Week 2 Tutorials

Introduction to Python for Data Science

This is an introductory activity for FIT1043 Introduction to Data Science. Python is currently the most popular programming language, ahead of Java and JavaScript. It is especially popular for Data Analytics.

This, and the following activities, will quickly introduce the basic syntax of Python and will look at some of the functionality and libraries/tools that are useful for Data Science.

1. Loading Libraries

In programming, a library is a collection of functions or methods that perform different computational actions. In short, commonly used functions or methods are provided for within the library and hence saving you from needing to write your own code.

To load a library in Python we use the keywords: "`from` ... `import` ... `as` ...". The "`as`" is optional and is used to give the imported library a shorter or meaningful name, for example:

```
from matplotlib import pyplot as plt
```

Here, `pyplot` is the plotting framework of `matplotlib` library. This will load the `pyplot` library from `matplotlib`. You can also use a shorter "." notation rather than "`from`" to load the library, for example:

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

In either case, we have given the name "`plt`" to the imported library and will use it whenever we want to call a function provided by the library, for example "`plt.boxplot(…)`"

2. DataFrames in Python

In Data Science applications we need to manage and manipulate large data tables. For this purpose the "`pandas`" library provides a data structure called a DataFrame. To use the data structure we need to first import the pandas library (type the following into your Jupyter notebook):

```
import pandas as pd
```

Now we'll use the `DataFrame()` method to create a new table structure from a set of lists:

```
df = pd.DataFrame({
'StudentID' : [264422,264423,264444,264445,264446],
'FirstName' : ['Steven','Alex','Bill','Mark','Bob'],
'EnrolYear' : [2010,2010,2011,2011,2013],
'Math' : [100,90,90,40,60],
'English' : [60,70,80,80,60]
})
```

“`df`” is just a variable name for the pandas DataFrame. You can print out the new DataFrame to see what it looks like:

```
df
```

Practise 1: Create another data table called "`df2`" that contains the height of some of the students. It should contain the following data:

	Height	*Student*
0	*160*	*264422*
1	*155*	*264423*
2	*175*	*264444*
3	*175*	*264445*

2.1 Column and row selection

We can select a column using its name and the bracket syntax:

```
df['FirstName']
```

Or using the somewhat simpler dot notation:

```
df.FirstName
```

We can also select only elements that fit certain conditions. What does the following command produce?

```
df[df['FirstName'] == 'Alex']
```

Why does "`df`" appear twice in the command? If you are unsure, try to take the output from “`df.FirstName`“ and substitute it into “`df['FirstName']`”

Practise 2: Show the details of the students who enrolled in the year 2011?

We can select rows with ever more complicated conditions, for example:

```
# select rows where first-name isn't 'Bob' (!= is not equal)
# and student ID is larger than 2644423
filt = (df.FirstName != 'Bob') & (df.StudentID > 264423)
df[filt]
```

Before you continue to the next section, explore how you can select different columns and rows of the DataFarme that you have created above.

2.2 Modifying the structure of a table

It is possible to modify the structure of DataFrames in Python. For example, we might want to add a new column containing the total of the "Math" and "English" marks for the students. To do this we can simply write:

```
df['Total'] = df['Math'] + df['English']
```

The code above will create a new column call “Total” and populate it with the sum from the two columns “Math” and “English”. Print out the table to see the result.

```
df
```

Practice 3: Add a new column showing the average mark of "Math" and "English" for each student. (Average = Total/number of subjects)

Practice 4: Write a filter to select students who have scored more than 150 in total and who have achieved a subject score (in Maths or English) of 90 or more. (Hint: careful with the parenthesis)

Oftentimes, we need to modify the layout of a table so that the data is in the right format for further processing or visualisation. In our current table we have two columns "Math" and "English", both of which contain student marks. We can split the information for each student into different rows using the "`melt`" command, as follows:

```
# turn column names 'Math' and 'English' into values
# for a new column 'Subject'
df = pd.melt(
    df,
    id_vars=['EnrolYear','FirstName','StudentID'],
    value_vars=['Math','English'],
    var_name='Subject')
```

Print out the new table. What have we done here?

We have taken each record (row) from the original table, and turned it into two rows. What does the new column "`Subject`" contain?

Practice 5: Based on the output table, explain what the arguments "`id_vars`", "`value_vars`", and "`var_name`" to the `melt()` command are doing.

The values that were in the columns Math and English now appear in a different column "`value`". We can rename this new column to make the new table more understandable:

```
df.rename(columns = {'value':'Score'}, inplace = True)
```

2.3 Merging DataFrames

Suppose now that for some reason you would like to merge the two data sets, `df` and `df2`. We can do that simply by calling the "`merge`" command:

```
df3 = pd.merge(df, df2, on=['StudentID'])
```

Practice 6: What is the difference that can you see after merging to datasets?

Practice 7: Print out the merged table. Based on the output, explain what you understand about the parameter `on=['StudentID']`.

Practice 8: Looking closely at the data in the resulting DataFrame, is there a student missing after the merge? How can you display all students?

3 Reading CSV and Excel files into DataFrames

We can read DataFrames directly from a CSV file using the "`read_csv`" command. (CSV stands for Comma Separated Value, meaning that the fields stored in the text file are separated using a ‘,’, you can open the file in a text editor and have a look). Let's read in the file `uforeports.csv` that's available on Moodle. In order for the Jupyter Notebook to have access to the file, you'll need to first download it from Moodle and copy it to the Jupyter Notebook folder (for easier access). Once you've done that, you can simply type the following:

```
ufo_reports = pd.read_csv('uforeports.csv')
```

If you copied it to another folder, then you will need to provide the path name to the file. Print out the first five records from the file using the "head" command:

```
ufo_reports.head() # print the first 5 records
```

Practice 9: How can we display the last 5 records? How can we display the last record only?

To read in an Excel .XLS file, we need to specify not only the filename but also the name of the spreadsheet within the workbook. Download the file `uforeports_excel.xls` and again copy it to you Jupyter Notebook folder and type:

```
ufo_reports_xls = pd.read_excel(
                     'Uforeports_excel.xls',
                     sheet_name='uforeports')
```

Use the "`head`" command to print out the first 5 records of the new table. Notice that the CSV file and the XLS one contained the same data, but in different formats.

It looks like Python has recognised the Time column differently for the two datasets. Let's print out the datatype for the Time column to see what's going on:

```
print(ufo_reports.Time.dtypes)
```

and do the same for the XLS version::

```
print(ufo_reports_xls.Time.dtypes)
```

Notice that Time columns in two DataFrames are indeed different. (Note: If they both return as objects, it is likely because your date time setting for your computer is set to US style of Month-Day-Year. You can use the “`uforeports_excel_us.xls`”) In the table read in from the CSV file, the Time column is just an object, while for the XLS file it was given a datetime format. We can change the format so they are consistent:

```
# change to datetime format
ufo_reports.Time = pd.to_datetime(ufo_reports.Time)
print (ufo_reports.Time.dtypes) # print its datatype now
ufo_reports.head() #
```

Check whether it looks the same as the other dataframe Has it fixed the problem?

4 Basic data auditing

We could easily do some auditing or exploration of the data using Python. We'll leave graphical data exploration and outlier identification to next week. Instead, we'll use the titanic data set (a list of passengers on the Titanic) to investigate some basic auditing functions.

Practice 10: Write a statement to read in titanic data (`titanic.csv`) into a pandas DataFrame called '`titanic`'. Then print out the first few rows.

We'll now check various characteristics of the data. First let's have a look at the dimensions of the table:

```
print(titanic.shape)
```

How many rows and columns are there?

We can investigate the data types of each column, by calling "`dtypes`" on the entire DataFrame:

```
titanic.dtypes
```

We can do summary statistics to see the count, number of unique values, as well as the value range of numeric fields.

```
titanic.describe()
```

What does the describe method print out?

We can have a look at the unique values for each column using the "`value_counts()`" method:

```
titanic.age.value_counts()
```

Practice 11: Share your findings of the previous two statements with other students, e.g. what is the average age of the Titanic's passengers, and the mode?

5 Aggregation and group-by

Continuing with the titanic data set, let's now focus on the age of passengers. We've already seen that we can calculate the average value of a column using the `describe()` function. Alternatively, we could make use of the "`mean()`" function directly, by typing:

```
titanic['age'].mean()
```

There are many other aggregation functions we might wish to use to investigate the age of the passengers. For example:

```
titanic['age'].max() # oldest
titanic['age'].min() # youngest
titanic['age'].sum() # total years
titanic['age'].std() # standard deviation
titanic['age'].median() # half of the people were older (/younger)
titanic['age'].mode() # most common age
```

Knowing the average age over all passengers is interesting, but knowing the average age for different groups of passengers may be more useful. We could then answer questions like: How old were the first-class passengers on average? Was the average age the same for men and women? And so on ...

Let's now have a more detailed look at the age of different groups. We can do that simply by making use of the `groupby()` command. We'll use the command to create a (hierarchical) index on "`sex`" and "`class`" values, so that we can then compute aggregate values (the mean) over each of the groups:

```
sex_class = titanic.groupby(['sex','class'])['age']
sex_class.mean()
```

Practice 12: Interpret the output, e.g. what did you notice about the average age of the Titanic's passengers in regards to their classes? How about the relationship between age and gender?

Practice 13: Use other aggregation functions and groupings to determine which class had the oldest and youngest passengers and which gender had the largest amount of variation in age (standard deviation).

Next week we'll investigate some more sophisticated `pandas` commands as well as look at graphing data using Python.

End of Tutorial 2