INTRODUCTION TO PYTHON FOR EMATM0044 (ORIGINALLY CREATED FOR EMAT31530)

RYAN MCCONVILLE

1. Introduction

In this first document we will setup a Python environment that is typical for working with data and some other scientific applications. This is by no means any where near a sufficient introduction to Python, but many such introductions do exist[1]. Rather, this is a concise and sufficient introduction to Python for our Introduction to AI course.

2. Environment

2.1. **Option 1.** If you would rather not mess around with setting up an environment on your local machine then you can use Google Colab (`https://colab.research.google.com/`). This is very similar to running Juypter on your local machine, but instead it is running on some Google servers somewhere.

2.2. **Option 2.** We need to install an environment in which Python and the required libraries / packages can run from. Thankfully this has been slowly improving over the years, and as such, we will use a virtual environment[2].

- Download Miniconda for Python 3.7 from `https://docs.conda.io/en/latest/miniconda.html`. For Windows and MacOSX you can follow the install.

  For linux based machines you need to execute the install script downloaded. cd to the directory containing the download.
- From a terminal execute the following and follow the instructions.

```
sh Miniconda3-latest-Linux-x86_64.sh
```

  Then using your favourite terminal emulator

```
conda update conda
conda create -n introtoai python
source activate introtoai
conda install jupyter numpy scikit-learn matplotlib pandas
```

Anytime you open a new terminal and want to do some work for this class (or within the introtoai environment) run

[1] Searching the web will uncover many, if you want some advice let me know.

[2] Consider this a self contained container with isolated packages/libraries.

```
source activate introtoai
```

and all the packages you have installed will be available.

3. Basics

Executing a python script can be done as follows from the terminal

`python FILENAME.py`.

You can also use an interactive shell, ipython, to run scripts, or interactively code from the terminal.

```
ipython
# the next line will fail, unless you happen to have a
# python script called FILENAME.py in the current working directory
run FILENAME
print("Hello, World!")
print(sorted(set(range(2, 1000+1)) - set([p*i for p in range(2, 1000+1)
        for i in range(p, 1000+1)])))
```

You can also use a browser, via jupyter notebook. For the type of coding and analysis we are doing, this is probably the best option (this is more or less what Google Colab is). From your terminal, within the introtoai enviroment, run

```
jupyter notebook
```

and select New → Python 3 in the top right.

4. Python

This course previously used Matlab, but we're going to use Python as it's just better. As well as being a better language, more AI research and applications are done in Python vs. Matlab.

4.1. **NumPy.**

NumPy is an extremely popular library used when working with numbers (hence the name). Working within the AI domain, it is typical to make use of NumPy because it is fast and highly optimised.. Note that it is a convention (because people are lazy) to import numpy as np. Try running the code below to get an idea about what is happening.

```
import numpy as np

x = np.array([1,2,3])
print(x.ndim)
print(x.shape)
y = np.array([[1], [2], [3]])
print(y.ndim)
print(y.shape)
A = np.array([[1,1,1] ,[2,2,2], [3,3,3]])
```

```
print(A.ndim)
print(A.shape)
```

However, rarely would we create arrays like this.. NumPy has some nice functions for creating arrays.

```
a = np.arange(100) # 0 to n-1
print(a)
b = np.linspace(0, 1, 10)   # start, end, num-points
print(b)
c = np.ones((4, 4))  #(4, 4) is a tuple
print(c)
d = np.eye(4)
print(d)
e = np.diag(np.array([1, 2, 3, 4]))
print(e)
d = np.random.rand(4) # uniform in [0, 1]
print(d)
e = np.random.randn(4) # draw from Gaussian
```

4.1.1. *Arithmetic.* We can do typical operations (**element wise**) using our NumPy operations.

```
x = np.arange(3).reshape(3,1)
print(x)
xp1 = x + 1
print(xp1)
print(xp1 - x)
y = np.arange(3).reshape(1,3)
print(y)
print(2**y)
A = np.arange(9).reshape(3,3)
print(A)
```

Matrices have their own functions.

```
print(A * A) # compare with the next..
print(A.dot(A))
print(A.dot(x)) # scalar product
print(A.dot(y.T)) # scalar product with transpose of y
```

There are also a bunch of other nice 'reductions' in NumPy.

```
a = np.arange(100) # 0 to n-1
print(np.sum(a))
print(np.min(a))
print(np.max(a))
```

```
print(np.mean(a))

# axis lets you control columns vs rows
a = np.array([[1,1], [2,2,], [3,3]])
print(a)
print(a.sum(axis=0))
print(a.sum(axis=1))
```

4.1.2. *Indexing.* Matrices are indexed by [ROW, COL]. You can access all elements using [:]. Using the index -1 is cool as it lets you access the last element, -2 the second last, and so on.

```
A = np.arange(9).reshape(3,3)
print(A[0])
print(A[0][0])
print(A[1])
print(A[1][2])
print(A[2])
print(A[-1])
print(A[-2])
print(A[2, 0]) # third row, first column
```

4.2. **Pandas.** Pandas is a popular library for data analysis. In this subsection I'll try and give you a basic and concise overview of some typical Pandas functionality. There are two primary data structures in pandas, the **DataFrame** which you can imagine as a table with rows and named columns, and the **Series**, which is a single column. Thus, as you imagine, a DataFrame is made up of many (at least one) Series.

4.2.1. *Loading.* Let's create our own sample data..

```
import pandas as pd

books = pd.Series(['Harry Potter and The Philosophers Stone',
 'Artificial Intelligence: A Modern Approach',])
book_class = pd.Series(['Fiction', 'Non-Fiction'])
authors = pd.Series(['JK Rowling', 'Russel and Norvig'])
# note these numbers may have been made up
sold = pd.Series([120000000, 10000])

book_frame = pd.DataFrame({'Book Title': books, 'Book Classification': book_class,
'Author Name': authors, 'Sales': sold})
book_frame # the output of the last expression in a cell is displayed
```

However, typically, you will be reading the data in from a file using the function **read_csv**. This can either be a local file or a remote file over HTTP.

```
# don't worry about the .data extension, it's really just a CSV file.

# this won't work, unless we have the crx.data file in this directory.
df = pd.read_csv('crx.data', sep=',', header=None)

# the following does the same as above, but instead loading it over the Internet
df = pd.read_csv(
'http://archive.ics.uci.edu/ml/machine-learning-databases/credit-screening/crx.data',
 sep=',', header=None)


# print a summary of the dataframe we loaded.
print(df)

# print the first few rows of the dataframe
print(df.head())

# print the last few rows of the dataframe
print(df.tail())
```

4.2.2. *Accessing.* You can access the DataFrame much the same as normal Python data structures.

```
print(book_frame['Book Title'])
print(book_frame['Book Title'][1])
print(book_frame['Book Title'][0:2])
```

You can access slices of rows as follows

```
print(book_frame.iloc[0])
print(book_frame.iloc[1]['Book Title'])
print(book_frame.iloc[0:1])
```

4.2.3. *Manipulation.* It's also pretty straightforward to manipulate the data within your frames.

```
# Oops, maybe we underestimated the sales of books by a factor of 10

book_frame['Sales'] = book_frame['Sales'] * 10
print(book_frame)

import numpy as np
log_sales = np.log(book_frame['Sales'])
print(log_sales)
book_frame['Log_Sales'] = log_sales
print(book_frame)
```

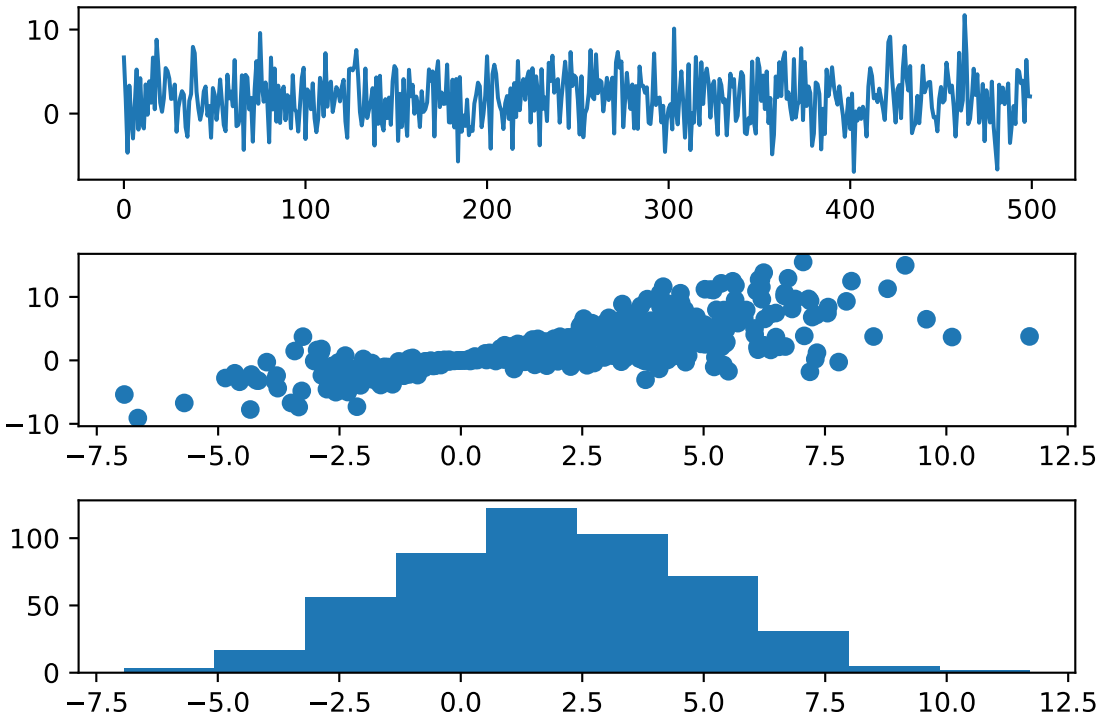
Try adding a column to the book_frame with your own rating of the books out of 10.

4.3. **Matplotlib.** Matplotlib is the most popular library for visualisation with Python. As with NumPy it is generally imported in a shorthand way (matplotlib.pyplot as plt). Two important aspects of matplotlib are figures and axes. Figures correspond to a physical figure, while an axes corresponds to regions or sections of the figure. Therefore, naturally, you first create a figure and add axes too it.

```
%matplotlib inline # `magic' command to get images to display inline
import matplotlib.pyplot as plt
fig, (ax1, ax2) = plt.subplots(2) # two plots..
fig, (ax1, ax2) = plt.subplots(2, sharex=True, sharey=True) # same as above,
# but with a shared x and y axis between the two plots
```

Now lets generate some data and plot it on a few axes.

```
fig, (ax1, ax2, ax3) = plt.subplots(3)
sigma=3
mu = 2
n = sigma * np.random.randn(500,1) + mu
ax1.plot(n)
# add some noise to n for visualisation purposes
mu_noise, sigma_noise = 1, .7
noise = sigma_noise * np.random.randn(500,1) + mu_noise
y = n * noise
ax2.scatter(n, y)
ax3.hist(n)
# annoyingly we sometimes need this to display plots with better spacing
plt.tight_layout()
```



5. Random Variables

We can generate random variables using NumPy quite easily. Line 1 will generate 100 numbers from a gaussian pdf and store them in the variable n100. Line 2 plots the distribution. Line 3 plots a histogram with 10 intervals from n100.

```
1 n100 = np.random.normal(1,1,100)
2 plt.hist(n100, bins=10)
```

Repeat these steps with a sample size of 5000 (call the variable n5000), with 10 and 100 bins.

5.1. **Transformations of 1-d random variables.** As you have seen before, NumPy makes it very easy to transform random variables.

Apply the following transformations to n5000:

```
1         p3 = n5000 + 3
2         p4 = n5000 * 0.5 + 3
```

Compute their normalised histograms, plotting them in different figures. Have a look at the NumPy reductions from above to calculate the same mean, standard deviation and variance of the variables.

6. Real data

And finally, let's work with some real data. Using pandas read in the file

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

df = pd.read_csv('https://seis.bris.ac.uk/~rm17770/data1414.txt', sep='\s', header=None)
labels = df.iloc[:,-1] #select all rows (`:`) against the last column (`-1`)
data = df.iloc[:,:-1] # select all rows, but this time note the extra :,
# this says select all columns from the start up to the last
plt.imshow(np.array(data.iloc[10]).reshape(14,14), cmap='gray')
```

6.1. **Data analysis.** Using pandas it is easy to do some exploratory data analysis.

```
# how many images do we have
print(len(data))
# what is the average value for each pixel
print(data.mean())
# what is the max / min value for each pixel
print(data.max())
print(data.min())
```

One nice property of images is that we can visualise them ;), so let's visualize the average image of a few different classes (digits).

```
 # let's visualise the average of *all* classes.
 plt.imshow(np.array(np.mean(data, axis=0)).reshape(14,14), cmap='gray')
 # ok that's not very useful..

# select all rows of the dataframe where the class is digit N
zero_idx = df.index[df[df.columns[-1]] == 0]
#loc is similiar to iloc, which we saw earlier. loc access rows via the index label,
#rather than the position of the index.
zero = data.loc[zero_idx]
plt.imshow(np.array(np.mean(zero, axis=0)).reshape(14,14), cmap='gray')
```

```
ones_idx = df.index[df[df.columns[-1]] == 1]
ones = data.loc[ones_idx]
plt.imshow(np.array(np.mean(ones, axis=0)).reshape(14,14), cmap='gray')
```

6.2. **Simple handwritten digits recognition.** A first problem posed by the dataset is digit recognition. One intuitive approach to this task is based on identifying which pixels are key to distinguish two digits. Each pixel in X can be considered as a random variable with values in [0,255]. Imagine we had a **very very** simple AI that can only

look at one pixel. Let us analyze the random variables corresponding to four pixels to determine which of them would be more useful to distinguish 0 and 1. The pixels to analyse are those in positions 49, 50, 60, 102.

6.3. **Question.** Try the following code and determine which pixel is the most relevant for this task.

```
pixels=[49,50,60,102]


for pixel in pixels:
    fig, (ax1, ax2) = plt.subplots(2, sharex=True, sharey=True)
    ax1.set_title('Zero Pixel '+str(pixel))
    ax1.hist(zero.iloc[:,pixel], bins=20)
    ax2.set_title('One Pixel '+str(pixel))
    ax2.hist(ones.iloc[:,pixel], bins=20)
    plt.show()
```

7. Conclusion

Hopefully this intro has covered lots of interesting things which you will find useful in this course (and maybe beyond). Soon we will look at machine learning, and how we can build models that can learn from data.