TASK 1: For Task 1 I imputed the data set using SimpleImputer(). Within SimpleImputer(), I selected the strategy 'median' to impute the NaN values in each attribute with the attribute's median value. I specifically chose 'median' over 'mean' or 'most_frequent' after comparing all 3 strategies with a simple decision tree and seeing which one performed the best. Naturally, this might not be the case with another dataset, where mean or most_frequent might perform better, but that was my reasoning in this instance. I considered using IterativeImputer() to impute multiple times, but due to the NaN values in the dataset being a VERY small proportion of each attribute, I felt that SimpleImputer() was enough to get a 'close enough' value and that the difference between these 2 methods wouldn't be significant at all. This is also why I was happy with settling on 'median' imputation: in a dataset with around 20% NaN values, using this method could shift the spread of the data towards the centre-values, but again, with the NaN's being such a small proportion, this felt fine.

TASK 2: For this task, my chosen method of feature selection was Recursive Feature Elimination (RFE), a method that works by Step-wise Attribute Elimination, recursively removing the least important feature in the dataset until it reaches the user defined amount of features to keep, which in this case was 10. I chose RFE over a Univariate method like SelectKBest as I felt that it would perform better at picking out features due to its recursive nature, running the model in each recursive step based on the new feature set sans the least important feature removed from $step_{i-1}$, whereas SelectKBest in this instance would select the 10 best features based on the initial feature set only. Where I could've improved my choice here would be by implementing Relief: both RFE and SelectKBest suffer from the same issue which is that, if feature1 is the best and also happens to be highly correlated with feature2, both methods would select feature2 to keep as well, however the information gained from feature2 wouldn't be much due to this correlation, so this is something I would change in a future experiment.

TASK 3: After running my 4 models, the results returned in terms of classification performance ranking were as expected, with the ordering being RandomForest(Mean = 72.1%, SD = 2.8%), PrunedDT(Mean = 65%, SD = 3.8%), DecisionTree(Mean = 60.9%, SD = 4.9%), DecisionStump(Mean = 58%, SD = 4.5%). In terms of why this is the case, I believe the results were as expected due to the nature of these models. With RandomForest, of all the models, it has the greatest access to the overall hypothesis space of the data. This is due to it being an ensemble method, so by being able to create 10 decision trees instead of one and then returning the best result, it should naturally outperform the other 3 models. While DecisionTree and PrunedDT have equal access to the hypothesis space, PrunedDT should perform better as it's able to remove branches in the tree that might be leading to the tree overfitting, which is why when introduced to the test set, it gains performance over the DecisionTree. Finally, DecisionStump comes in last as it has the least access to the hypothesis space. With the data having such an even class distribution, I believe the reason DS is performing quite a bit better than 50/50 (relatively) is due to our feature selection, meaning this model can choose and decide via a feature that carries greater decidability than just 'coin-flip'.

TASK 4: After adding 20% additive noise to my data, DT and DS didn't improve nor worsen, staying at mostly the same classification accuracy (DT(Mean = 60%, SD = 4.3%), DS(Mean = 58.9%, SD = 3.6%)), however RF and PrunedDT both lost between 0.5-0.6% accuracy (RF(Mean = 71.6%, SD = 2.7%), PrunedDT(Mean = 64.4%, SD = 5.1%) , with all results being compared to TASK3. As this drop in accuracy for PrunedDT and RF isn't statistically significant, and DT/DS are basically unchanged, this is causing me to hypothesize that additive noise has low/no effect on the dataset, at least in this experiment. I think this is because, while we **are** changing the values in our data, the overall

distribution and pattern of this data remains quite unchanged, therefore the models are able to adapt to the modified data as they can still pick up on the general pattern that leads to classification.

TASK 5: Then we added 20% multiplicative noise and observed the performance of our 4 models. In a similar trend with what I noticed after adding 20% additive noise, DT & DS remain unshaken in their accuracy (DT(Mean = 60.5, SD = 3.6%), DS(Mean = 58.2, SD = 3.5%)), while again, RF and PrunedDT lose accuracy, this time by between 1.6-1.9% (RF(Mean = 70.4%, SD = 4.0%), PrunedDT(Mean = 63.1%, SD = 4.5%)). Again, the loss in accuracy isn't by a statistically significant margin on its own, however when observing TASK3 VS TASK5 performance and TASK3 VS TASK4 performance, we **can** see that there's a difference in multiplicative noise vs additive noise. I believe this is the case due to multiplicative noise creating a larger change in the values of the dataset, which while possibly still maintaining a similar distribution to the original data, is probably also making it harder for the models to find a 'best fit' to the data for classification in comparison to a totally clean dataset.

TASK 6: Upon adding 5% class noise to the dataset, the decrease in accuracy for the models is quite significant compared to the decreases observed in TASK4&5. Again, DT & DS hover around very similar classification values (DT(Mean = 59.9%, SD = 3.1%), DS(Mean = 58.2, SD = 2.8%)) but this time, RF and PrunedDT take significant hits to their classification accuracy relative to the prior performance drops, ranging from 4.3-4.5% (RF(Mean = 67.8%, SD = 2.2%), PrunedDT(Mean = 60.5%, SD = 4.3%)). My hypothesis as to why the drop in accuracy was so large for class noise in comparison to additive and multiplicative noise is that by flipping a portion of the class labels, we're effectively changing the position of these values while simultaneously having the overall data-pattern remain the same. I believe that by doing this, we're tricking the models into making different assumptions about what is classified as a '1' and what is classified as a '0' due to examples in the data that previously would've classified as '1' now classifying as '0' (and visa-versa), throwing off the models understanding of the data.

TASK 7 (TrainSet Noise): To start with this task, I must note that how I've approached CV in this task is incorrect, as it's cross-validating the model against the original dataset, and not against (trainNoise+test, y) or (train+testNoise, y) and therefore the CV in this task is invalidated. Now in regard to the performance of the models with Training set noise, the ranking still remains the same, and the change in DT(Mean = 60.9%, SD = 2.9%) is more or less the same as well. DS(Mean = 59.9%, SD = 4.4%) however gains an improvement of 1.9%, which while not statistically significant, is still an improvement compared to its performance in the last 4 tasks. RF and PrunedDT lose accuracy, ranging from 0.65-2.1% (RF(Mean = 70%, SD = 1.7%), PrunedDT(Mean = 64.3%, SD = 4.3%)). I surmise that the reason the models performed the way they did in this instance was because when adding noise to only the training set, the overall pattern in the data in comparison to the test set is different, and therefore where the classifiers learnt to classify an example as '0' based on the observed pattern in its attributes, upon arrive at the test set with a different data pattern but the same classification, the models slipped up more and made incorrect classifications.

(TestSet Noise) When noise was added to the test set, again, RF and PrunedDT recorded losses in accuracy (RF(Mean = 69.9%, SD = 2.9%), PrunedDT(Mean = 63.4%, SD = 4.1%)) but this time DT and DS both gained in accuracy, scoring (DT(Mean = 61.4%, SD = 3.6%), DS(Mean = 59%, SD = 4.4%). Much akin to my hypothesis for the trainSet noise in TASK7, I believe the models behaved in a similar way, making more misclassifications due to the pattern change in test data compared to the pattern in the training data. As to why performance for DS and DT slightly improved, while I believe DS improved simply because it cannot overfit to the data and is therefore more resilient to noise in the test set, I'm actually unsure as to why DT gained where PrunedDT lost, as DT would be more overfit.