# Week 4 Laboratory Activity

## Data Wrangling in Python

Upload the file `Patients.txt` (from Alexandria) to the same folder location as your Jupyter Notebook and then start a new Python 3 Notebook. The first 4 lines of the file `Patients.txt` content looks like

```
001M11/11/1998 88140 80  10
016F11/13/1998 84120 78  X0
033X10/21/1998 68190100  31
004F01/01/1999101200120  5A
```

The data in `Patients.txt` is not formatted as a CSV file, but rather in a fixed-width format, so we'll try to load the data into a DataFrame by calling the `read_fwf()` function (`fwf` stands for **f**ixed **w**idth **f**ormat) rather than the `read_csv()` function.

```
import pandas as pd
df = pd.read_fwf('Patients.txt', colspecs='infer', header=None)
```

Note that we have asked the Pandas library to try to work out what columns are in the data using the parameter `colspecs="infer"`. We have also told it that there are no column headers on the first line of the file.

Did it work? Print the DataFrame to see:

```
print(df)
```

How many columns did Pandas find? To see the dimension of the table found you can also type:

```
df.shape
```

You should get (30, 2) which means that it has 30 rows and 2 columns. From our manual inspection, we probably can tell that this is not what we want. Looking at a line of the raw content, it seems that the first 3 digits means something, followed by an indication that the person is either a Male (M) or Female (F), then followed by a date and so on.

Let's assume that we are aware that the data should contain the following information.

- A 3 digit identity
- Gender
- Date of visit
- 3 digit heart rate reading
- 3 digit Systolic blood pressure
- 3 digit Diastolic blood pressure

- 3 digit Diagnostic code
- 1 digit to indicate whether it is an Adverse Event

# 1. Reading the File

Since Pandas can't do it automatically (the `colspecs="infer"`), we will help Pandas to load the data by specifying the width of each column in the original data:

```
df = pd.read_fwf('Patients.txt', widths=[3,1,10,3,3,3,3,1],
header=None)
```

Did it work? Print out the data.

```
df.head()
```

*Practise 1: What does `df.shape` output?*

*Practise 2: Find out how to rename the DataFrame columns (which is labelled as 0, 1, 2, 3, 4, 5, 6, 7) at the moment. Rename them to 'ID', 'Gender', 'Visit', 'HR', 'SBP', 'DBP', 'DX', 'AE'.*

If you inspect the whole DataFrame, there will be `NaN` in some of the fields. `NaN` stands for "Not A Number".

## 1.1 Visualizing the Data

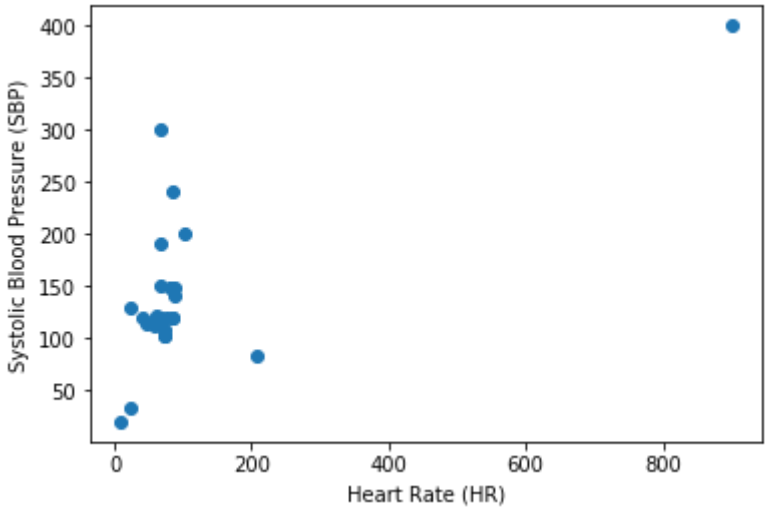
Having named the columns we can now refer to them by name. For example, we can print the Systolic Blood Pressure (SBP) values by typing:

```
df['SBP']
```

*Practise 3: Plot a scatter plot of Heart Rate (HR) against Systolic Blood Pressure (SBP) using matplotlib:*

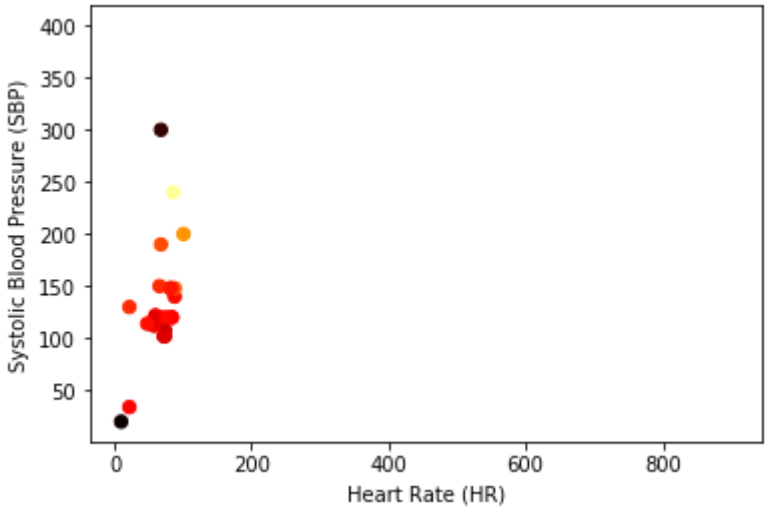
When you look at the plot, you should see a point on the top right hand side of the graph that looks suspicious. We need to investigate this further and check on the values. To be clear, we should label our axes.

*Practise 4: Label the axes for HR and SBP as "Heart Rate (HR)" and "Systolic Blood Pressure (SBP)". You should get a graph that looks like:*




A 2 dimensional plot would normally mean a comparison between 2 variables. However, we can use other means, such as colour to help us visualize more. If we wanted to see more information on the same plot, we could use a third column to colour the dots by modifying the call to "scatter" above:

```
plt.scatter(df['HR'],df['SBP'], c=df['DBP'], s=40, cmap='hot')
```




## 1.2 Substituting Values

Now, let's have a closer look at the data. Print out the column "Gender":

```
df['Gender']
```

We can see a missing value ('NaN'), some likely incorrect values ('X', '2'), and some inconsistency in the labelling, (both 'f' and 'F' used to denote female). We can select the rows containing lowercase 'f' as follows:

```
df.loc[df['Gender']=='f']
```

We can fix the inconsistent case by replacing the value of 'f' by 'F' in each of these rows:

```
df.loc[df['Gender']=='f','Gender'] = 'F'
```

Print out the table to see if we've fixed the problem.

```
df['Gender']
```

## 1.3 Removing Entries

We can aggregate values over columns.  Let's say we want to know the average Systolic Blood Pressure (SBP) across the patients, we can use the mean function to aggregate the values:

```
df['SBP'].mean()
```

And if we'd like to know the average blood pressure for males and females separately, all we need to do is first "groupby" the column "Gender":

```
df.groupby('Gender')['SBP'].mean()
```

Note that we are seeing aggregate values also for the erroneous values '2' and 'X'.

*Practise 5: We might want to see average values for other variables as well, such as the Diastolic Blood Pressure (DBP). Write a groupby() on the column "Gender" for the mean() values of SBP and DBP.  Hint: You can create a list for the SBP and DBP.*

If we want to see the aggregate values for all other columns we can do that by simply not listing any columns:

```
df.groupby('Gender').mean()
```

Looking back at the previous plots. Imagine we wish to remove anomalously large values for SBP. From the plot it looks like SBP is generally below 350. Let's remove values larger than that:

```
df = df[df['SBP'] <= 350]
```

Print out the shape of the resulting table to find out how many rows are left:

```
df.shape
```

Which row(s) were removed? Print out the table to see:

```
df
```

Note that the original row numbers are still being used even though certain rows have been removed.  If you inspect it closely, you will notice that rows 7, 19, 20 and 28 are missing. For the Heart Rate (HR) column, let's assume we know that values ought to normally lie in the range 40 to 100. We could remove any values below 40, but that might remove some low but possibly correct values.

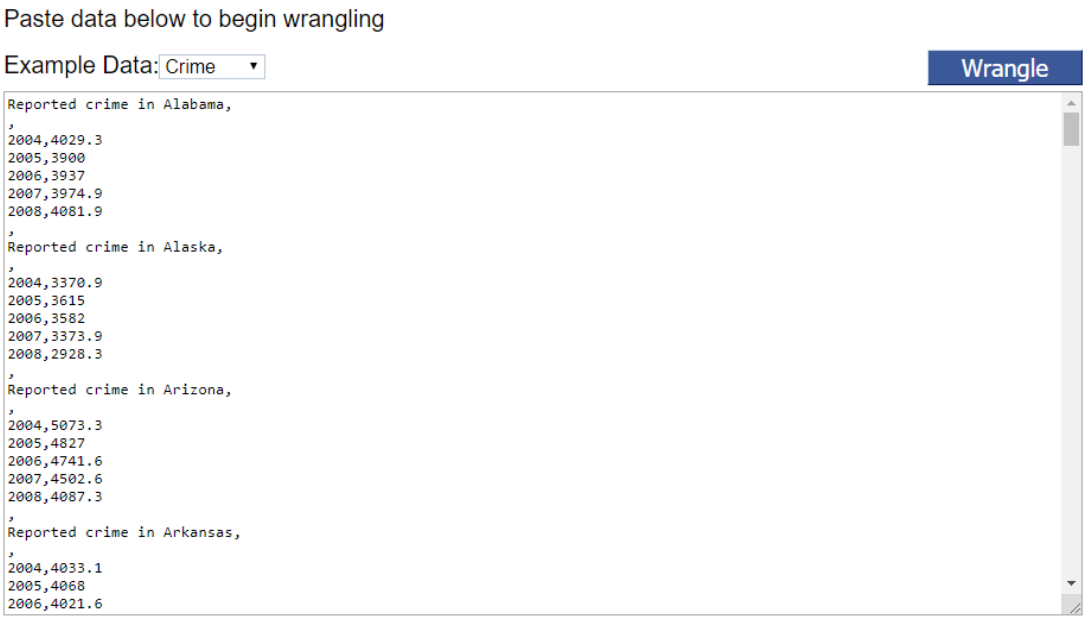
*Practise 6: Let's remove patients with a heart rate lower than 30 to be safe:*

*Practise 7: Plot out the data again to see if it's in better shape.*

The activity for this week is shorter as you will already have looked at your Assignment 1 and should start on it.  However, any general clarifications, do use the Moodle forum but you can ask your tutors on any coding issues that you may have.

## 2. Stanford Data Wrangler

This section is to allow you to explore another wrangling tool that has been successfully developed into a commercial product called Trifacta.  This was originally jointly developed at Stanford/Berkeley and you can still access a demo version of it at http://vis.stanford.edu/wrangler/ . Do watch the short demo video and then explore the tool online using the 3 examples they provided.  For a start, you can try the example data ("Crime") by clicking on the "Wrangle" button on the top right.



Upon completion of the exercise (on your own), try to copy and paste the Patients.txt date into it and try it out (note that this is not that important but it is something for you to experience).  You can also download the free version of the Trifacta software to explore.

End of Tutorial 4