1 Introduction

For this assignment, you will write a code which implements basic data processing functionalities such as:

- Reading data from a file and "clean"ing it.
- Normalizing this data.
- Measuring the difference of each data element from the average.
- Plotting the data.

The deadline for the submission is Wednesday, 23 June 2021 (today) 22:00 Turkish Time.

NOTE that this is NOT a group-work. You should write your solution yourself and without getting help from anyone.

2 Data

The data you will be using is consumer confidence index (CCI) values. CCI is an economic indicator that measures the confidence of customers regarding their expectations, spendings and savings. This data includes monthly measurements of CCI for approx. the last 2 decades.

Sample content of the file can be seen below:

```
X,Y
1,106.1
2,104.1
3,105.4
4,104.9
5,-93.7
6,104.6
7,117.0
8,101.0
9,-26.4
10,100.7
11,99.1
12,96.1
13,-73.3
14,97.5
15,98.5
```

You will be provided a set of **X** values, representing the months, and **Y** values, representing the CCI value for those months. However, some of these measurements are noise and one of your tasks will be to eliminate them.

Note that your submission **will be tested with another CCI data**. Therefore, in your code, you should not make any assumptions based on the current CCI data. You must implement your code for the general case as outlined below.

3 Your Task

Your task is to fill in the "TODO" bits provided in the code given in the following subsection. You can copy this code or download the file named "lab9.py" through the assignment named "Lab 9" on the class4 server.

A sample data file, named `data.csv`, is available for you under the files of assignment activity and in the external emulator to test your implementations.

At this point, you have three alternatives:

1. You can work on the external emulator without making any effort for the sample file or installation of the required modules. This is recommended, since you can directly complete the implementations on the template code by making a copy-paste from the given "lab9.py" file. (*If the figure you plot is not shown automatically, click the icon left to the title "Browser Python" in the emulator page*)

2. If you have the necessary modules installed on your own computer, you can work on your computer. The sample file is available under the assignment files. Make sure that your code and the data file is under same directory.

3. Another alternative is to use Google Colab. You can download the given "Lab9.ipynb" file and go to Google Colab. From "Open notebook" under the "File" menu, upload ipynb file where the template code is already included. Here is an official documentation for uploading an external file which will be necessary to work on the data file:

   https://colab.research.google.com/notebooks/io.ipynb

   If you are not used to work on Google Colab with an external file, this alternative might be less preferable despite of its advantages for providing installed modules.

Submission: You will submit the "lab9.py" file with your solutions at the same activity on the class4 server.

Versions of Python and Libraries: Suggested version of Python is 3.7 since there may be minor incompabilities between the versions. Also here are the official versions of modules to help you to avoid any conflicts:

- pandas → 0.23.4
- numpy → 1.15.1
- matplotlib → 2.2.3

3.1 Solution Template

See Section 4 for the detailed explanations of the functions.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


# 30 points
def read_and_filter(filepath, upper_bound, lower_bound):
    #TODO Implement
    pass # pass is a placeholder statement that does nothing, you can remove it after your implementation is
           complete.

# 30 points
def plot_data(data, title, label_of_x, label_of_y, output_path):
    #TODO Implement

    # THE FOLLOWING CODE BLOCK IS CONTROL LOGIC FOR DEBUGGING AND TESTING PURPOSES
    # WRITE YOUR CODE ABOVE THIS PART
    # DO NOT DELETE OR CHANGE BELOW!
    if output_path == "":
        plt.show()
    else:
        plt.savefig(output_path)

# 10 points
def difference_with_average(arr):
    #TODO Implement
    pass # pass is a placeholder statement that does nothing, you can remove after your implementation is
         complete.

# 10 points
def normalize_the_data(arr):
    #TODO Implement
    pass # pass is a placeholder statement that does nothing, you can remove after your implementation is
         complete.

# 20 points
def get_averages_in_periods(data, period):
    #TODO Implement
    pass


# Testing function, you can change it as you wish, it will not be graded.
def test():
    try:
        arr = read_and_filter("data.csv", 100, 0)
        plot_data(arr, "CCI", "Months", "Percentage", "")
    except Exception as e:
        print("ERROR:",e)

    try:
        diff_arr = difference_with_average(arr)
        plot_data(diff_arr, "DIFFERENCE␣WITH␣AVERAGE", "Months", "Percentage", "")
    except Exception as e:
        print("ERROR:",e)


    try:
        normalized_data = normalize_the_data(arr)
        plot_data(normalized_data, "NORMALIZED␣CCI", "Months", "Percentage", "")
    except Exception as e:
        print("ERROR:",e)

    try:
        avg_data = get_averages_in_periods(arr, 3)
        plot_data(avg_data, "AVERAGE␣CCI", "Quarters", "Percentage", "")
    except Exception as e:
        print("ERROR:",e)

# Calling test function to ease your debugging.
# This condition ensures that this function will not be called when your code is imported in the actual
      evaluation.
if __name__ == "__main__":
    test()
```

3.2 Important Notes

- For this assignment you will complete the implementations of the functions (`read_and_filter`, `plot_data`, `difference_with_average`, `normalize_the_data`, and `get_averages_in_periods`). Anything you write outside of these functions will not be graded.
- The `test` function is just to show you an example run of the functions. You can change it as you like. It will not be graded.
- Do not change the signature (function names and parameters) of the functions and be careful about their return types since any function which does not comply with the specifications will get 0.
- Evaluation of each function will be carried out independently. Therefore, an incorrect implementation of a function will not affect others.
- Keep in mind that the given sample data is just just a sample and not the one which will be used for evaluating your codes. Avoid any code that is specific to the values in the given sample data.

4 Explanations of the Functions

4.1 `read_and_filter(filepath, upper_bound, lower_bound)` [30 points]

It takes a `str` argument as the `filepath` of the data file (a CSV file) and two `float` arguments as the upper and lower bounds of the noise-free data. It returns a 1D numpy array, containing the **Y** axis of the data where the noisy entities are filtered.

It should read the CSV file with the given filename. The retrieved data should have these two columns in this order: **X** and **Y**. Furthermore, this function should remove any row from the dataset where the corresponding **Y** values which are greater than the given upper_bound or smaller than the given lower_bound. Lastly, it should return the filtered **Y** values as a 1D numpy array of floats.

4.2 `plot_data(data, title, label_of_x, label_of_y, output_path)` [30 points]

This function takes a 1D numpy array and 4 strings which are the title of the plot, label of the x-axis, label of the y-axis and finally output path if the plot will be saved.

You should draw the plot by placing the given data to y-axis. x-axis must include the new enumeration of the given data starting from **1**. (you can use

`range()` function for that purpose).

You must properly configure the title and labels the of the axes and use the `plot()` function with the correct arguments. If you draw the figure with another function, such as `scatter()`, you will lose points redundantly.

You do not need to do anything about the last parameter since the control logic based on it is already handled. Output path is given as empty string in the template code for all of the function calls which will trigger the `show()` function to ease your debugging. The actual output paths will be given in the evaluation process. Do not leave any code in the function under this block.

4.3 `difference_with_average(arr)` [10 points]

This function takes a 1D numpy array and return a new 1D numpy array where the values are the differences of the elements with the average value of the original array. For example, if an element is 77.5 and the average is 76.5, then the corresponding entity will become 1.0 in the new array.

To keep the precision of the floats to 1 digit after decimal point you **must** use the `np.round()` function.

4.4 `normalize_the_data(arr)` [10 points]

In this function, you will be given a 1D numpy array and normalize it by using *min-max normalization* where its formula is given below (a is the input array and b is the new one):

$$b[i] = \frac{a[i] - \min(a)}{\max(a) - \min(a)}. \tag{1}$$

The function should return b, a 1D numpy array with the normalized values.

4.5 `get_averages_in_periods(data, period)` [20 points]

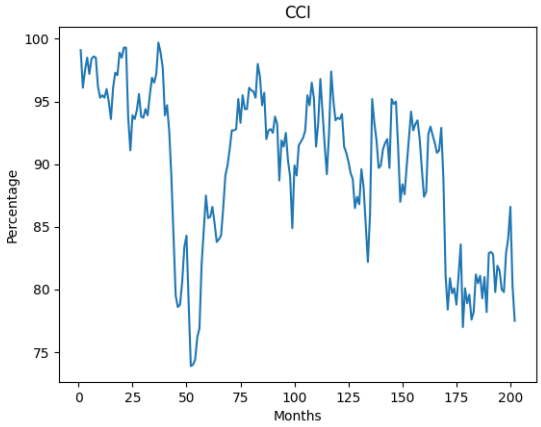
This function takes a 1D numpy array and an integer as the period and return a `list of floats` as the averages calculated within the given period. For example, if the period is given as 3, then the first float in the list is the average of the first 3 entities in the given array. The second one is the average of the next 3 entities and so on.

1 digit after decimal point is required in the elements of the returned list. Hence, you can use `round()` function for that purpose.

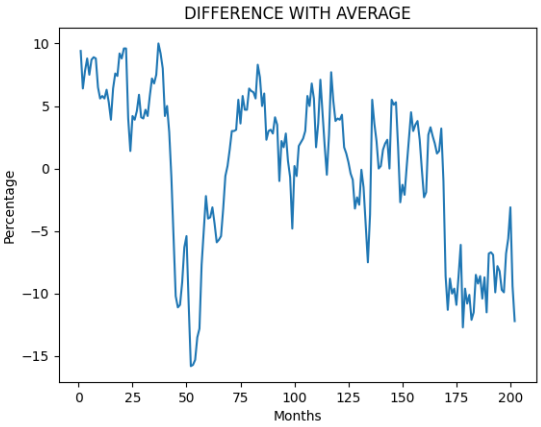
4.6 The `test()` function and Expected Results

The test function **will not be graded**. It is just to show an example usage of the other functions and to ease your testing/debugging process. You can change the body of this function as you like to test the individual functions, the arguments etc., but make sure you do not introduce any errors preventing the code from running since it will result in **0 as your grade**.

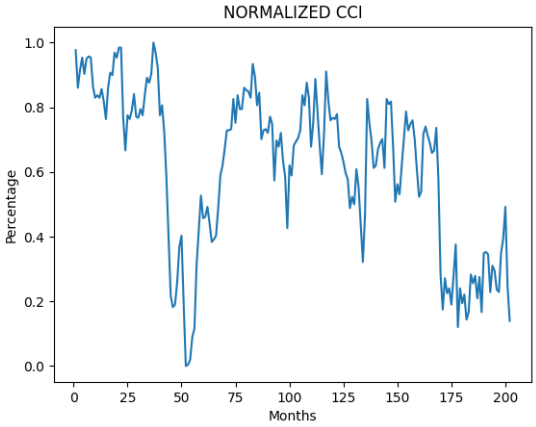
The example function reads data from `data.csv` and filters anything whose Y value is greater than 100 or smaller than 0. Then it draws the plot of noise-free data which must look like as below:



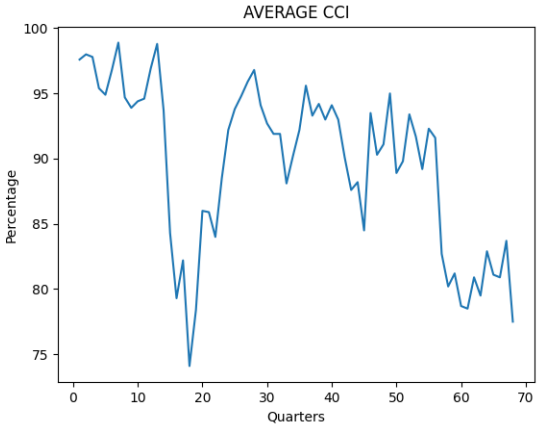
Then it calls the `difference_with_average` function and draw its data as below:



The next step is testing the normalization of the data and resulting plot must look like following figure:



Finally, the last step is calculating and drawing periodic averages:



As you may already noticed, none of the operations above change the main characteristic of the data. Therefore, the first 3 figures are actually the same plot with different values. The last one also keeps the main flow of the data, however, its plot is a little bit smoother.