1. **K-Nearest Neighbors (KNN) Classifier**

   (a) Import the packages that you are going to use and print their versions before starting your analysis. Remember to use comments to clarify your code as you go through your analysis. Please refer to the Style Guide for Python Code for details on how to utilize comments correctly and other styling issues.

   (b) Import *your own* KNN classifier that you implemented in the course to work on data released by the High Time Resolution Universe Survey. The README.txt file contains all the relevant information. The label column of this data set is named "ispulsar". Note that the KNN classifier should have fit and predict functions, as we discussed during the lecture.

   (c) Inspect the distribution of the features across the data. Plot each feature against each other. In these plots, please present points that belong to class 0 with the color blue and class 1 with the color orange. You can use Matplotlib, Pandas, Seaborn packages, or any other package of your preference.

   (d) Split the data into train and test samples. This data set contains 17898 samples. I suggest you use a sub-sample of the data. Suggestion: training data with shape (5000, 8) and test data with shape (500, 8).

   (e) Create various KNN models with different hyperparameter choices, i.e., number of nearest neighbors and different distance metrics, such as Euclidean or Manhattan. Fit train data and predict the labels of test samples using these models.

   (f) Calculate the number of correct predictions, wrong predictions, and fraction of correctly predicted samples (also known as the accuracy score) for each model. Discuss the suitability of the accuracy score for this data set.

   (g) Display the confusion matrix. Investigate precision, recall, and f1-score evaluation metrics and report their results. `https://scikit-learn.org/stable/modules/model_evaluation.html#`

   (h) Perform a cross-validation procedure for values of k = 1, 2, 3, 5, 10, 15, 20. Split the training data into five folds. For each value of k, use 4 folds as the training data and the remaining 1 fold as the test data. You should permute the folds also, i.e., each fold should be used as the test data once. Calculate and report the accuracy for each k and for all folds. Calculate the mean and standard deviation of the accuracies for each k and for all folds. Plot the k values vs accuracy graph. Overplot mean accuracies and their standard deviations. Which k value do you prefer, according to this analysis?

---

2. **Stochastic Gradient Descent**

   Create a 2D grid with coordinates changing from -8 to 8 for each dimension with a step size of 0.1 and evaluate the function

$$F(x, y) = 1.8 - e^{-0.1\left[2.5(x+3)^2+(y+3)^2\right]} - 1.5e^{-0.05\left[2.5(x-3)^2+(y-3)^2\right]}$$

   on the grid. Write a program that applies the stochastic gradient descent with a batch size of 10 (you will use 10 points from the grid at each iteration to find the gradient). Use `numpy.gradient` to calculate the gradient of the function, and use `RegularGridInterpolator` of `scipy.interpolate` to find the gradient at a specific point and/or at specific points. Use three different initial points and make a contour plot to show the updated values. Experiment with various learning rates and determine what seems reasonable. Based on your experiments, list the advantages and disadvantages of this method.
