Part 1 – 20% of the Mark – Deadline 5th of July 4pm

This part of the assignment consists in creating a piece of code that can read two lists of prices (given in CSV files). The first list contains current prices for some products, and the second list the corresponding reference prices for the same products. Your code should calculate the relative discount that the product currently has when compared to the reference price. For a product with current price C and reference price R, the relative discount D should be calculated as:

$$D = \frac{R - C}{R}$$

Note that if the product has increased in price (C > R) the value of D will be negative. The discounts are then classified in four categories depending on their discount level:

Value of D	Category
$D \leq 0$	`no_discount`
$0 < D \leq 0.1$	`discounted`
$0.1 < D \leq 0.2$	`good_deal`
$D > 0.2$	`buy_now`

Your task is to add code to the provided template file in order to complete the following three functions:

`read_files(current_price_filename, reference_price_filename)` – This function reads the input CSV files and return two Numpy arrays with the values in the file. These should be named `current`, for the current prices and `reference`, for the reference prices.

`discount = calculate_discount(current, reference)` – This function uses the values read by the previous function to calculate a Numpy array of the same size, that contains the discounts for each product calculated as per the above formula.

`price_classification = classify_prices(discount)` – This function uses as input the array with the discounts and outputs a list that contains, for each price, their price classification as a text string. Please note that the text strings should be exactly as described in the table (i.e., no spaces, capital letters, etc.).

Your submission should be a single Python file based on the provided template. Only Numpy is required for this part of the assignment. The code will be tested with different lists of prices, so it should be robust enough to work on a variety of valid situations. The files will always contain only numbers and be of the same size, for example, but they might contain a different number of product prices on them.

Part 2 – 80% of the Mark – Deadline 27th of July 4pm

This part of the assignment consists of three different tasks, each will require you to write a different code. Tasks 1 and 2 contribute 25% to the total mark of the module, and Task 3 contributes 30%. Your submission will be split in two parts, one report (in Word or PDF, TurnitIn submission) and three Python files (.PY, Additional Python Files submission). Details on the content and naming convention for these are given at the end of this document.

Task 1 – Data manipulation (25%):

You are given a dataset ("Vegetables.csv") which contains information on the prices of some wholesale vegetable prices for the year 2013 in the US. The data is given monthly and annually and split by several categories: whether it is organic or conventional and in which terminal the price was recorded (Atlanta or San Francisco).

Your task consists in using Pandas to read the data and create the following four functions:

1. `read_vegetable_data(filename)` – this function reads the data file provided correctly and returns a data frame "df"
2. `generate_plot_1(df)` – This function creates a plot (**Plot1**) where the X axis represents the months of 2013 and the Y axis represents the price. The plot should have one line per vegetable type (i.e. one line for Cabbage, one for Carrots, etc.) which will represent the average price of each vegetable price each month.
3. `generate_plot_2(df)` - This function creates a plot (**Plot2**) for only one vegetable type (eg. only Lettuce), that plots the average price of the vegetable for both its organic and conventional counterpart over time. The plot should have two lines (one for organic, one for conventional).
4. `find_cheapest_month(df, vegetable, organic)` – This function should return the month (as text taken from the file, for example "Jan-13") where the vegetable "vegetable" was the cheapest across all stations. If "organic" takes the value True it should refer to the organic version, otherwise to the conventional version. For example, `find_cheapest_month(df, 'Carrots', False)` should return 'Feb-13'

You must complete these making sure that:

- The dataset is not altered in any way (i.e. it is not modified in Excel or similar before reading into Python).
- Invalid data "N\A" is ignored, you might remove rows when they have some missing data.
- You follow the provided Python template to write your functions, without changing their name or input arguments. Once the functions are filled in, running the code should provide all the outputs required.

It is likely that you will need to consult the official Pandas documentation to complete this task. You must provide a small commentary (200 words) describing the advanced features of Pandas you have used for this, and how you handled the challenges that appeared (i.e. invalid or missing data, etc.).

Task 2 – Dice rolling (25%):

In this task you should create a function called "`dice_probabilities(n_dice, replications)`" that is able to estimate the probability of all possible outcomes of rolling a given number of dice (`n_dice`) by simulating a number of random rolls (`replications`) and observing the output. If there is more than one dice, the output is considered to be the sum of all the dice. The dice are considered standard six side dice.

The function should return two Numpy arrays (or Python lists): `dice_values, dice_probabilities`. In `dice_values` the outcome of the dice roll should be stored, whereas `dice_probabilities` should contain the probability of the corresponding roll to happen. The probabilities must be numbers between 0 and 1 (where 1 represents 100%). Note that these should be estimations and will not be precise, especially for low values of "`replications`".

You are given a template file to code your function, and here we provide two examples of possible outputs of the code

For n_dice = 1 and replications = 1000:

```
dice_values: [1 2 3 4 5 6]

dice_probabilities: [0.173 0.172 0.176 0.148 0.164 0.167]
```

For n_dice = 2 and replications = 1000000

```
dice_values: [ 2  3  4  5  6  7  8  9 10 11 12]

dice_probabilities: [0.027858 0.056126 0.083222 0.110896 0.138488 0.166662 0.138903
0.111272 0.083266 0.055206 0.028101]
```

Note that these probabilities can be calculated exactly, but this is not required by this task. The answer **must** be a simulation as described.

Task 3 – Vaccination appointment allocation (30%):

In this task you are required to build a piece of software that can allocate patients to a vaccination centre, taking into account the availability of the centre for the day, the distance between the centre and the patient and whether the patient is eligible for a vaccine or not.

You will be given two datasets in CSV format, with the following headers:

Patient dataset:

Patient dataset: `patient_id, x_coord, y_coord, eligible`

Centre dataset: `centre_id, x_coord, y_coord, capacity`

The IDs are natural numbers to identify patients and centres your solution should refer to these when reporting the allocation. The coordinates are given as numbers between 0 and 100. For the patients, eligible will be 1 if they are eligible to have a vaccine and 0 if they are not. Only eligible patients should be allocated. The capacity of the centre refers to the daily capacity, the number of patients they can vaccinate each day. On top of these, your code will have a parameter "`days`" (already given in the template) which gives the maximum days in which the vaccine can be administered. For example, if `days = 3`, a patient must be assigned to be vaccinated in day 0, day 1 or day 2.

The allocation must be done as follows:

Going through the patient list in the given order, eligible patients must be allocated to the closest centre that has capacity to vaccinate them. Within that centre, they should be assigned to the earliest possible day.

The vaccination must be completed in a certain timeframe, which is given as the parameter "`days`" to your function. This means that a centre might be completely full at some point and unable to accept any more patients, even if it is the closest to them. The distance between the patients and centres must be calculated as the Euclidean distance. If (p_x, p_y) are the coordinates of the patient, and $(c_x, c_y)$are the coordinates of the centre, the distance between them, d, is given by:

$$d = \sqrt[2]{(c_x - p_x)^2 + (c_y - p_y)^2}$$

The code should follow the template provided on Blackboard without altering its structure, and the task consists in completing the following three functions:

`read_data(file_patients, file_centres)`, which reads the two CSV data files and returns its contents as numpy arrays: patients and centres.

`euclidean_distance(x1, y1, x2, y2)`, which implements the above formula for the distance.

`allocation(patients, centres, days)`, which depends on the data read by the `read_data` function and the parameter "days", which is a natural number. This function must implement the allocation algorithm described above and return the following three variables: `patient_allocation, centre_usage, total_distance`

- `patient_allocation`: should be a numpy array with as many rows as eligible patients and four columns. For each patient, the first column must contain its ID, the second column the centre to which they are allocated, the third column the day in which they have been allocated and the fourth column the distance they have to travel to reach the centre and the fourth column. For example, the row `[6, 1, 3, 31.06]` means that the patient with ID 6 has been allocated to the vaccination centre 1 on day 3, and need to travel 31.06 units to reach it.
- `centre_usage`: should be a numpy array with as many rows as centres present in the CSV file for centres and as many columns as days given by the parameter "`days`". An element `centre_usage[i,j]` of the array should contain the number of patients allocated to centre i on the day j.
- `total_distance`: should be a single number, which is the sum of all the distances all eligible patients have to travel (one way only) to get their vaccine.

Note that there might be better ways to create a shorter/fairer allocation, but for this task you are required to implement the algorithm exactly as described above. Your code might be tested with different input files and values of "`days`".

It can happen that for some combinations of input files and "`days`" there is not enough capacity to vaccinate all patients. In this case your code should still run and allocate as many patients as possible following the proposed algorithm. The patients that are unable to be allocated (which will be the last on the list) should still get a row in `patient_allocation`, where the first column has their ID and the rest have the value "-1".

Further instructions:

There are two submission points for this assignment, a short report to be submitted in TurnitIn and two Python files to be submitted in the Blackboard entry labelled "Additional Python files".

The report can be in Word or PDF and the structure must be as follows:

Task 1:

Task 1 commentary – Max 200 words

[Task1 Plot 1]

[Task1 Plot 2]

Appendix:

Source code for Task 1

Source code for Task 2

Source code for Task 3

The Python files **MUST** be based on the templates that have been provided for the assignment, and should be named as follows: “task_1_STUDENTID.py”, “task_2_STUDENTID.py” and “task_3_STUDENTID.py”, where “STUDENTID” must be replaced by your Student ID number. The templates should not be modified in their structure, and the code should only be added inside the provided functions, where the comment:

```
# Please, introduce your answer here
```

is located. Note that this might be in more than one place throughout the file. You should **NOT** modify the name of the functions, their input arguments or the name/quantity of return. Doing so might lead to your assignment being incorrectly marked. All the template codes should run as provided, but will provide dummy answers (normally set everything to zero, nor ‘Not completed yet’). The lines that you might change have been labelled by:

```
# Change/remove this line
```

Or similar. You might also choose to keep those lines and modify the variable values later on in your function.

Only Numpy, Pandas and Matplotlib are allowed/required to complete the assignment. These have already been imported appropriately in the template files. If you are in doubt of what you might or might not modify in the template, please contact one of the module instructors.