## Exercise Background

This small application based coding exercise is ment to expose you to the use of the numpy library as well as give you a taste of tasks that you might be needed to perform during machine learning.

Usually, machine learning involves working on large data sets. This notebook will walk you through normalising the data and then dividing the data set into smaller subsets. It is recommended that while attempting each of the tasks visit the NumPy library to find the most appropriate function which can help you achieve the desired result. More often than not you will find the functions which you require prewritten in the library. The **numpy library** can be found here.

Without further ado, the first task is to mean normalise a data set. Mean normalising is a data transformation done to reduce the variations in the data set. For example, consider a data set which has integers between 0 and 10000. That is a lot of variation, and it becomes difficult to build ML algorithms on this data. So mean normalisation is done on such data, after the transformation, the mean of the data will be zero, and standard deviation will be 1. Even though the actual values of data will change a lot, but the overall variation is still kept intact. If the concept of normalisation feels a bit unclear dont worry all of this will be covered in the future sections of this program. For now, let’s concentrate on the tasks at hand.

## Task 1: Mean Normalisation:

**Question 1.1** Create a 2D of random integers between 0 and 10,000 (including both 0 and 10,000) with 25000 rows and 15 columns. This will be the dataset you will use in the notebook.

In [ ]:
```python
# import NumPy into Python
import numpy as np

# Create a 25000 x 15 ndarray with random integers in the interval [0, 10000].
x = np.random.randint(0,10000,(25000,15))

# print the shape of X
x.shape
```

Out[ ]:
```
(25000, 15)
```

In [ ]:
```python
# print the first row of X
x[0,:]
```

Out[ ]:
```
array([8446, 9430, 6651,  904, 1920, 1803, 6559, 4093, 3069, 5621, 7724,
       8695,  421, 8083, 8364])
```

Now that you created the array we will mean normalize it. The equation for normalisaing the data is given below:

$$\mathrm{Norm\_Col_i} = \frac{\mathrm{Col_i - \mu_i}}{\sigma_i}$$

where $\mathrm{Col_i}$ is the $\mathrm{i}$th column of $\mathrm{X}$, $\mu_i$ is average of the values in the $\mathrm{i}$th column of $\mathrm{X}$, and $\sigma_i$ is the standard deviation of the values in the $\mathrm{i}$th column of $\mathrm{X}$. To put it simply, to find the new value of each element, you have to subtract the mean of respective column form that value and divide the result with the standard deviation of that columns. Now the question is, Why are these operations being done column-wise? That is because usually all the procedures in ML are done column-wise. So it will be beneficial for us to develop the habit of thinking about data column-wise.

**Question 1.2** Find the mean and the standard deviation of each of the columns in the dataset. The result will be two 1D arrays with 15 elements each, representing the mean and standard deviation for each of the columns in the dataset.

In [ ]:
```
# Average of the values in each column of X
avg_cols = x.mean(axis=0)

# print ave_cols
print(avg_cols)

# Standard Deviation of the values in each column of X
std_cols = x.std(axis = 0)

# print std_cols
print(std_cols)
```

```
[4981.42124 4989.02264 5002.19796 4999.721   5006.76464 5024.81176
 4975.5938  5011.88692 4986.02916 4993.24288 5014.4846  5026.15752
 5008.6982  5003.71212 5002.01656]
[2891.84442087 2888.67468565 2885.55387889 2892.95774814 2892.81365526
 2876.07394705 2904.20301059 2889.76199057 2879.95439918 2899.28747602
 2881.03981329 2881.62071963 2883.54057146 2889.77062336 2889.36243166]
```

**Question 1.3** Print the shape of each both the arrays, they should have 15 elements each.

In [ ]:
```
# Print the shape of ave_cols
print(avg_cols.shape)
# Print the shape of std_cols
print(std_cols.shape)
```

```
(15,)
(15,)
```

**Question 1.4** Now that you have mean and standard deviation calculated, it is time to apply the transformation to the dataset.

**HINT** The broadcast property of NumPy can make this a lot easier. You can read about it here. All you have to do is create one row of transformation values and repeat them through all the values.

```
In [ ]: # Mean normalize X
        x_norm = (x - avg_cols) / std_cols
        x_norm.reshape(25000,15)
        x_norm
```

```
Out[ ]: array([[ 1.19805157,  1.53737539,  0.57139881, ..., -1.59099485,
                 1.06558211,  1.16357277],
               [-0.88159004, -1.53392927,  1.1588077 , ..., -1.36627112,
                 0.80223941, -1.6834221 ],
               [-1.23465191,  0.88690408,  1.34248127, ...,  1.43479923,
                 0.95657692,  0.33674676],
               ...,
               [-0.53613577, -0.01870153,  0.21340861, ..., -0.77671812,
                 0.76452015,  1.33627523],
               [ 0.50506824, -1.33764548,  1.21044423, ...,  0.82790644,
                 0.60118539,  0.08928732],
               [ 0.99783333, -1.45153854, -1.28474397, ..., -0.77047579,
                 1.51752109, -0.05018981]])
```

**Question 1.5** If the transformation has been performed correctly, the mean of elements in each column will be approximately 0. Also, the average of the **minimum** value in each column of X_norm and the average of the **maximum** value in each column of X_norm will have almost the same face value with opposite signs. Let's confirm if the transformation has happened correctly.

```
In [ ]: # Print the average of all the values of X_norm
        avg_norm = x_norm.mean(axis=0)
        print(avg_norm)
        # Print the average of the minimum value in each column of X_norm
        avg_min_norm = x_norm.min(axis=0)
        print(avg_min_norm.mean())

        # Print the average of the maximum value in each column of X_norm
        avg_max_norm = x_norm.max(axis=0)
        print(avg_max_norm.mean())
```

```
[ 1.03050901e-16 -4.88764584e-17 -1.39941392e-16  1.73092651e-16
 -1.07895914e-16  1.23101529e-16  1.29463107e-16  8.37996339e-18
 -1.09867671e-17  5.36903855e-17  1.11066711e-16  8.65529870e-17
  4.00168787e-17  2.68229883e-18 -2.43086107e-17]
-1.7316571374333665
1.7300595490510067
```

Be mindful that the exact values might not match since the dataset was initialized using the random function.

## Data Spliting

After data processing, it is a regular practice in ML to split the dataset into three datasets.

1. A Training Set
2. A Cross Validation Set
3. A Test Set

The ratios in which the data is split varies a bit from case to case. But the accepted standard 6:2:2 for train, test, and validation respectively. That is 60% for training data and so on. Again why is the data split or what is the signification of these smaller data sets? These questions are better left unanswered for now. The tanks assigned to you is to split the data in the given proportions randomly. For instance, if the data set had ten elements, this is how you would do it.

In [ ]:
```
# We create a random permutation of integers 0 to 9
np.random.permutation(10)
```

Out[ ]:
```
array([9, 6, 4, 0, 3, 2, 7, 5, 1, 8])
```

1. training set = 8,3,7,5,2,6
2. Cross Validation Set = 1,9
3. Test Set = 0,4

**Question 2.1** Similarly, create a 1D array representing the indexes of the rows in the dataset X_norm. U can use the `np.random.permutation()` function for randomising the indexes.

In [ ]:
```
# Create a rank 1 ndarray that contains a random permutation of the row indices
row_indices = np.random.permutation(25000)
row_indices
```

Out[ ]:
```
array([20960,  1461, 15419, ..., 24507, 17935, 18407])
```

In [ ]:
```
# Print the shape of row_indices
row_indices.shape
```

Out[ ]:
```
(25000,)
```

**Question 2.2** Split the row indexes in the needed proportions. You can use the slicing methods you have learnt in this session to make the job easier.

In [ ]:
```
# Make any necessary calculations.
# You can save your calculations into variables to use later.
train = row_indices[:15000]
test = row_indices[15000:20000]
val  = row_indices[20000:]
```

**Question 2.3** Now make use of the indexes that you made to split the data also similarly once the data is split print the shape of each of the smaller data sets. `X_train` should have 15000 rows and 15 columns. `X_test` should have 5000 rows and 15 columns. `X_val` should have 5000 rows and 15 columns.

In [ ]:
```
# Create a Training Set
x_train = x_norm[train]

# Create a Cross Validation Set
x_val = x_norm[val]
```

```
# Create a Test Set
x_test = x_norm[test]
```

In [ ]:
```
# Print the shape of X_train
print(x_train.shape)

# Print the shape of X_crossVal
print(x_val.shape)

# Print the shape of X_test
print(x_test.shape)
```

```
(15000, 15)
(5000, 15)
(5000, 15)
```