Exercise 1

We define entropy of a class Y as:

H(Y) = - Σ_{k, i=1} [P(Y=y_{i}) * log2(P(Y = y_{i}))]

While the conditional entropy of a class Y as:

H(Y|X) = - Σ_{u, j=1} [P(X = x_{j}] * Σ_{k, i =1} [P(Y=y_{i} | X = x_{j}) * log2(P(Y=y_{i} | X = x_{j}))]

Here the classes are Virus (we will call it Y=1) and Not_Virus (we will call it Y=0).

Looking at the train set (Table 1) we can see that:

- P(Y = 0) = ¾ (Not virus)
- P(Y = 1) = ¼ (Virus)

Replacing those values at H(Y) we can can calculate entropy at the root (before split):
H(Y) = - ((P(Y=0) * log2(P(Y=0))) + (P(Y=1) * log2(P(Y=1)))) = -(¾*log2(¾) + ¼*log2(¼)) ~= 0.81127

In order to assess which attributes (features) to choose as the root of the decision tree, we need to calculate the information gain (IG) and choose the one with the highest IG. In case of equal IG we choose one at random.

IG(X) = H(Y) - H(Y | X)

We can name Body Length as **W**, Bold letters as **K** and susp. Address as **Z.**
Thus we have:

- **IG(W) = H(Y) - H(Y | W)**
- **IG(K) = H(Y) - H(Y | K)**
- **IG(Z) = H(Y) - H(Y | Z)**

Since **W** is a continuous variable, we need to define a threshold. We can choose mean, median or any other metric as long as we explain the reason. Let us go for the mean in this case because median might not give us wanted results due to the small sample size (median is robust against outliers but in small datasets does not capture variability while).

mean(W) = (23 + 18 + 43 + 68)/4 = 38. Thus 38 will be our threshold

For susp. address, we will also set Yes = 1 and No = 0. Thus we can compute probabilities.

From table 1:

- P(W > 38) = ½ , P(W <= 38) = ½
- P(K = 0) = ¾, P(K = 1) = ¼
- P(Z = 0) = ½, P(Z = 1) = ½

- P(Y = 0 | W > 38) = ½ , P(Y = 1 | W > 38) = ½
- P(Y = 0 | W <= 38) = 1, P(Y = 1 | W <= 38) = 0
- P(Y = 0 | K = 0) = ⅔ , P(Y = 1 | K = 0) = ⅓
- P(Y = 0 | K = 1) = 1, P(Y = 1 | K = 1) = 0
- P(Y = 0 | Z = 0) = ½ , P(Y = 1 | Z = 0) = ½
- P(Y = 0 | Z = 1) = 1 , P(Y = 1 | Z = 1) = 0

Now we can calculate conditional entropy:
(In all calculations involving entropy we define 0*log(0) to be 0.)

- H(Y | W) = - P(W > 38) * [P(Y = 0 | W > 38) * log2(P(Y = 0 | W > 38)) + P(Y=1 | W > 38) * log2(P(Y = 1 | W > 38))] - P(W <= 38) * [P(Y = 0 | W <= 38) *log2(P(Y = 0 | W <= 38)) + P(Y = 1 | W <= 38) * log2(P(Y = 1 | W <= 38))] =
  = - ½ * [½ * log2(½) + ½ * log2(½)] - ½ *[1*log2(1) + 0*log2(0)] =
  = -½ * [½ * (-1) + ½ * (-1)] -½ * [1*(0) + 0] =
  = -½ * [-½ -½] =
  = -½ * -1 = ½ = 0.5

- H(Y | K) = - P(K = 0) * [P(Y = 0 | K = 0) * log2(P(Y = 0 | K=0)) + P(Y=1 | K=0) * log2(P(Y = 1 | K=0))] - P(K = 1) * [P(Y = 0 | K = 1) *log2(P(Y = 0 | K = 1)) + P(Y = 1 | K = 1) * log2(P(Y = 1 | K = 1))] =
  = -¾ * [⅔ * log2(⅔) + ⅓ * log2(⅓)] - ¼ * [1*log2(1) + 0*log2(0)] =
  = -¾ * [⅔ * (-0.585) + ⅓ * (-1.585)] - 0 =
  = -¾ * [-0.39 - 0.5283] =
  = -¾ * (-0.9183) = 0.688725

- H(Y | Z) = - P(Z = 0) * [P(Y = 0 | Z = 0) * log2(P(Y = 0 | Z=0)) + P(Y=1 | Z=0) * log2(P(Y = 1 | Z=0))] - P(Z = 1) * [P(Y = 0 | Z = 1) *log2(P(Y = 0 | Z = 1)) + P(Y = 1 | Z = 1) * log2(P(Y = 1 | Z = 1))] =
  = -½ * [½ * log2(½) + ½ * log2(½)] - ½ * [1 * log2(1) + 0 * log2(0)] =
  = - ½ [½ * (-1) + ½(-1)] - 0 =
  = -½ * (-1) = 0.5

- IG(W) = H(Y) - H(Y | W) = 0.81127 - 0.5 = 0.31127
- IG(K) = H(Y) - H(Y | K) = 0.81127 - 0.688725 = 0.122545
- IG(Z) = H(Y) - H(Y | Z) = 0.81127 - 0.5 = 0.31127

Since IG(W) and IG(Z) have the same information gain, we can choose either one. Let us choose IG(Z) as the root attribute for the split since it is binary and the threshold we chose might not necessarily be the best. In general, what we could do is try different thresholds and move on to the decision tree by using each one at a time. For the purpose of the exercise we will only choose IG(Z) to make the decision tree.

For Z = 0, we need a further split since classes are not seperated. The samples in the training set where Z = 0 are s2 and s4. So once more we need to look for the variable that outputs the maximum information gain while excluding Z.

The new table:

	W	K	Y
s2	**18**	**1**	**¬Virus**
s4	**68**	**0**	**Virus**

Again for W we will choose mean as threshold, so mean(W_new) = (18+68)/2 = 43.

- **IG(W) = H(Y) - H(Y | W)**
- **IG(K) = H(Y) - H(Y | K)**

P(Y = 0) = ½, P(Y=1) = ½
P(W > 43) = ½ , P(W <= 43) = ½
P(K = 0) = ½, P(K=1) = ½
H(Y) = - ((P(Y=0) * log2(P(Y=0))) + (P(Y=1) * log2(P(Y=1)))) =
= -(½ *log2(½) + ½ * log2(½)) = -(log2(½)) = 1

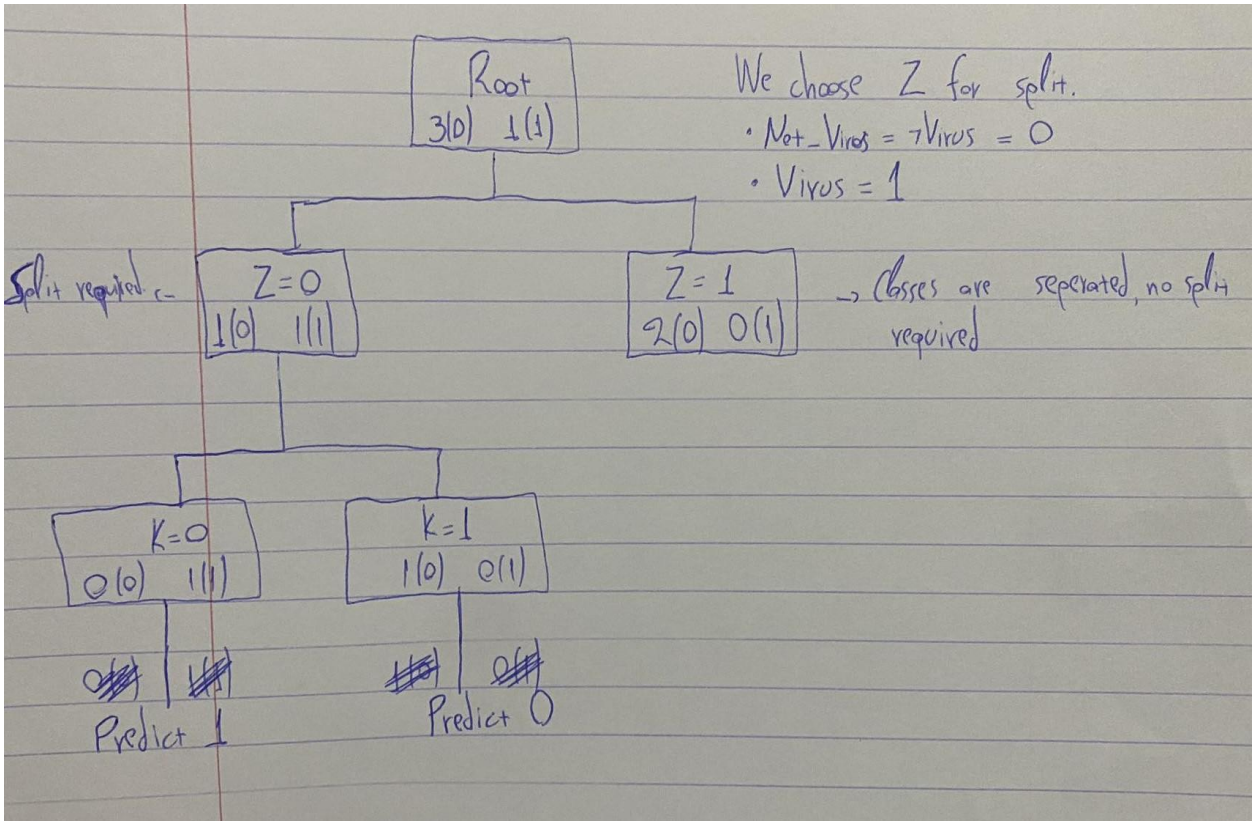
P(Y = 0 | W > 43) = 0, P(Y = 1 | W > 43) = 1
P(Y = 0 | W <= 43) = 1 ,P(Y = 1 | W <= 43) = 0
P(Y = 0 | K = 0) = 0, P(Y = 1 | K = 0) = 1
P(Y = 0 | K = 1) = 1, P(Y = 1 | K = 1) = 0

H(Y | W) = - P(W > 43) * [P(Y = 0 | W > 43) * log2(P(Y = 0 | W > 43)) + P(Y=1 | W > 43) * log2(P(Y = 1 | W > 43))] - P(W <= 43) * [P(Y = 0 | W <= 43) *log2(P(Y = 0 | W <= 43)) + P(Y = 1 | W <= 43) * log2(P(Y = 1 | W <= 43))] =
= -½ * [0 * log2(0) + 1 * log2(1)] -½ * [1*log2(1) + 0*log2(0)] = 0

H(Y | K) = - P(K = 0) * [P(Y = 0 | K = 0) * log2(P(Y = 0 | K=0)) + P(Y=1 | K=0) * log2(P(Y = 1 | K=0))] - P(K = 1) * [P(Y = 0 | K = 1) *log2(P(Y = 0 | K = 1)) + P(Y = 1 | K = 1) * log2(P(Y = 1 | K = 1))] =
= -½ * [0 * log2(0) + 1 * log2(1)] -½ * [1*log2(1) + 0*log2(0)] = 0

- **IG(W)** = 1 - 0 = 1
- **IG(K)** = 1 - 0 = 1

Again we have the same IG so we can choose any one. We will once again choose the binary **K** (Bold letters) for threshold uncertainty reasons, especially since the samples now are even less (2). So the tree overall is like this:



To compute the accuracy, we calculate:
testaccuracy = #correctlyclassified/#all

So first of all, let us predict classes for the test set (s5-s8). If Z = 1, then we predict 0 (no virus)

- s5 : Z=1 so we predict 0 (not virus – True: Virus)
- s6 : Z=0, K = 1 so we predict 0 (not virus – True: not virus)
- s7 : Z = 1 so we predict 0 (not virus – True: Virus)
- s8 : Z=0, K = 0 so we predict 1 (virus – True: not virus)

Out of all, we only predicted 1 correct, that would mean that:
testaccuracy = ¼
In this case we could try different split attributes and different thresholds to try and improve the accuracy, but in general when we have low samples this is not so uncommon.

Exercise 2

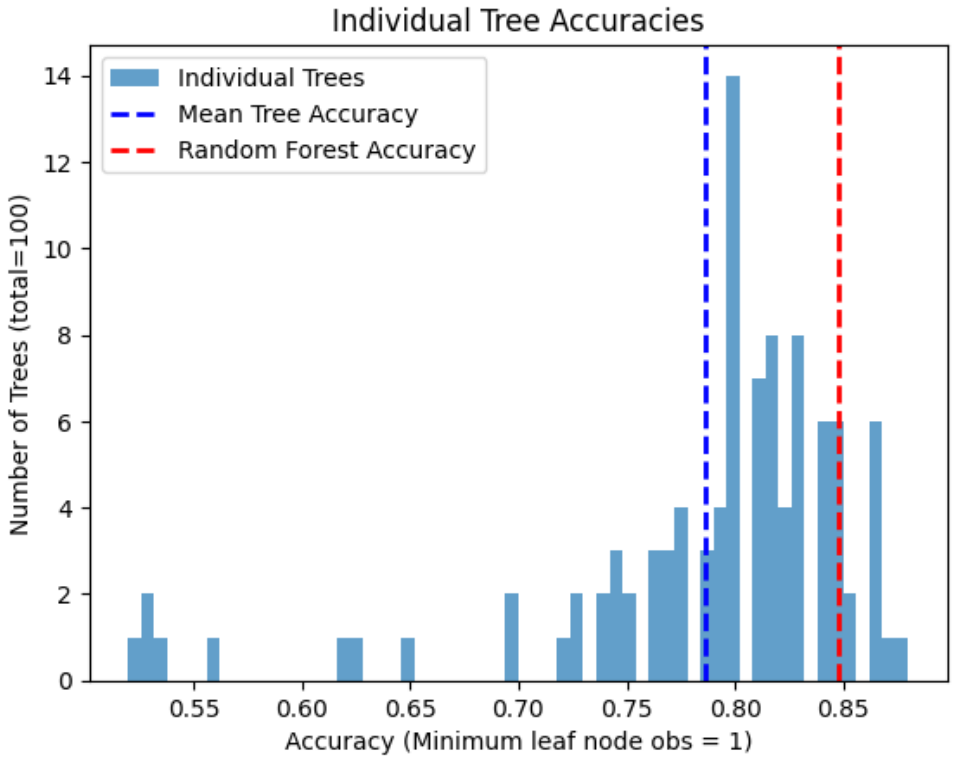
The first function ***def_TrainRF(X, Y, n_trees, min_samples_leaf)*** uses as input the features (X), the classes (Y), the number of trees we want to create in the Random Forest, and the minimum number of leaf node observations. The function uses a dictionary with "trees" and "features" as keys. The former will contain the tree classifier obtained from *sklearn DecisionTreeClassifier* and the latter the features used (as index) to create the classifier. First, we find the number of features that will be used at each split, which we will calculate based on a common heuristic, which is the square root of the total features we have. So if we had 9 features, 3 would be selected.
Next we need to create the bootstrapped dataset (which is permutation with replacement) in order to introduce variability to our Random Forest (*ind = np.random.choice(X.shape[0], size=X.shape[0], replace=True*). We then use the outputs of the shuffled indices, to get the appropriate values from both X and Y dataframes. We select the random features (different for each tree) and then we create the tree classifier through the sklearn package. After that, we train it to obtain our tree using the bootstrapped data, and keep both the tree and features used in the dictionary. This is done for a number of time equal to the initial *n_trees,* each time having a different bootstrapped dataset, and different features.

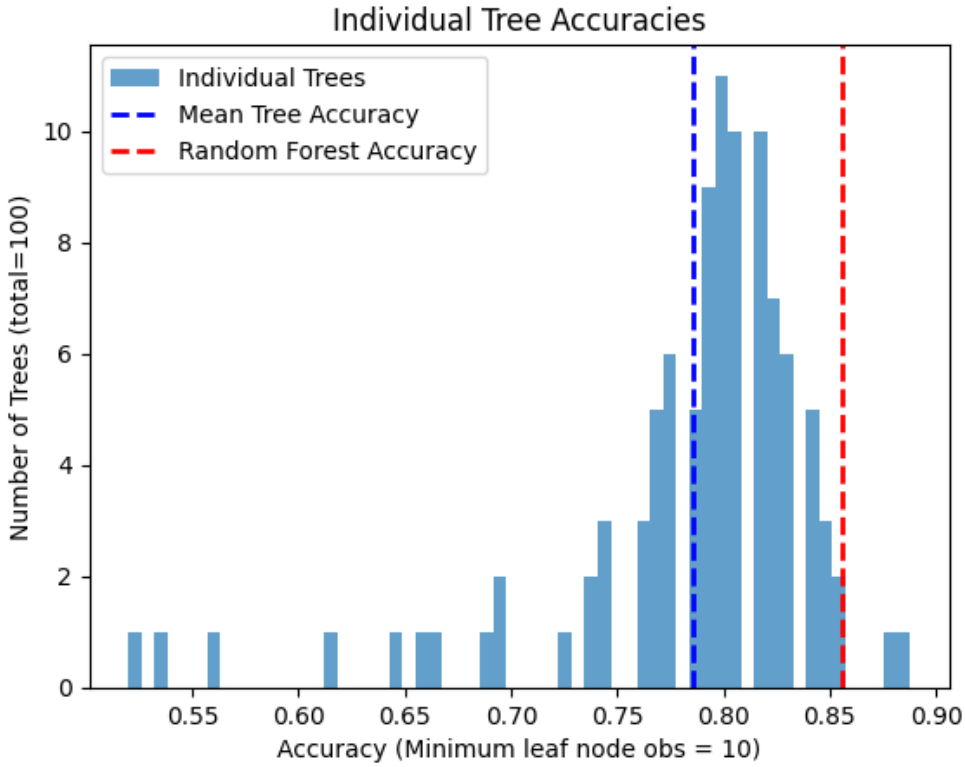
This model will then be used in our *predictRF(model, X)* function, which will use the test dataset (dictionary) in order to obtain a class for each sample. For this, we use the trees and features

contained in our dictionary by accessing the keys and by also using the .predict function contained in sklearn. After that we create an array where we use np.apply_along_axis(lambda x: *np.bincount(x).argmax(), axis=0, arr=predict).* Since each tree has made different predictions, we will assign the final class based on the majority rule, meaning that if 50 trees classified sample 1 as Y = 0 and 150 as Y = 1, then it will classify it as Y = 1. This is the power of random forests. Since in the code we will have lists on top of lists containing classes for each sample from each tree, we want to find the one that was assigned by the most trees. Bincount will create a list with length equal to the classes, and assign the number it was found (like we did in decision trees for example 1(0) 2(1) etc), but will look like [1,2]. Argmax will return the highest of those occurrences. This function is applied on the columns (axis = 0).

For part B, we first split using *sklearn train_test_split* to split into a 70% train and 30% test set. After we train and predict using our models, we want to calculate the accuracy, which is done as #Correct_Predictions / #Total_samples.
We create a function to calculate it *get_accuracies.* This function is applied both on each individual tree, and appending it, and then by using the whole Random Forest. (More details within the code).

Overall what we expect from the plots is that changing min_samples_leaf from 1 to 10 is expected to reduce model complexity, potentially reduce overfitting, and result in a more consistent and possibly more accurate Random Forest model on unseen data. The trade-off is between capturing the detailed variance of the training data versus ensuring the model's robustness and generalizability. While running this many times did provide different results, we notice indeed a more consistent mean accuracy for the combined Random Forest.


Individual Tree Accuracies
Individual Trees
Mean Tree Accuracy
Random Forest Accuracy
Number of Trees (total=100)
0
2
4
6
8
10
12
14
0.55
0.60
0.65
0.70
0.75
0.80
0.85
Accuracy (Minimum leaf node obs = 1)


Individual Tree Accuracies
Individual Trees
Mean Tree Accuracy
Random Forest Accuracy
Number of Trees (total=100)
0
2
4
6
8
10
0.55
0.60
0.65
0.70
0.75
0.80
0.85
0.90
Accuracy (Minimum leaf node obs = 10)