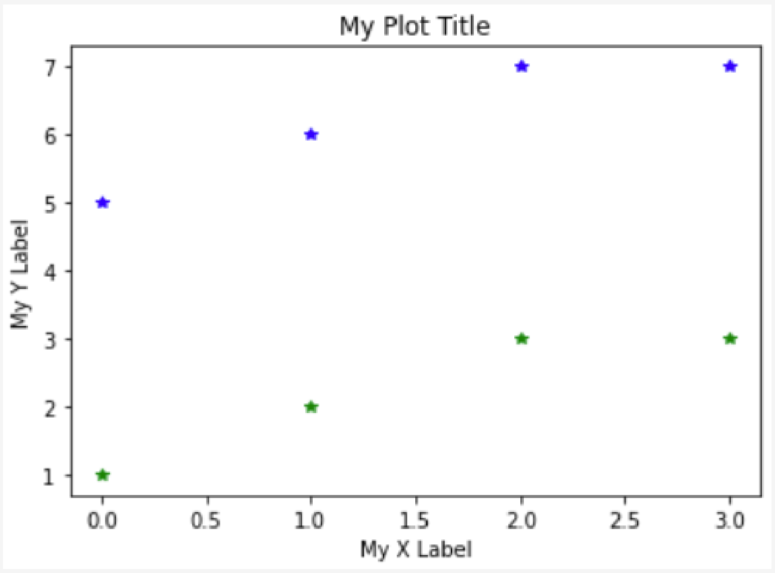
2.7 Try changing the g in format to r,b,y see the colours that come out from the list in 2.3. The * asterisk means small stars, try out some of the other options for plotting markers, there are crosses (‘+’), circles (‘o’). To see full help test you can run:

```
help(plt.plot)
```

2.8 To get the other list plotting alongside our green stars, we need to call the plot function twice then use the show function called `plt.show()`. In this example we will also add an x, y label and a title to the plot:

```
# Draw two sets of points

plt.plot(exam_score_maths, 'g*')

plt.plot(exam_score_english, 'b*')
```

```
# Add figure labels and titles

plt.title('My Plot Title')

plt.xlabel('My X Label')

plt.ylabel('My Y Label')


# Call the show function

plt.show()
```



Part 3: Creating more plots

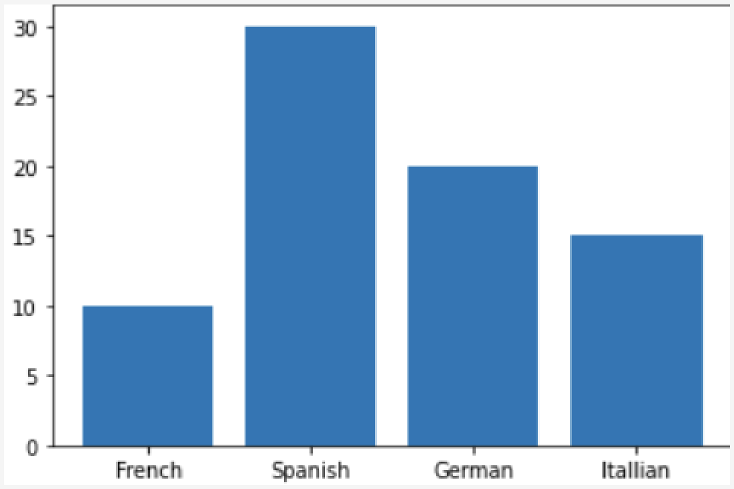
Barplots

3.1 Now that we know how to create different plots and we have seen examples of these running. Time to look at how to create some other visualisations. The first is a bar chart. Bar charts can be achieved with the `plt.bar` function. The function has the following parameters:

```
plt.bar(x, height, width, bottom, align)
```

3.2 Let’s see an example, create two new lists, this time one contains strings and the other integers:

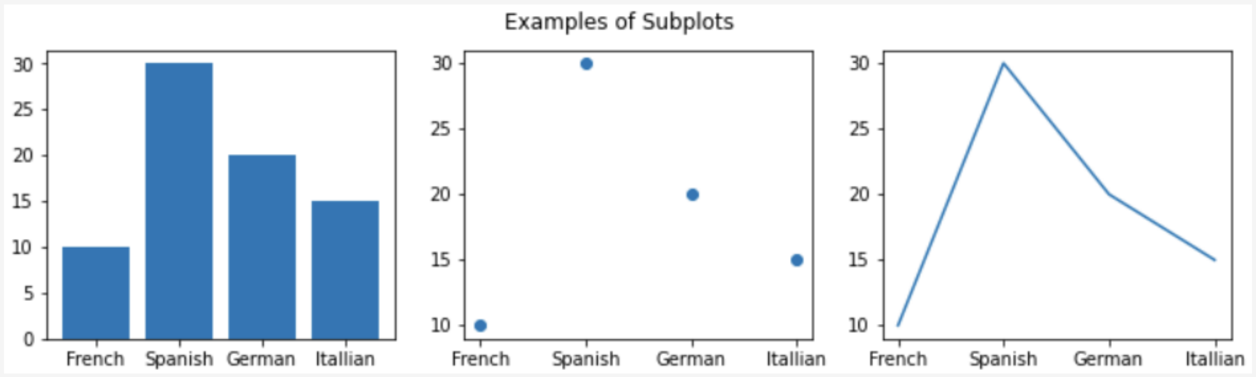
```
# Create two new example lists
languages = ['French','Spanish','German','Italian']
students = [10,30,20,15]


# Specify the bar chart information and then plot
plt.bar(languages,students)
plt.show()
```



Categorical Plotting and Subplots

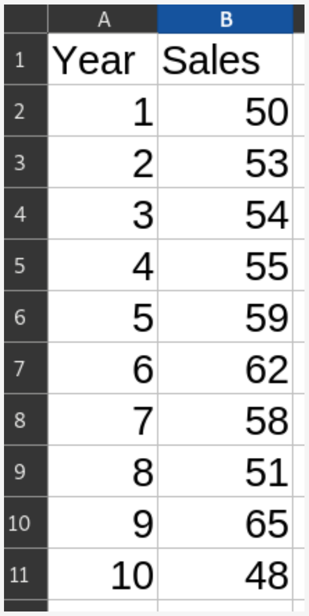
3.3 Let us take our two previous example lists and look at how we can plot those side by side as subplots with three different visualisations, first the plot size is specified. Then 3 subplots are defined, then the plot function is called

```
# Different ways to look at the same lists
plt.figure(figsize=(12, 3))


plt.subplot(131)
plt.bar(languages, students)
plt.subplot(132)

plt.scatter(languages, students)


plt.subplot(133)

plt.plot(languages, students)


plt.suptitle('Examples of Subplots')

plt.show()
```



Part 4: Getting data from csvs

4.1 Download the example CSV file from moodle, this contains 10 sales records. This file is ideal to test code with, because we can see all of the data. If we opened the file using spreadsheet software it would look like this:

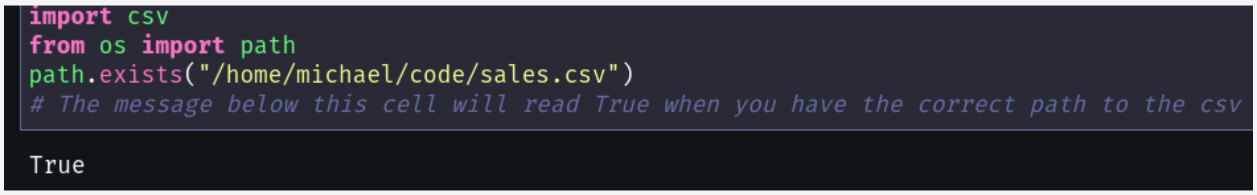


	A	B
1	Year	Sales
2	1	50
3	2	53
4	3	54
5	4	55
6	5	59
7	6	62
8	7	58
9	8	51
10	9	65
11	10	48

4.2 We need to place this in a location that python can read the file from, we also need to tell python the path to this location, the following code will check if the path to the file is valid.

```
import csv

from os import path

path.exists("YOUR FILE PATH HERE/sales.csv")
```

4.3 The path will be different on your machine depending on your folder structure, when you run the code in the notebook the word True or False will appear below. Look at this example, in this case I have the file in my code folder and when the cell is run with CTRL+Enter the value below shows True.



```
import csv
from os import path
path.exists("/home/michael/code/sales.csv")
# The message below this cell will read True when you have the correct path to the csv

True
```

4.4 It is important to get the path to the file right so this True message appears below. Once this step is complete we can move onto extracting the data from the file.

```
# Tell python to open the file

with open(datafile) as sales:

    reader = csv.reader(sales)

# Loop over the file and print out all the rows

    for row in reader:

        print (row)

Expected output:

['Year', 'Sales']

['1', '50']

['2', '53']

...

['9', '65']

['10', '48']
```

Pandas

4.5 This should print out every row in the file, and this is a good test that we can read in a file and print out all of the contents. The basic csv module is what you would expect, a basic CSV module. What we really need to use for data science is a Library called Pandas.

4.6 First pandas needs to be downloaded and installed on your local machine, then import and alis it to the python notebook:

```
pip install pandas

import pandas as pd
```

4.7 Pandas is a fantastic library for working with data. By convention we read the csv into a pandas DataFrame. DataFrame is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects. It is generally the most commonly used pandas object.

```
# import pandas and alias as pd

import pandas as pd

# create the data frame (df) variable and then print

df = pd.read_csv(datafile)

print(df)
```

4.8 Pandas dataframes have a large amount of functions which are documented here: https://pandas.pydata.org/docs/reference/frame.html

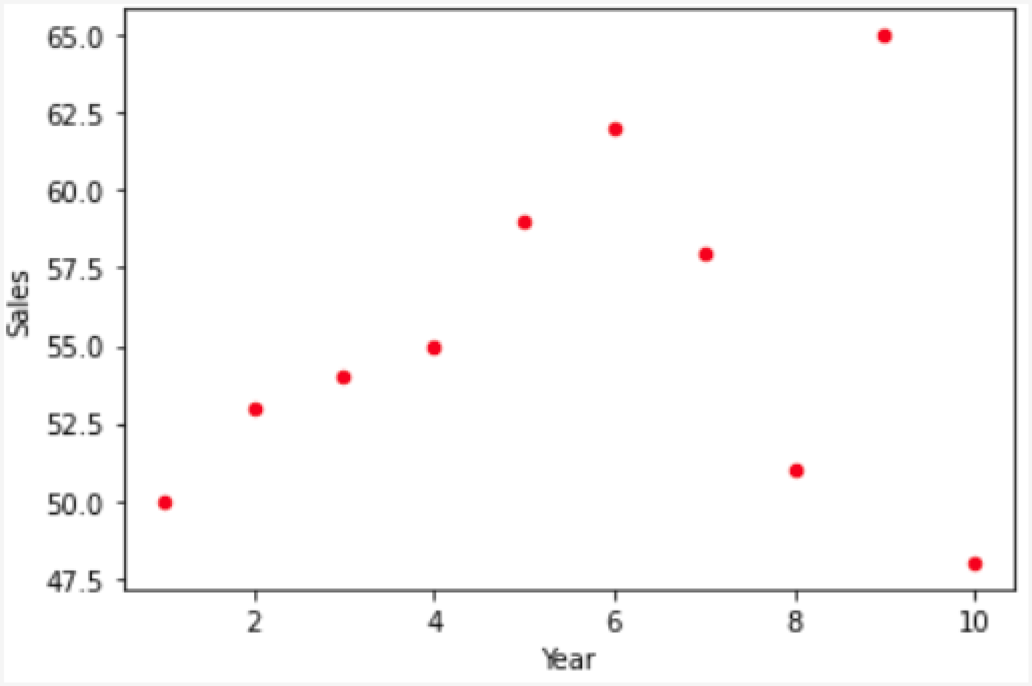
Consider the following examples:

```
# Print specific column

print (df['Sales'])

# Sort values by specified column

print (df.sort_values(by='Sales'))
```

```
# Sum of values in column

print ("The total values in the sales column are: ",
df['Sales'].sum())
```

4.9 Once we can extract data from a file with pandas, we can easily combine this with pyplot to generate visualisations, let's consider the following code. The df variable is our dataframe, plot is our plotting function, then inside the function we can specify what we want. The x values, y values and a color(Americanized spelling). This will generate the plot shown below.

```
df.plot(kind='scatter',x='Year',y='Sales',color='red')

plt.show()
```



Part 5: Conclusion

This week's topics have covered importing libraries that we can use for data science. The two main tasks are getting data into our system, and generating output with that data. Now, please complete the questions in Part 3 of the notebook.