COMP 4448: Data Science Tools II Assignment 4

Sean Wendlandt

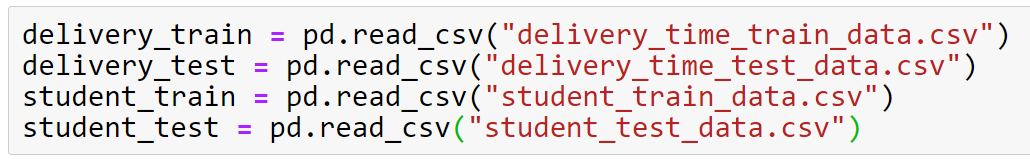
**Directions:** Do this assignment in Jupyter Notebook and provide screenshots of code and output in this word document wherever required. You will upload this word document containing screenshots of code and answers as well as your Jupyter Notebook to Canvas. All assignments will be submitted and graded through canvas and grades will be transferred to the 2U platform.

**Goal:** The goal of this assignment is to give you the opportunity to implement K-NN from scratch as well as using tools built into sklearn. You would also compare your algorithms with other algorithms.

**Packages:** Core packages you may need for this assignment include numpy, pandas, sklearn, matplotlib.pyplot and/or seaborn.

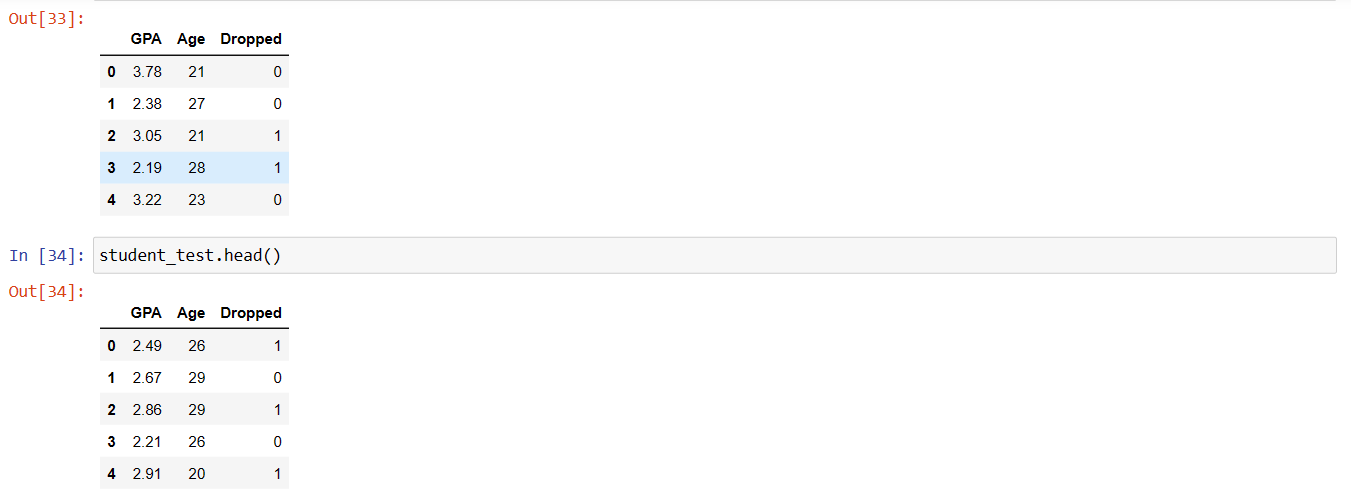
**Question 1**

1. Read the data provided on canvas into Python as follows:

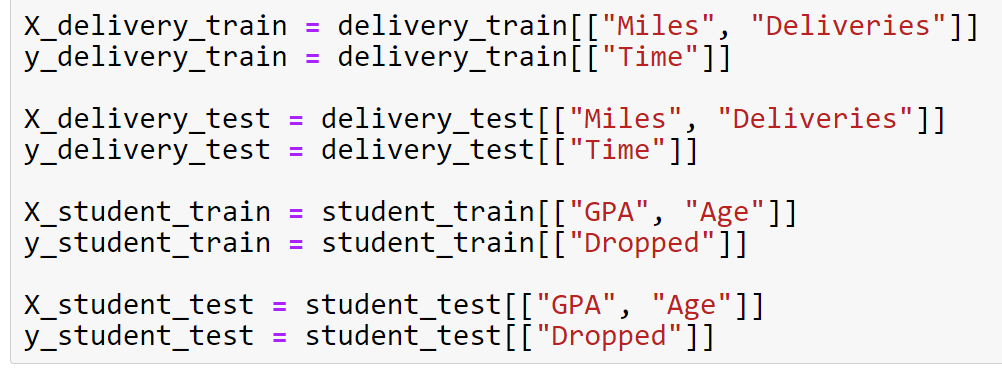


Print the first 5 rows of each dataset to inspect the nature of data

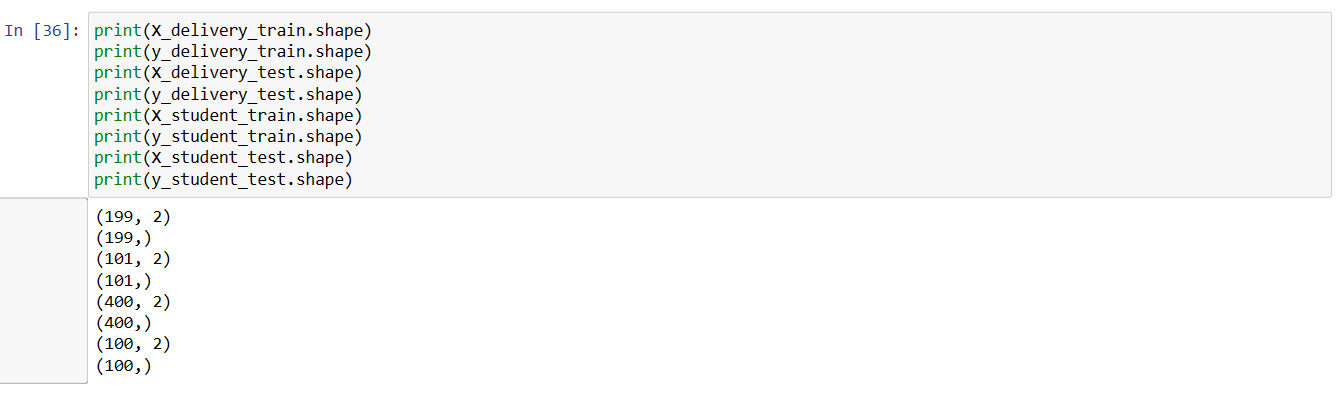




1. Split the data into training and test set as follows:



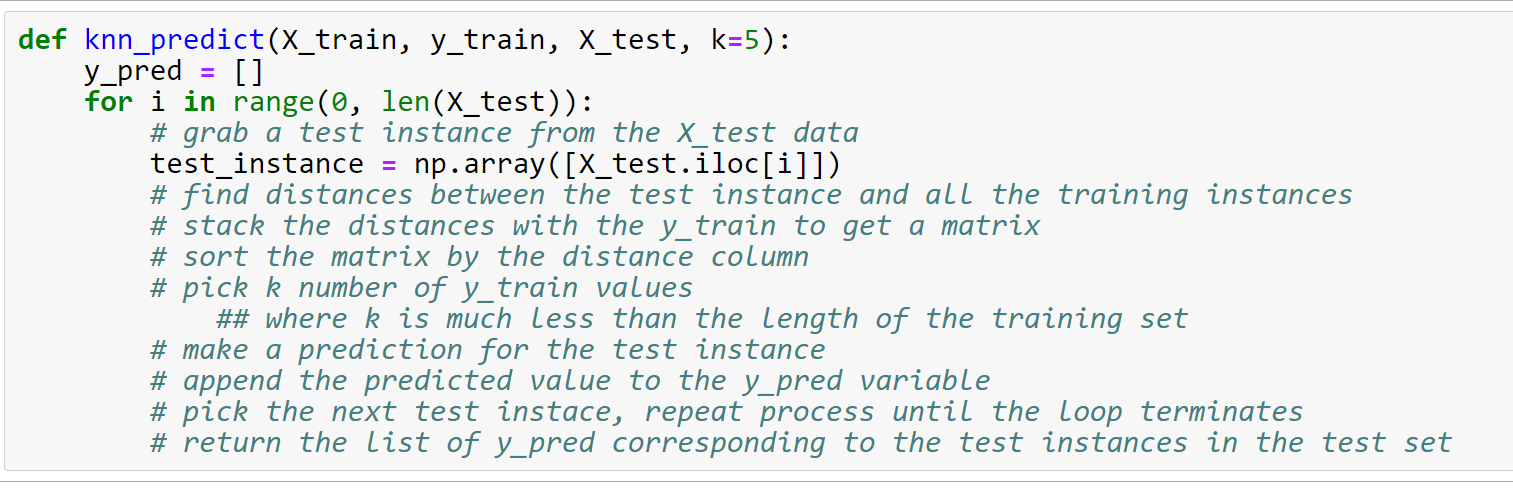
Print the shape of input and output data for the training and test sets



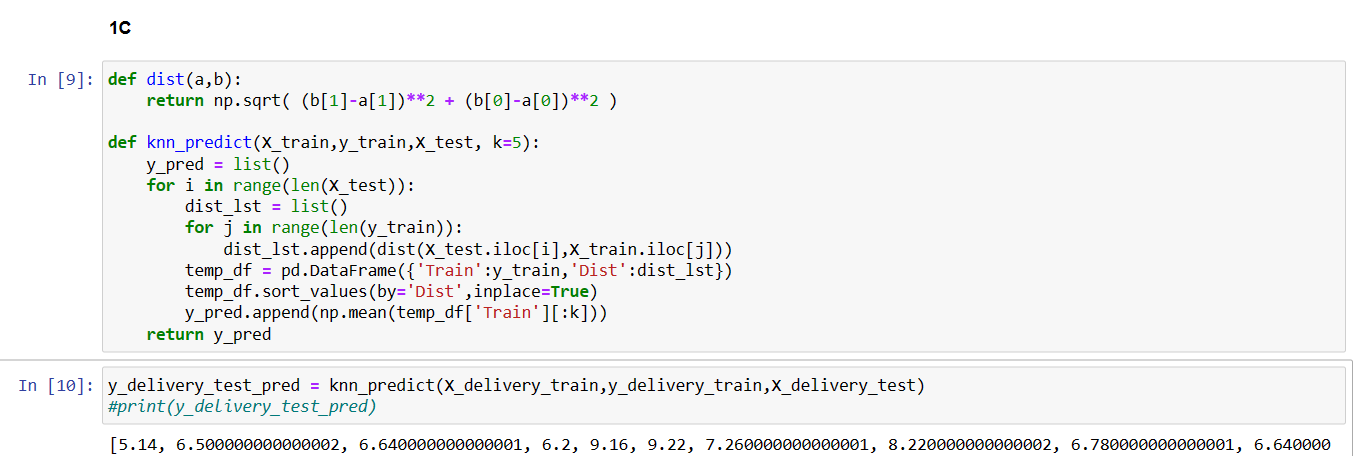
1. You would use the delivery data for regression: that is, miles (“Miles”) and number of deliveries (“Deliveries”) would be use an input to predict the total time it takes for a delivery truck to deliver Amazon packages. You would write a k-nearest neighbor algorithm function that learns from the training set and predicts the output or y values of given test instances. The input that the function takes include:

* X\_train: Pandas DataFrame or 2D NumPy array
* y\_train: Pandas Series or 1D NumPy array
* X\_test: Pandas DataFrame or 2D NumPy array
* K: the number of nearest neighbors. The default value should be k=5 if no k value is passed during function call.

Use the following pseudo code to implement the algorithm (also see class notes on ideas for writing knn from scratch to see the details of each step in the body of the code). Here is the pseudo code:



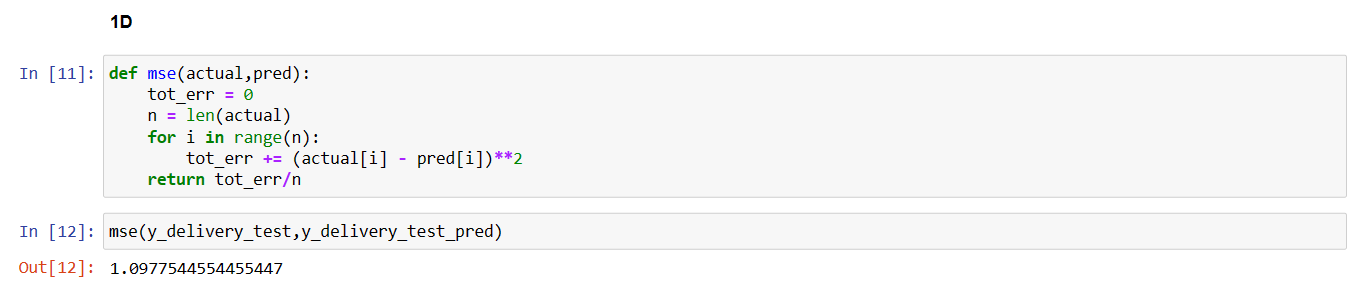
Call the knn\_predict() function and pass the X\_delivery\_train, y\_delivery\_train and X\_delivery\_test to compute the predicted output values for the X\_delivery\_test set. You can reference the results using a variable, **y\_delivery\_test\_pred.**



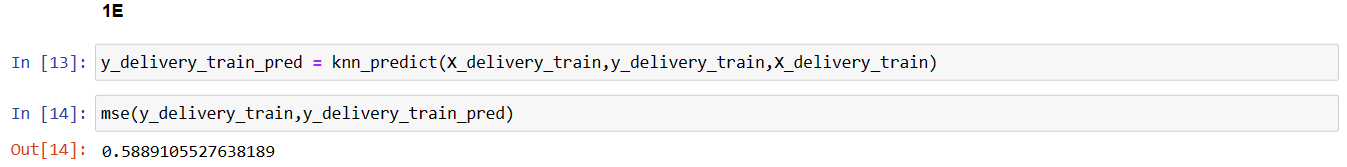
1. Write a function that computes the mean squared error to measure the performance accuracy of your knn algorithm. The input that goes into the function is the actual output values(y) and the predicted output values (y\_pred). The mean squared error (mse) is calculated as follows:

mse = where n = length of y or length of y\_pred

You can name the function **mse.** Then call the function to compute the mse for the test set for the delivery data. That is, use **mse(y\_delivery\_test, y\_delivery\_test\_pred).**

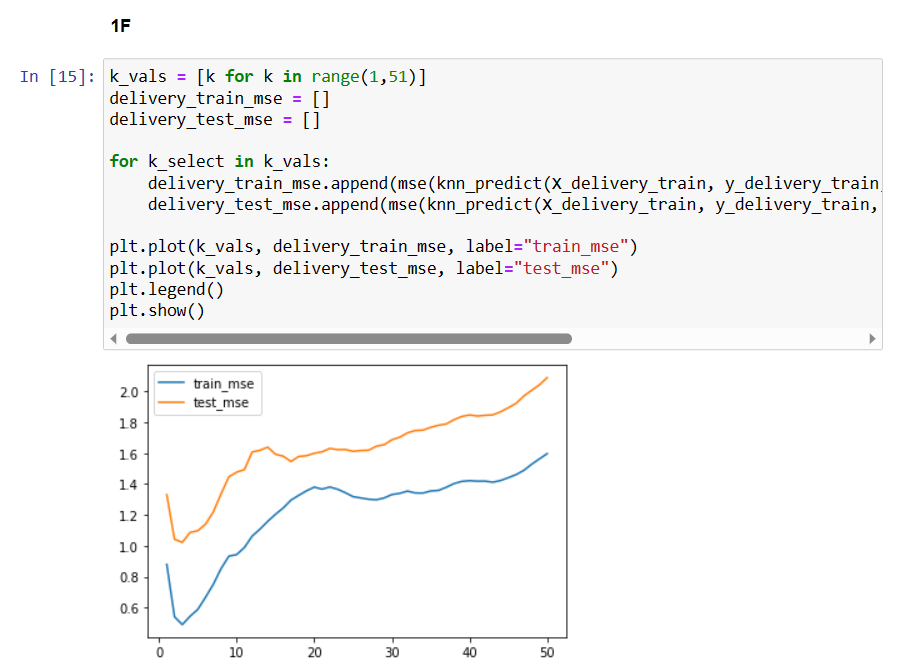


1. Does your algorithm overfit the data? To answer this question, you need to compute the accuracy or mse of the algorithm on the training set, and compare with the accuracy or mse you got for the test set. You will need to start with making predictions for the training set. Note that you will need to pass the training set into the X\_test parameter of the knn\_predict() function, in addition to the other arguments of the function. When you get the predicted values for the training set, use the mse() function to get the training mse.

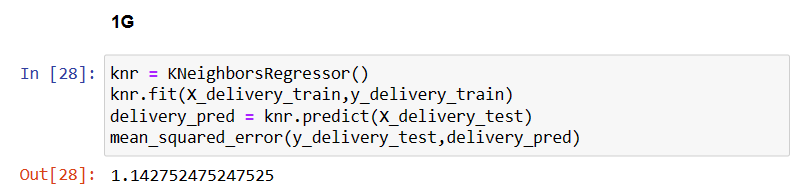


While the MSE is low, however not low enough to make the argument that the algorithm overfits the data.

1. Now, tune the hyperparameter, k, of your algorithm. Specify a grid of hyperparameter values from k=1 to k=50. Use a for-loop, then inside the for-loop body, use the knn\_predict() function as well as the mse() function to compute the mse values for the training set and test set for different values of k in the hyperparameter grid. Plot the training and test error rates of the model versus the grid of hyperparameter values.

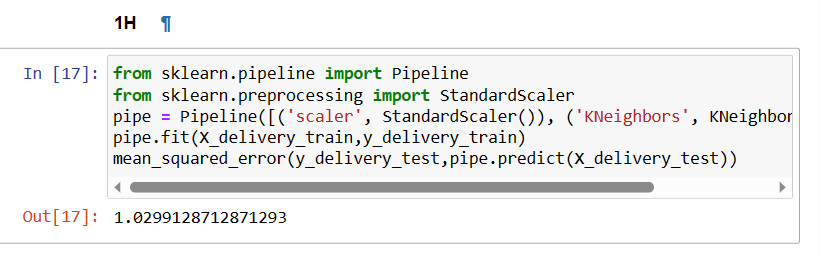


1. Now, construct and fit a k-nearest neighbor model using the KNeighborsRegressor() estimator in sklearn.neighbors module. Use the mean\_square\_error() function inside the sklearn.metrics module to compute the mse of the model on the test set. Is this similar to the mse value you got from the algorithm function you wrote from scratch?



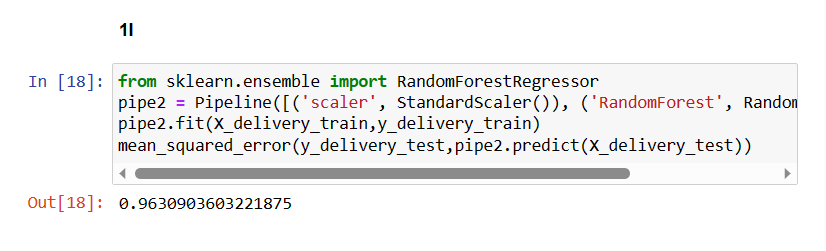
The MSE is similar to the function I wrote from scratch although this is slightly larger.

1. Using the Pipeline constructor inside the sklearn.pipeline module, scale the X\_delivery\_train with the StandardScaler(), and also construct the KNeighborsRegressor() inside the Pipeline(). Then the call the fit() method on the pipeline object to fit the model into the delivery training data. Transform the X\_delivery\_test data using StandardScaler() and use the **mean\_squared\_error** () function inside the sklearn.metrics module to compute the mse of the model on the test set. Is this mse similar to the mse you got when you used the unscaled input data in (g) above?



It is similar but has a slightly lower MSE indicating a better fit with scaled data.

1. Use the scaled X\_delivery\_train and X\_delivery\_test data and the RandomForestRegressor constructor inside the sklearn.ensemble module to construct/fit a random forest model, the compute the mse for this model. Is this mse similar to the one you got in question (h) above?



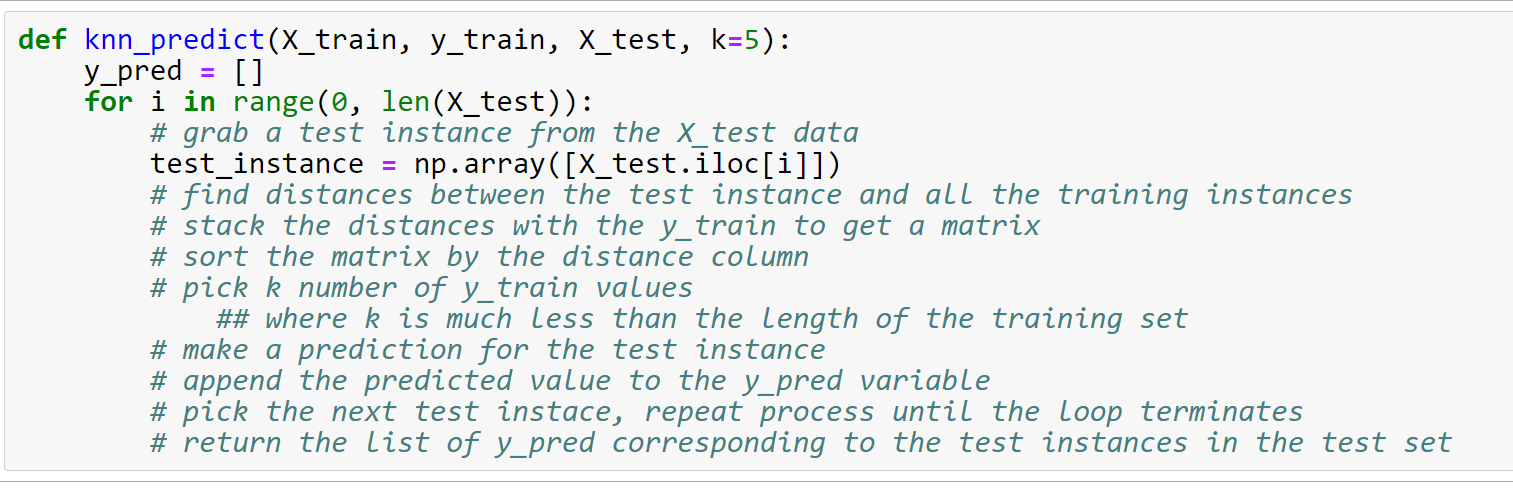
The MSE is again similar but slightly lower indicating the random forest model may be a better model for this data.

**Question 2**

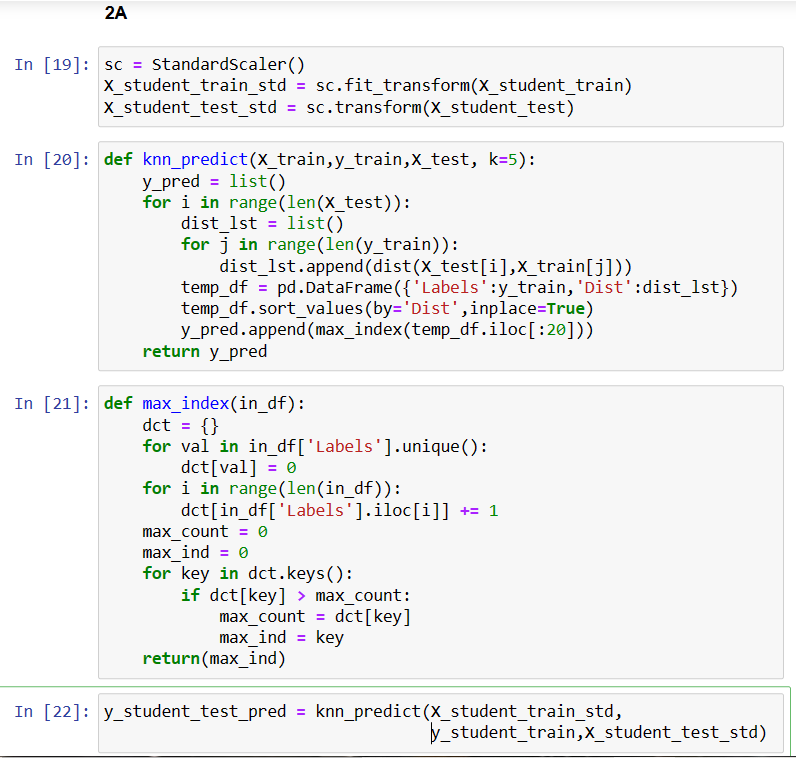
1. You would use the student data for classification: that is, GPA and Age would be used as input to predict wither a student will drop out of school (Dropped=1) or not drop (Dropped=0). You will write a k-nearest neighbor algorithm function that learns from the training set and classifies given test instances to class 0 or class 1. The input that the function takes include:

* X\_train: Pandas DataFrame or 2D NumPy array
* y\_train: Pandas Series or 1D NumPy array
* X\_test: Pandas DataFrame or 2D NumPy array
* K: the number of nearest neighbors. The default value should be k=5 if no k value is passed during function call.

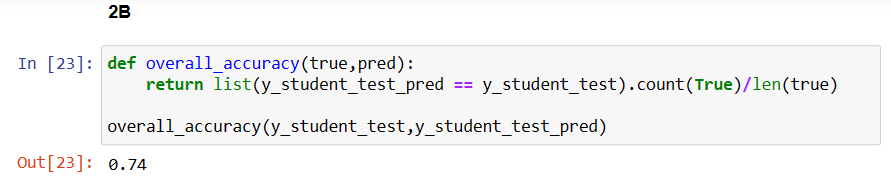
Use the following pseudo code to implement the algorithm (also see class notes on ideas for writing knn from scratch to see the details of each step in the body of the code). Here is the pseudo code:



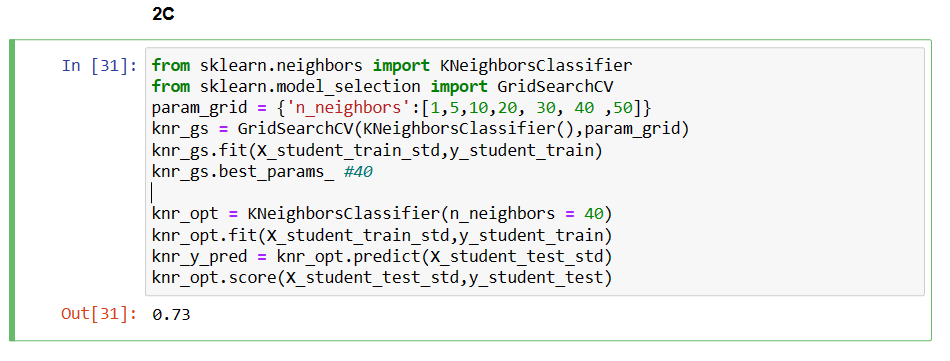
Scale the X\_student\_train and X\_student\_test data, and pass these scaled data sets in addition to y\_student\_train data into the knn\_predict() function call to compute the predicted output values for the X\_student\_test set. You can reference the results using a variable, **y\_student\_test\_pred.**



b) Define a function called **overall\_accuracy,** that computes the overall accuracy of your algorithm. You can let your function compare the actual values (y) and the predicted or classified y values (y\_pred), for example: overall accuracy could be computed using **mean(y==y\_pred)**

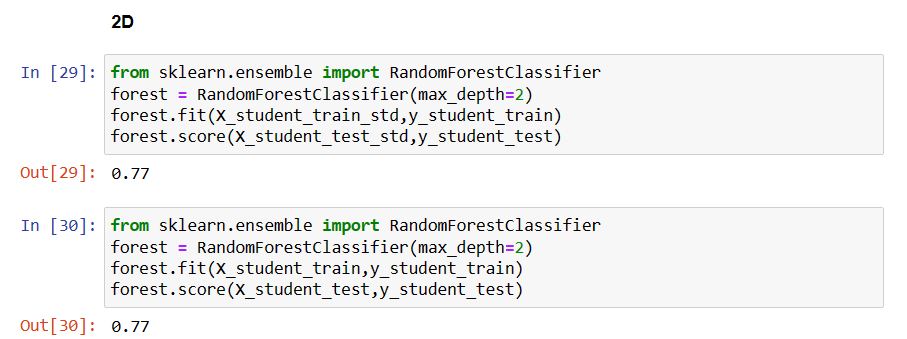


c) Use cross validation in scikit learn to construct a k-nearest neighbor classifier. Is the score obtained through cross validation similar to that obtained using your knn algorithm from scratch?



Yes they are quite similar, my k-nn algorithm from scratch performed slightly better which is surprising.

d) Also build another model using a random forest classifier and compute the accuracy score on the test set. Which of these models is the better, the model is question 2c or this model in question 2d? Why or why not?



Based on these outputs the Random forest (2D) performs better. This is in line with the above MSE outputs which also indicate the fit is slightly better using this model.