F.A.Q.

1) **Can I use NumPy?**
You can use NumPy if you wish to. You'll not get more marks for using it given this was not part of CE705. You must make sure your functions still receive and return variables using the data types in the assignment brief. For instance, if a function is supposed to return a list of lists and you return a NumPy array you will lose all marks related to this function.

2) **Can I import modules?**
You can import any modules that come with Python (eg. math, os, etc). You cannot use any module that requires extra installation (eg. Pandas). The only exception to this rule is NumPy (see question 1).

3) **Can I make a small change in the return type of a function?**
No. If a function is supposed to return a number, say 5, and you return '5', [5] or something other than just 5, you will lose all marks related to this function.

4) **Can I make a small change in the data type of a parameter of a function?**
No. Such changes will lead to you losing all marks related to the function in question.

5) **Can I add or remove a parameter of a function?**
No. Such changes will lead to you losing all marks related to the function in question.

6) **Can I make a minor change in the name of a function?**
No. Such changes will lead to you losing all marks related to the function in question. Please note that Python is case-sensitive. For instance, the name run_test is not the same thing as Run_Test.

7) **Can I implement extra functions to make my code easier/cleaner?**
Yes. Please note you must implement all the functions in the assignment brief, if you'd like to implement more functions that is fine.

8) **Can I implement the algorithm in this assignment in any other way than what the assignment brief describes?**
No. In this assignment we are trying to measure your ability to code a programme following a specification. Hence, you must follow this specification.

9) **Why am I not allowed to make changes?**
Large pieces of software (eg. Windows) are not written by a single programmer, but by many. All programmers will be working on different parts of the software, but all of these parts are likely to interact in some way. The programme specification makes sure everybody knows what each function expects to receive and what each function should return. If one programmer unilaterally decides to make a small change that goes against the specification… then the software will not work as expected.

Marking Scheme

Characteristics of an excellent project (70% or more):

- Excellent code documentation
- Excellent use of Python's native methods and code standards
- Excellent use of relevant data types
- Follows carefully the specification provided
- Implements the described run_test, which shows the expected results.
- Excellent code optimisation in terms of memory, speed and readability
- Generally, an excellent solution, carefully worked out;

Characteristics of a good project (60%):

- Good code documentation
- Good use of Python's code standards
- Good use of relevant data types
- Follows the specification provided, with no deviations.
- Implements the described run_test, which shows the expected results.
- Good code optimisation in terms of memory, speed and readability
- Generally a good solution, which delivers what the final user would expect.

Characteristics of a fair project (50% or less):

- No meaningful code documentation
- Code tends to be more verbose than actually needed or at times difficult to read
- No real thought on the relevance of data types
- Does not follow the specification provided (this alone will indicate a fail).
- Does not implement run_test as described, or this does not show the expected results.
- A solution that only seems to deliver what the final user would expect.

Please note:

- You must submit only one file
- You must follow the instructions for each function in terms of parameters and returns
- You should document your code.
- It is a good idea to test each function at a time, and only afterwards test the whole project

Assignment: Automatic Cancer Diagnostic

It is quite expensive to determine whether a particular patient has cancer or not. With this in mind, your local NHS trust hired a software developer to design and develop a software capable of automatically making this diagnostic. This software will read CSV files with measurements taken from blood tests of patients and produce a diagnostic.
Unfortunately, for unknown reasons this software developer had to leave mid-project. The NHS trust has now hired you to complete the project.

Technical details:
You'll be using different data structures to accomplish the below. In the text any mention to *matrix* should be read as a list of lists. Your assignment must contain the code for the functions below (you may wish to read the classification algorithm in the appendix first):

load_from_csv
This function should have one parameter, a file name (including, if necessary, its path). The function should read this CSV file and return a *matrix* (list of lists) in which a row in the file is a row in the *matrix*. If the file has n rows, the matrix should have n elements (each element is a list). Notice that in CSV files a comma separates columns (CSV = comma separated values). You can assume the file will contain solely numeric values (and commas, of course) with no quotes.

get_distance
This function should have two parameters, both of them lists. It should return the Euclidean distance between the two lists. For details about this distance, read the appendix.

get_standard_deviation
This function should have two parameters, a matrix and a column number. It should return the standard deviation of the elements in the column number passed as a parameter. For details about how to calculate this standard deviation, read the appendix.

get_standardised_matrix
This function should take one parameter, a matrix (list of lists). It should return a matrix containing the standardised version of the matrix passed as a parameter. This function should somehow use the get_standard_deviation function above. For details on how to standardise a matrix, read the appendix.

get_k_nearest_labels
This function should have four parameters: a list (a row of the matrix containing the data of the file data), a matrix (list of lists) containing the data from the file learning_data, and a matrix containing the data of the file learning_data_labels, and the last parameter is a positive integer *k*.
This function should find the *k* rows of the matrix learning_data that are the closest to the list passed as a parameter. It should somehow use the get_distance function to do so. After finding these *k* rows, it should find and return the related rows in the matrix learning_data_labels.

For example: *if* k=3, and the function finds that the closest rows to the list are the rows 13, 26, and 34 from the matrix learning_data_labels, then it should return a matrix (list of lists) containing solely the rows 13, 26, and 34 from the matrix of "learning_data_labels".

get_mode
This function should have one parameter, a matrix (list of lists). This matrix will have only one column (which will be returned by get_k_nearest_labels). This function should return the mode of the numbers in this matrix. The mode of a sequence of numbers is the number with the highest frequency (the one which repeats the most).
The mode of the sequence 5, 5, 4, 5, 4 would be 5.
If more than one number has the highest frequency, then you should return one of the numbers with the highest frequency at random.
In the sequence 5, 5, 4, 5, 4, 1, 4 the numbers 4 and 5 have the highest frequency. You should return either 4 or 5 at random.

classify
This should have 4 parameters:
- The matrixes for data, learning_data, and learning_data_labels. For the algorithm to work correctly the matrixes data and learning_data should be standardised beforehand.
- k, a positive integer

This function follow the algorithm described in the appendix. It should return a matrix; in this document we call this matrix data_labels.

This function should use the other functions you wrote as much as possible. Do not keep repeating code you already wrote.

get_accuracy
This function should have two parameters. A matrix containing the data from the file correct_data_labels, and a matrix containing the matrix data_labels (the output of the function classify)

This function should calculate and return the percentage of accuracy. If both matrixes have exactly the same values (in exactly the same row numbers) then the accuracy is of 100%. If only half of the values of both tables match exactly in terms of value and row number, then the accuracy is of 50%, etc.

run_test
The aim of this function is just to run a series of tests. By consequence, here you can use hard-coded values for the strings containing the filenames of data, correct_data_labels, learning_data and learning_data_labels.

This function should create one matrix for each of these: correct_data_labels, learning_data and learning_data_labels (using load_from_csv). It should standardise the matrix data and the matrix learning_data (using get_standardised_matrix). Then, it should run the algorithm (using classify) and calculate the accuracy (using get_acuracy) for a series of experiments. In each experiment you should run the algorithm (and calculate the accuracy) for different values of *k* (go from 3 to 15 in steps of 1), and show the results on the screen. For instance, if with *k* = 3 the accuracy is 68.5% it should show:
k=3, Accuracy = 68.5%
Again, it should do the above for the values of *k* from 3 to 15 (inclusive), in steps of 1.

More details

The aim of this software is to classify each row of the data matrix (representing the measurements of one patient). The algorithm you should use is explained in the appendix.

The number of columns of the matrix learning_data and the matrix data must match. The number of rows of the matrix learning_data and the matrix learning_data_labels must also match. The matrix learning_data_labels and the matrix data_labels (the latter, generated by your software) must have a single column. The value of *k* cannot be larger than the number of rows in the matrix learning_data, and it must be positive (note: zero is not a positive number).

Your software should follow the algorithm described in the appendix and generate a new matrix data_labels. This matrix should have a single column and the same number of rows as the matrix data. The entry (x,1) of the matrix data_labels will contain an integer representing the label of the row x of the matrix data. A zero means patient x does not have Cancer and a one means the opposite.

Notice: the matrix data_labels and the matrix correct_data_labels should each have a single column, and they should have the same number of rows.

Appendix

<u>Data standardization</u>

$$matrix(x,y) = \frac{matrix(x,y) - avg\big(matrix(:,y)\big)}{std\big(matrix(:,y)\big)}$$

where, *x* is a non-negative integer representing a row number, y is a non-negative integer representing a column number, avg(matrix(:,y)) is the average of column y over all the rows, and std(matrix(:,y)) is the corrected sample standard deviation of column y over all the rows.
The corrected sample standard deviation of the column y over all rows, matrix(:,y), is given by:

$$std\big(matrix(:,y)\big) = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\Big(matrix(i,y) - avg\big(matrix(:,y)\big)\Big)^2},$$

where N is the number of rows in the matrix.

<u>Classification algorithm</u>

1. Set a positive value for *k*.
2. For each row x in the matrix data
   a. Find the *k* rows in the matrix learning_data that are the closest to the row x in the matrix data according to the Euclidean distance (see below).
   b. Given the rows found on Step 2(a), find the values of the same row numbers in the matrix learning_data_labels.
   c. Given the values found in Step 2(b), find the most common value (that with the highest frequency). If two (or more) values have the same frequency, choose one of them at random.
   d. Set the row x of the matrix data_labels to the value found in Step 2(c).

Euclidean distance:
The Euclidean distance between a list_a and a list_b (both of size M) is given by:

$$d = \sqrt{\sum_{y=1}^{M}\big(list_a(y) - list_b(y)\big)^2}$$