In this homework, I used three algorithms to predict the performance column in the csv file. The algorithms that I used were naive bayes, decision trees, and K nearest neighbors. The features that I used were age, failures, free time, and absences. I followed a very similar logic flow for each of the algorithms. First, I read the data from the csv, then separated the features from the target column, used train_test_split on the features and target to get my training and testing data, fit the xTrain and yTrain on the model using the given model, predicted the features, and then calculated precision, recall and F1 scores. You can see each of these algorithms separated into its own python file. I also printed out to the console the precision, recall, and F1 scores. I wanted to use three fairly different algorithms when calculating my scores to see how much the results differed. For each algorithm, my precision, recall, and F1 scores were around the same, indicating that the models are able to predict a balanced portion of positive and negative cases. However, naive bayes seemed to perform the worst out of all these algorithms, having a precision, recall, and F1 score around 0.3. The most probable explanation for this is that naive bayes assumes independence between class variables. For example, one of the features that I used was the free time column and having independence between free time and performance would mean that performance does not depend on the amount of free time that someone has. K nearest neighbors and decision trees don't assume independence, which is a probable explanation as to why the other two algorithms performed better. Intuitively thinking, a person who has had a history of failing courses, getting bad grades, missing classes, etc is definitely more probable to perform worse than someone with a stellar academic record. Therefore, it is not reasonable to assume that all of these features are independent. In this particular case, that is a significant limitation of naive bayes. K nearest neighbors performed the second worst having a precision, recall, and F1 score around 0.35 and this could be due to the data having a lot of noisy data or high dimensionality. Since I only used four features, high enough dimensionality shouldn’t be a concerning factor, as four features is a pretty small dataset. So maybe the data that I used to had a lot of noisy features in it which caused it to not be as good as decision trees. Assuming that the data has some noisy features in it, this would make sense that the decision tree algorithm performed better than K nearest neighbors, having a precision, recall, and F1 score around 0.4, because decision trees are able to handle noisy data. They also work very well in high dimensional data sets, but again I don't think that really applies in this particular case since I only used four features. It is surprising that all of the algorithms that I chose did not perform as well as I was expecting. This is most likely due to me not using a lot of features and perhaps more data could have improved the performance of the models. In the future, it would be interesting to explore some unsupervised learning models such as k-means to be able to make a prediction about performance, though this may not be the best approach since supervised learning is much better for labeled data, which is why I chose not to use it. Since I don't have any machine learning background I chose not to explore any deep learning algorithms for this assignment, but in the future, it would also be interesting to see some of the outcomes of deep learning models and how much more accurate they might be. Finally, it would be interesting to use around 20 different algorithms and see how all of their precision, recall, and F1 scores compare to see which one would be the best in this particular case.

Table of Results on next page…

Table of Results:

	Naive Bayes	KNN	Decision Tree
Precision	0.30776515151515	0.35249042145594	0.4131652661064
Recall	0.39833822091887	0.35047747950974	0.3993157380254
F1	0.31153641679957	0.35095137420719	0.3893939393939