# Data Handling with Python – Equifax cohort

## 1. Introduction to Data Handling with Python

### # Store this two-dimensional array

```
y = np.array([[12, 5, 2, 4],
[ 7, 6, 8, 8],
[ 1, 6, 7, 7]])
```

1. Display the first 2 rows and the first 3 columns.

2. Display the first column of y.

3. Display the first row of y.

Read the contents of file *cdc_1.csv*, containing heights, weights and ages, into array data. To do this, you can use the below code:

```
data = np.genfromtxt('data/cdc_1.csv', delimiter=',',
skip_header=1)
```

4. Calculate the standard deviation and the mean for

- height
- weight
- age

Output the results in a printed summary

**Extension -** Next, read the contents of file cdc_nan.csv, containing heights, weights and ages, into array data.

1. Separate the heights (column 0) and the weights (column 1)
2. Calculate the median for the heights and the weights and assign the values to variables.
3. What has happened? Check if the arrays contain missing values.
4. Array weights contain three nan values. Find their positions.
5. Now calculate the median for the weights ignoring the nan values.

# 2. Data Structures, Functions, & Basic Types

## Objective

In this guided task, you will:

- Perform various operations on lists and strings.
- Perform operations related to tuples, dictionaries, and sets.
- Define and call functions.

## Lists

```
ages =

[12,18,33,84,45,67,12,82,95,16,10,23,43,29,40,34,30,16,44,69,7
0,74,38, 65, 36, 83, 50, 11, 79, 64, 78, 37, 3, 8,68, 22, 4,
60, 33, 82, 45, 23, 5, 18, 28, 99, 17, 81,14, 88, 50, 19, 59,
7, 44, 93,35,72,25,63,11,69,11,76,10,60,30,14,21,82,47,6 , 21,
88,46,78,92,48,36,28,51]
```

1. Record the length of the ages list in a variable. Display the length.

2. Display the first element of the ages list.

3. Display the last element of the ages list.

4. Find the ages from the third to the last.

5. Show the first 10 ages backwards.

6. Spot the missing value in the list of odd numbers and insert it. Hint:

`.insert()`

```
# a list of odd numbers

odd = [1,3,5,7,9,11,15,17,19,21]
```

## Strings

1. Spell the greeting backwards. Hint: [::]

```
# a greeting

greeting = 'Hello'
```

2. Spell the name of the UK observing which letters should be capital and

which lowercase.

```
UK = 'UNITED KINGDOM OF GREAT BRITAIN AND NORTHERN IRELAND'
```

3. Find the position of the first occurrence of the word ‘forever’.

```
# from "Fare thee well", George Gordon Byron

byron = 'Fare thee well and if forever still forever fare thee
well'
```

## Functions

1. Having created a function that calculate the area of a circle with:
   def AoC(r):
     pi= 3.14
     return pi* (r ** 2)
2. Create function that converts Fahrenheit to Celsius,
3. Validate the datatype of the input for both functions.

## File handling

1. Read file data.txt

2. Calculate the sum, minimum, maximum, and the average of the values

in the file. Hint: All data read in from the file will be of string type

3. Output the results to a newly created file.

## 4. Statistics and Mathematics

Descriptive statistics with NumPy

```
# import necessary libraries

import numpy as np

import scipy.stats
```

1. Read the contents of file cdc_nan.csv, containing heights, weights and

ages, into array data. To do this, you can use the below code:

```
data = np.genfromtxt('data/cdc_nan.csv', delimiter=',',
skip_header=1)
```

2. Separate the heights (column 0) and the weights (column 1).

**Univariate data**

Measures of central tendency

Median

1. Calculate the median for the heights and the weights and assign the values to variables.
2. What has happened? Check if the arrays contain missing values.
3. Array weights contain three nan values. Find their positions.
4. Now calculate the median for the weights ignoring the nan values

**Bivariate data**

1. Calculate the correlation between heights and weights.
2. Then plot them with the following:

```
import seaborn as sns

sns.scatterplot( data = , x = , y =   )
```

## 5. Introduction to Pandas

### Reading in data

1. Read the file mortgage_applicants.csv, which sits in the data folder, into a variable called mortgage.

Hint: Your path will need to reflect the location of the file.

2. The same data sits in an excel file called mortgage_applicants.xlsx, in a sheet called PrevYear. Read that into another DataFrame called mortgage_excel.

3. Some weather data is held in a file called weather_data.json. Read it into a dataframe called weather.

## Changing types and parsing times

1. What data type has each column in mortgage been read in as? Use a method to find out.

2. Convert the ID column so that it is instead represented as a Unicode string. Hint: the type is denoted 'string'

3. Convert the day column of the weather Dataframe to an appropriate Type.

## Querying DataFrames

1. Compute the monthly earnings of mortgage applicants, just using Income.
2. Compute the ratio of debts to assets for each mortgage applicant.
3. Display all mortgage applicants who have a Balance greater than £1000.
4. Display all mortgage applicants who have a Balance greater than £1000 and a Debt below £50.
5. Display all loan applicants who have an Income greater than 30,000 who have a 10-year loan, and those with an Income greater than 20,000 who have a 20-year loan (together!)

## Aggregating DataFrames

1. Compute the average Balance of mortgage applicants depending on the Term of their loan.

2. Compute the average Income of mortgage applicants depending on whether they defaulted or not.

3. Compute the average Debt, Income, and Balance of mortgage

applicants based on whether they defaulted or not, and the term of their loan.

4. Add a column to the DataFrame called MeanTermIncome, which contains the mean Income of mortgage applicants based on their Term.

## Data Cleaning with Pandas

Load in the dataset renfe_trains.csv.

**Initial data inspection**

1. Inspect the columns of the DataFrame. Specifically, consider the type of each column and whether it seems reasonable. If not, investigate why.

2. It seems like we have some bad values in the price column with the value 'price'.

You can see them by using the method `.value_counts()`.

Inspect the specific rows where this is the case.

3. It looks like some sort of error has meant the column names have been fed into the data in intervals. Let's drop these rows as they are clearly an accident.

4. We can now represent price using the appropriate type. Convert it to the appropriate data type.

**Missing values**

1. Identify whether there are missing values in the DataFrame.

2. Which columns are they in?

3. Inspect some rows which contain them.

4. Drop all rows which have missing `vehicle_class` and `price` and `fare`. Hint: how='all'

5. Run the below code. What does it suggest about ticket price with respect to vehicle_class and fare?

```
df[['vehicle_class', 'fare',
'price']].groupby(['vehicle_class',

'fare']).mean()
```

6. Fill the remaining missing price values with the mean of all the prices.

7. Check you have gotten rid of all NaN values in df.

**Deduplication**

1. Use duplicated to see whether the dataset contains any duplicated rows.

2. As the dataset constitutes ticket price search results, there's a good chance duplication has come about due to the data collection method. E.g., there are many tickets available on each train.

We would typically investigate this further, but in order to see the functionality pandas offers, just remove the duplicate rows

**Extension:**

**Additional string data manipulation**

1. Capitalise the word renfe in the company column.

2. Generate a statistical summary of trips whose vehicle_class contains Turista.

3. Display all rows whose fare ends with a +

## Data Manipulation with Pandas

Load in the dataset `renfe_trains_cleaned.csv.`

**Pivot tables**

1. Use a pivot table to explore how price differs with respect to the type of fare for each destination.

**Working with time series**

1. Convert departure & arrival to a more appropriate datatype.
2. Calculate the duration of each train journey and add it as a column called duration.
3. Make departure the index of the DataFrame.
4. Sort the index low to high (earlier to later). This will make slicing possible later.
4. Create a subset of the DataFrame called madrid_to_barca which contains only journeys with origin as MADRID and destination as BARCELONA.
5. Select only those tickets in madrid_to_barca which are in the Promo category for fare and Turista for vehicle_class. Update madrid_to_barca to only contain these.
6. Compute a seven day rolling average for price for the madrid_to_barca DataFrame. Add it as a column called rolling.
   (To try after completing the combining tables section) Plot the rolling average vs. the actual values of price.

**Combining tables**

1. Read in the fare_conditions.csv file. It contains the conditions for

the type of ticket that has been purchased.

2. Add the fare conditions to the original df DataFrame.