# BUSA3020-Tutorial-Week1

February 15, 2022

## 1 Week 1 Tutorial - Introduction

### 1.0.1 Unit Convenor & Lecturer

George Milunovich
george.milunovich@mq.edu.au

In this tutorial we will learn how to work with CSV/Excel files in Python. Our dataset contains credit card default data available from https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients. Within the dataset each row represents a different client and each column stores client attributes (e.g. age, merital status, etc) with the last column being the target variable - whether the client has defaulted on the payment or not.

---

**Objectives**

- Assign values to variables
- Explain what a library is and what libraries are used for
- Import a Python library and use the functions it contains
- Read tabular data (txt/csv/excel/etc) from a file into a program
  - pandas Python libarary
  - Default of Credit Card Clients Dataset
  - https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients
- Select individual values and subsections from data
- Perform operations on arrays of data
- Plot simple graphs from data
- Writing Functions in Python
- Writing Classes in Python

---

### 1.1 Variables

Any Python interpreter can be used as a calculator:

Type the following into the cell below and hit Shift+Enter at the same time. It will execute

```
2 * 3 + 5 - 1
```

[ ]:

In Python, we can also assign a value to a variable, using the equals sign =. For example, to assign value 60 to a variable weight_kg, we would execute:

```
weight_kg = 60
print(weight_kg)
```

[ ]:

From now on, whenever we use weight_kg, Python will substitute the value we assigned to it. In essence, a variable is just a name for a value.

```
print(weight_kg + 5)
print(weight_kg)
```

[ ]:

In Python, variable names:

- can include letters, digits, and underscores - `A-z, a-z, _`
- cannot start with a digit
- are case sensitive.

This means that, for example:

`weight0` is a valid variable name, whereas `0weight` is not `weight` and `Weight` are different variables

```
Weight_kg = 1
print(weight_kg, Weight_kg)

a = 2
A = 3

print(a)
print(A)
```

[ ]:

---

## 1.2 Types of data

Python has a number of different types of data, however we do not need to explicitely declare the types of data. Three common data types are:

- integer numbers (whole numbers)
- floating point numbers (numbers with a decimal point)
- strings (of characters, i.e. text)

In the example above, variable `weight_kg` has an integer value of `60`. To create a variable with a floating point value, we can execute:

```
print(f'This variable is an integer: {weight_kg}')
weight_kg = 60.0
print(f'This variable now is a floating point number (decimal number): {weight_kg}')
```

[ ]:

And to create a string we simply have to add single or double quotes around some text, for example:

```
weight_kg_text = 'weight in kilograms:'
print(weight_kg_text)
```

[ ]:

Moreover, we can do arithmetic with variables right inside the print function:

```
print('weight in pounds:', 2.2 * weight_kg)
```

[ ]:

The above command, however, did not change the value of weight_kg:

```
print(weight_kg)
```

[ ]:

To change the value of the ‘`weight_kg variable, we have to assign`weight_kg`a new value using the equals`=‘ sign:

```
weight_kg = 65.1
print('weight in kilograms is now:' , weight_kg)
print(f'weight in kilograms is now: {weight_kg}')
```

[ ]:

Lets check what kind of variables we’ve crated using the `type` command. Try:

```
print(weight_kg, type(weight_kg))
print(weight_kg_text, type(weight_kg_text))
```

[ ]:

---

### 1.3 Changing Variable Values

As the name implies, a variable is something which can change. Once a variable is assigned a value we can easily overwrite it by giving it another value. Try the following:

```
x1 = 10
print(x1)
```

```
x1 = 12.1
print(x1)
```

[ ]:

**Updating a Variable** Variables calculated from other variables do not change value just because the orignal variable change value (unlike cells in Excel):

```
x1 = 10
print(f'x1 is {x1}')

x2 = x1   # create variable x2 and assign it the value of x1
print(f'x2 is {x2}')

x1 = 5
print(f'x1 is now {x1}')
print(f'x2 is still {x2}')
```

[ ]:

Since `x2` doesn't *remember* where its value comes from, it is not updated when we change `x1`. Such variables are called **immutable**, i.e. if this variable is created by assigning to it a value from another variable, then changing the value of the original variables will not change the value of the new variable (as above). There are some variables for which this is not true, e.g. lists, which are said to be **mutable**. More on this later.

---

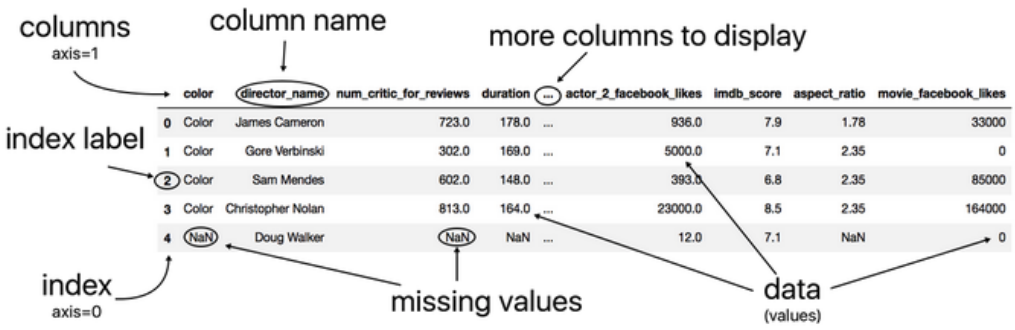
## 1.4 Libraries

Python libraries are collections of functions and methods that allow us to perform many actions without writing your code.

- While a lot of powerful, general tools are built into Python, specialized tools built up from these basic units live in *libraries* that can be called upon when needed.

## 1.5 pandas

pandas is a Python library written for data manipulation and analysis. It offers data structures and operations for manipulating numerical tables and time series. The name is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals.




Anatomy of a DataFrame

In order to load our credit card dataset into Python, we need to access (import in Python terminology) a library data allows us to read csv or excel files.

In general you should use this library if you want to do fancy things with numbers, especially if you have matrices or arrays. We can import `pandas` and call it by its abbreviation `pd` using:

```
import pandas as pd
```

[ ]:

Lets first download the data into the data directory within the current directory, by clicking https://archive.ics.uci.edu/ml/machine-learning-databases/00350/default%20of%20credit%20card%20clients.xls and copying the file into the data directory

Now we will import this saved excel file using the following command and assign it to the df (dataframe) variable

```
df = pd.read_excel('data/default of credit card clients.xls')
```

```
df
```

[ ]:

The expression `pd.read_excel(...)` is a function call that asks Python to run the function `read_excel` which belongs to the `pandas` library that we named `pd` here.

This dot `.` notation is used everywhere in Python: the thing that appears before the dot contains the thing that appears after.

See documentation for read_excel method

In `pd.read_excel(...)` we may three parameters: - the name of the file we want to read, which needs to be character strings (or strings for short), so we put it in quotes - which row of excel table to skip importing, here we skip row 0 (see original excel file for why we do this) - which varible will be used for index column

Note that instead of first saving the file to our hard drive we could have directly read it into Python as follows

```
df2 = pd.read_excel('https://archive.ics.uci.edu/ml/machine-learning-databases/00350/default%20
```

```
print(df2)
```

[ ]:

Now that the data is in memory, we can manipulate it. First, let's ask Python what type of thing `data` refers to:

```
print(type(df))

print(type(df2))
```

[ ]:

The output tells us that data currently refers to a pandas DataFrame object, the functionality for which is provided by the `pandas` library.

---

### 1.6 Working with pandas DataFrame

A few commands come in useful when working with pandas DataFrame objects. Try the following:

```
print(df2.info())    # DataFrame information

print(df2.columns)   # list columns

print(df2.shape)     # show dimensions of the dataframe, i.e. number of columns followed by the

print(df2.head())    # print the first 5 examples (observations)

print(df2.tail())    # print the last 5 examples (observations)
```

Also as you can see below the hash symbol `#` indicates a comment, which is a way of keeping notes in Python without having the text following it being executed.

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

If we wish to get a single column (last columns here) of the dataframe and lets say assign it to the variable `y`, while assigning the remaining variables a new dataframe `X` we could do the following:

```
y = df2['default payment next month']
print(y)
```

```
X = df2.copy()
print(X)
```

```
del X['default payment next month']  # to remove y from X
print(X)
```

Note: I used copy() here rather than a simple = assignment in the second line.
This is because pandas are **mutable** - if I delete a 'default payment next month' from X it would get also deleted from df. Using copy() avoids this..

[ ]:

[ ]:

[ ]:

### 1.6.1 Accessing Elements

- Selecting multiple rows, note: does not include 3, e.g. `df2[0:3]`
- Selecting multiple rows - method 2, note: includes 3, e.g. `df2.loc[0:3]`
- Selecting multiple columns, e.g.: `df2[['LIMIT_BAL', 'SEX', 'EDUCATION']]`
- Selecting multple rows and columns using .loc[], e.g.: `df2.loc[0:3, ['LIMIT_BAL', 'SEX', 'EDUCATION']]`

[ ]:

[ ]:

[ ]:

[ ]:

## 1.7 Descriptive Statistics

```
df2.describe() # print all descriptive statistics
```

```
df2.mean() # print mean
```

- etc.

[ ]:

[ ]:

---

## 1.8 Visualising Data - Basic Plots in Python

We start with a few features of Python's `matplotlib` library here.

- While there is no official plotting library, `matplotlib` is the de facto standard.
- First, we will import the `pyplot` module from `matplotlib` and use it to plot the bar chart of the means of all the columns in df DataFrame

```
import matplotlib.pyplot as plt

X.mean().plot(kind='bar')

plt.show()
```

[ ]:

There are many different types of plots we can do with `pandas` and `matplotlib`.

For instance we could do a scatter plot between BILL_AMT4 and BILL_AMT5 using the following command

```
df2.plot(kind='scatter',x='BILL_AMT4',y='BILL_AMT5',color='red')

plt.show()
```

[ ]:

Instead of using `matplotlib` we could also use `seaborn` another plotting library, which can also plot the fitted regression line

```
import seaborn as sns

sns.lmplot(x='BILL_AMT4', y='BILL_AMT5', data=df2)
```

[ ]:

---

## 1.9 Mystery Functions in IPython

How did we know what functions pandas has and how to use them?

If you are working in a Jupyter Notebook/JupyterLab (which we are), there is an easy way to find out.

- If you type the name of something followed by a dot `.`, then you can use `Tab` completion (e.g. type `pd.` and then press `tab`) to see a list of all functions and attributes that you can use.

```
pd.
```

[ ]:

After selecting one, you can also add a question mark `?` (e.g. `pd.read_excel?`), and IPython will return an explanation of the method!

This is similar to running `help(pd.read_excel)`.

[ ]:

[ ]:

---

### 1.10 Functions in Python

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

In Python a function is defined using the `def` keyword. Try the following

```
def my_function():
  print("Hello from a function")
```

[ ]:

As you can see nothing has happened. However, Python has created a function called `my_function`, which is now available for us to call by typing its name.

```
my_function()
```

[ ]:

We can pass parameters to the function inside the parenthasis. Lets make a new function to add two numbers and print out the answer as follows:

```
def add_two_numbers(number_one, number_two):
    sum = number_one + number_two
    return sum
```

[ ]:

We can now call the function `add_two_numbers` and give it which two numbers to add, e.g. 3 and 6

```
result = add_two_numbers(3, 6)
print(result)
```

[ ]:

[ ]:

---

## 1.11 Classes in Python

Classes provide a means of bundling data and functionality together.

- Creating a new class creates a new type of object, allowing new instances of that type to be made.
- Each class instance can have attributes attached to it for maintaining its state.
- Class instances can also have methods (defined by its class) for modifying its state.

Lets create a class which does algebra with two numbers, which will contains a function for adding two numbers and multiplying two numbers

```
class algebra_with_two_numbers:
    def __init__(self, n_one = 3, n_two = 6):
        self.number_one = n_one
        self.number_two = n_two

    def print_function_attributes(self):
        print(self.number_one, self.number_two)

    def add_two_numbers(self):
        self.sum = self.number_one + self.number_two
        return self.sum

    def multipy_two_numbers(self):
        self.product = self.number_one * self.number_two
        return self.product
```

[ ]:

Lets create two instances of `algebra_with_two_numbers` class and print their attributes

```
algebra_1 = algebra_with_two_numbers()
algebra_1.print_function_attributes()


algebra_2 = algebra_with_two_numbers(n_one = 1, n_two = 2)
algebra_2.print_function_attributes()
```

[ ]:

[ ]:

Lets create a new instance of the `algebra_with_two_numbers` class and use its two remaining methods:

```
one = 1
two = 15
algebra = algebra_with_two_numbers(n_one=one, n_two=two)

print(algebra.add_two_numbers())
```

```
four = algebra.multipy_two_numbers()
print(four)

algebra.add_two_numbers() + algebra.multipy_two_numbers()
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: