# Ensembles:

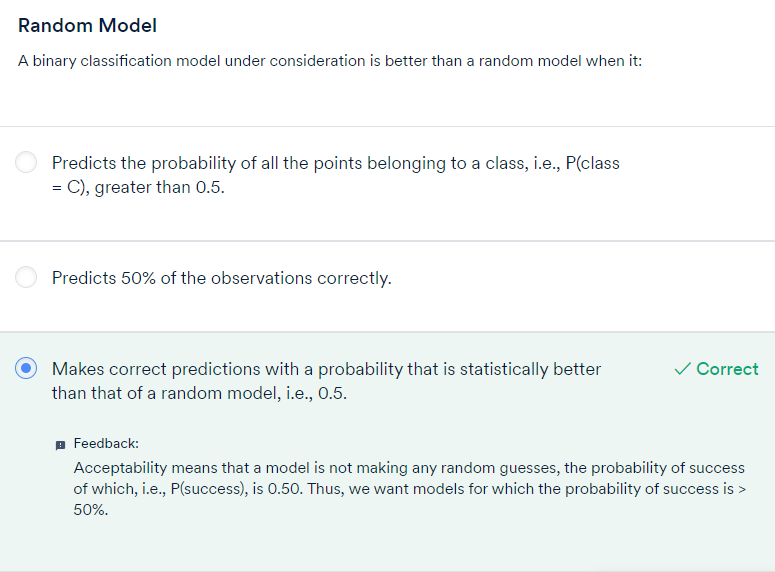
An ensemble refers to a group of things viewed as a whole rather than individually. In an ensemble, a **collection of models** is used to make predictions, rather than individual models. Arguably, the most popular in the family of ensemble models is the random forest, which is an ensemble made by the **combination of a large number of decision trees**.

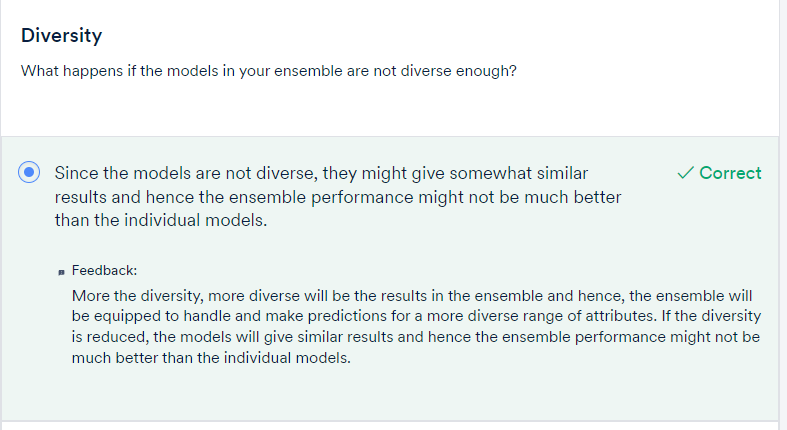
**Diversity** ensures that the models serve **complementary** purposes, which means that the individual models make predictions **independent of each other**.

**Acceptability** implies that each model is at least **better than a random model**. This is a pretty lenient criterion for each model to be accepted into the ensemble, i.e., it has to be at least better than a random guesser.

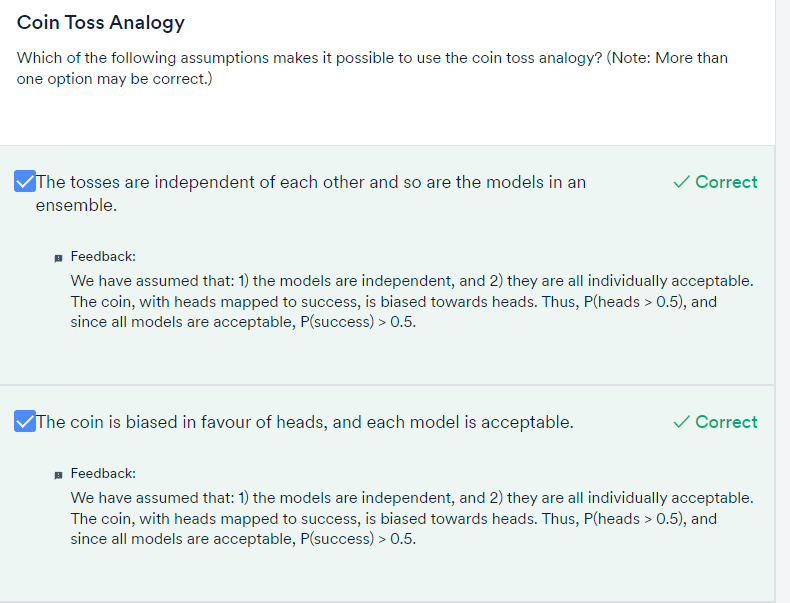
There are a number of ways in which you can bring diversity among your models you plan to include in your ensemble.

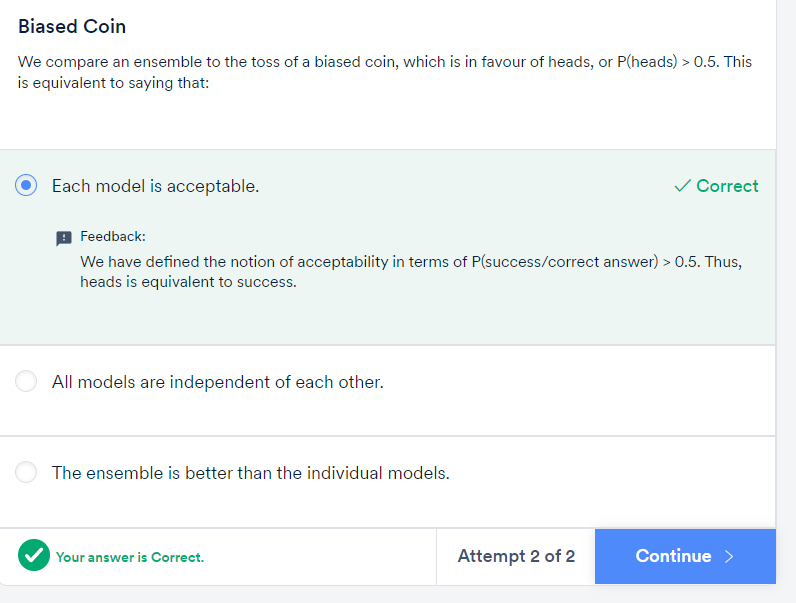
1. Use different subsets of training data
2. Use different training hyperparameters
3. Use different types of classifiers
4. Use different features

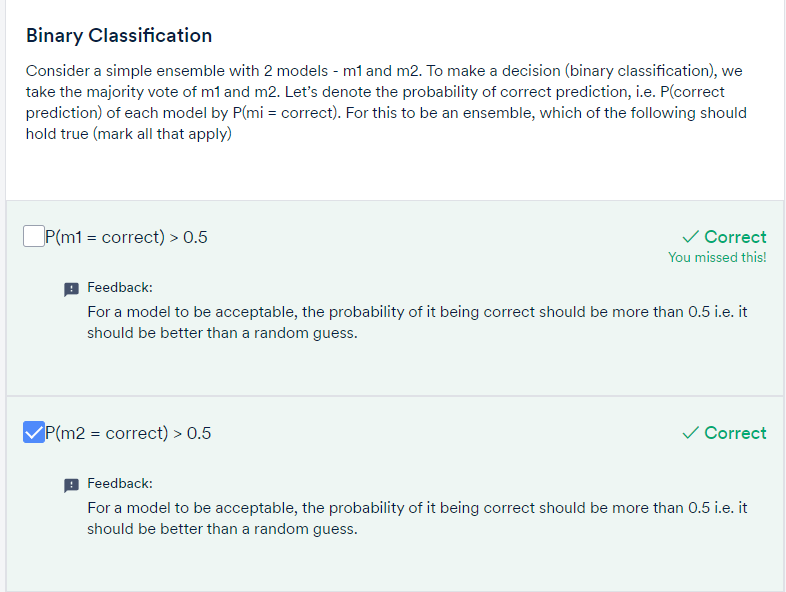


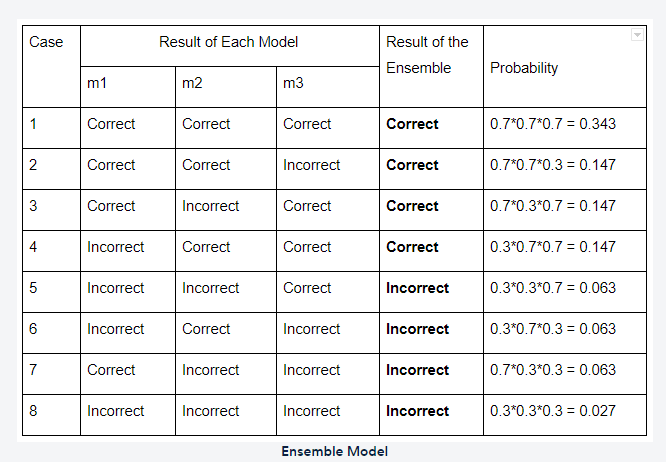


# Comprehension- Ensembles:









In this table, there are four cases each where the decision of the final model (ensemble) is either correct or incorrect. Let’s assume that the probability of the ensemble being correct is p, and the probability of the ensemble being incorrect is q.

For the data in the table, p and q can be calculated as follows:

* **p = 0.343 + 0.147 + 0.147 + 0.147 = 0.784**
* **q = 0.027 + 0.063 + 0.063 + 0.063 = 0.216 = 1 - p**

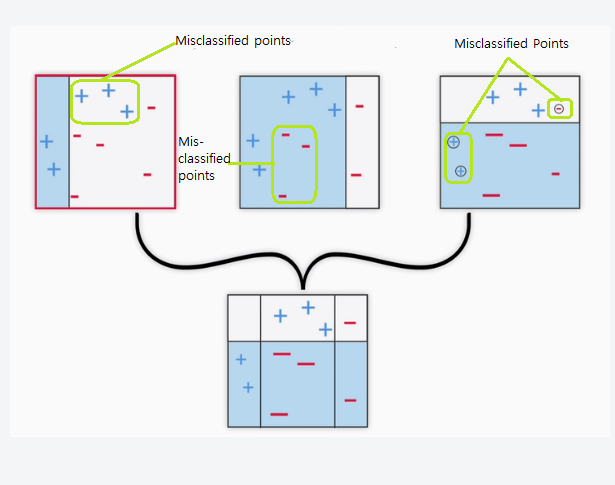
Notice how the ensemble has a higher probability of being correct and a lower probability of being incorrect than any of the individual models (0.78 > 0.70 and 0.216 < 0.30). In this way, you can also calculate the probabilities of the ensemble being correct and incorrect with 4, 5, 100, 1000, and even a million individual models. The difference in probabilities will increase with an increasing number of models, thus **improving the overall performance of the ensemble.**

# Popular Ensembles:

Voting combines the output of different algorithms by taking a vote. In the case of a classification model, if the majority of the classifiers predict a particular class, then the output of the model would be the same class. In the case of a regression problem, the model output is the average of all the predictions made by the individual models. In this way, every classifier/regressor has an equal say in the final prediction.

Another approach to carry out manual ensembling is to pass the outputs of the individual models to a level-2 classifier/regressor as derived meta features, which will decide what weights should be given to each of the model outputs in the final prediction. In this way, the outputs of the individual models are combined with different weightages in the final prediction. This is the high-level approach behind stacking and blending.

Boosting is one of the most popular approaches to ensembling. It can be used with any technique and combines the weak learners into strong learners by creating sequential models such that the final model has higher accuracy than the individual models. You saw the example shown below to see intuitively how adaptive boosting works.



# Bagging: Bootstrap Aggregation:

Bagging creates different training subsets from the sample training data with replacement, and an algorithm with the same set of hyperparameters is built on these different subsets of data. In this way, the same algorithm with a similar set of hyperparameters is exposed to different parts of data, resulting in a slight difference between the individual models. The predictions of these individual models are combined by taking the average of all the values for regression or a majority vote for a classification problem.

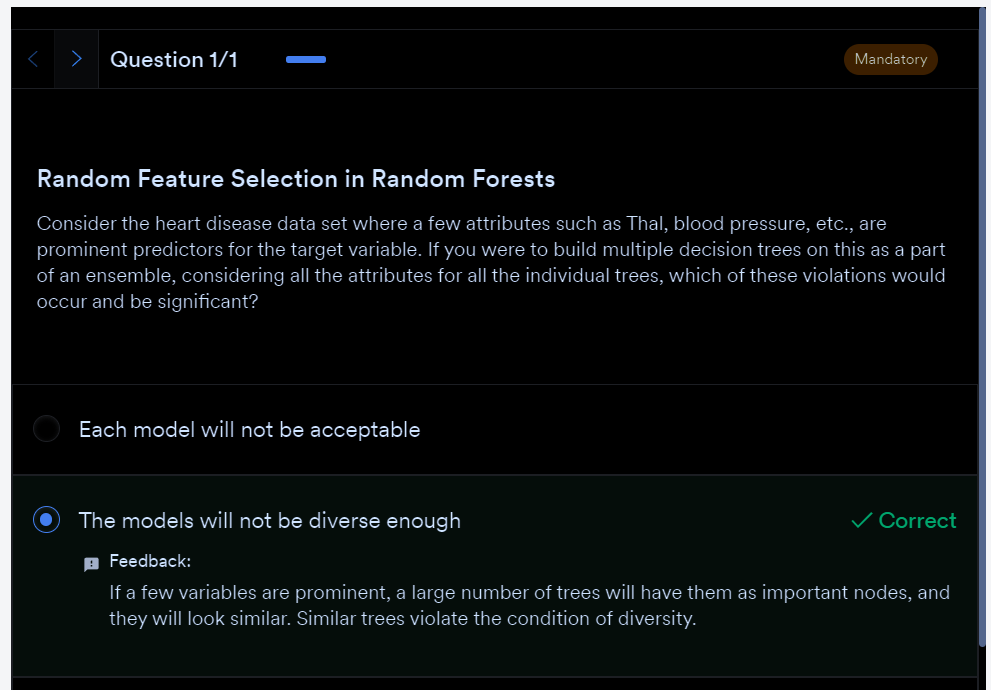
Bagging works well with high variance algorithms and is easy to parallelise. By high variance, we mean algorithms which change a lot with slight changes in the data as a result of which these algorithms very easily overfit if not controlled. If you recall, decision trees are very prone to overfitting if we don't tune the hyperparameters well. Hence, bagging works very well for high-variance models like decision trees.

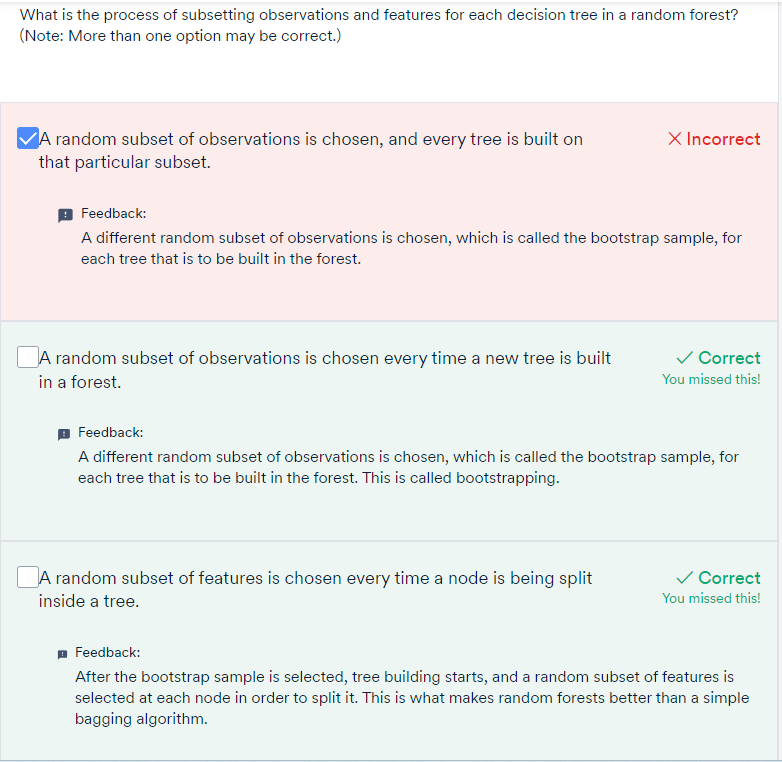
However,  it has got some disadvantages as well. In this approach, you cannot really see the individual trees one by one and figure out what is going on behind the ensemble as it is a combination of n number of trees working together. This leads to a loss of interpretability. Also, it does not work well when any of the features dominate because of which all the trees look similar and hence the property of diversity in ensembles is lost. Sometimes bagging can be computationally expensive and is applied depending on the case.

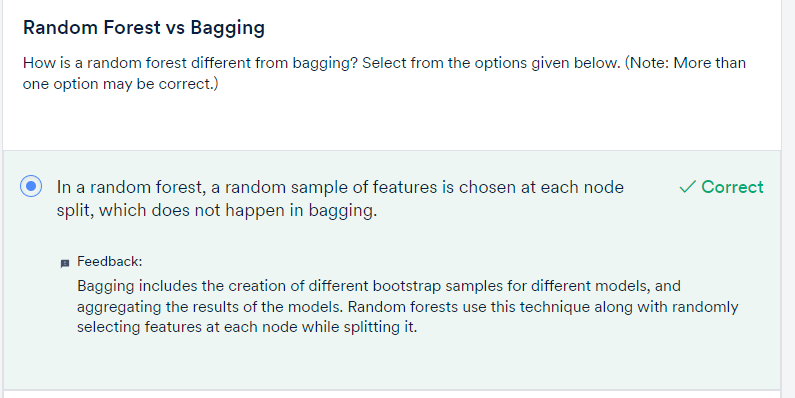
# Introduction to Random Forests :

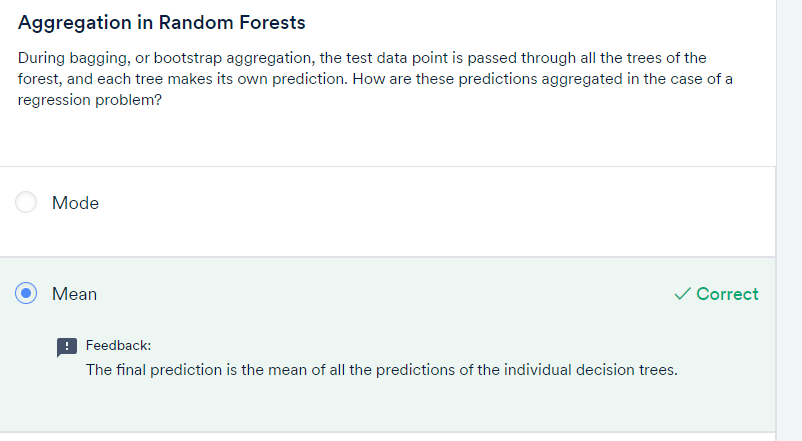
**Bagging** chooses random samples of observations from a data set. Each of these samples is then used to train each tree in the forest. However, keep in mind that bagging is only a **sampling technique** and is **not specific to random forests**.

In the bagging type of ensembles, random forests are by far the most successful. They are essentially ensembles of a **number of decision trees**. You can create a large number of models (say, 100 decision trees), each one on a **different bootstrap sample** from the training set. To get the result, you can **aggregate** the decisions taken by all the trees in the ensemble. Let's watch the following video to understand the same in detail.









## **Advantages of Blackbox Models Over Tree and Linear Models**

**Diversity**: Diversity arises because each tree is created with a subset of the attributes/features/variables, i.e., not all the attributes are considered while making each tree; the choice of the attributes is random. This ensures that the trees are independent of each other.

**Stability**: Stability arises because the answers given by a large number of trees average out. A random forest has a lower model variance than an ordinary individual tree.

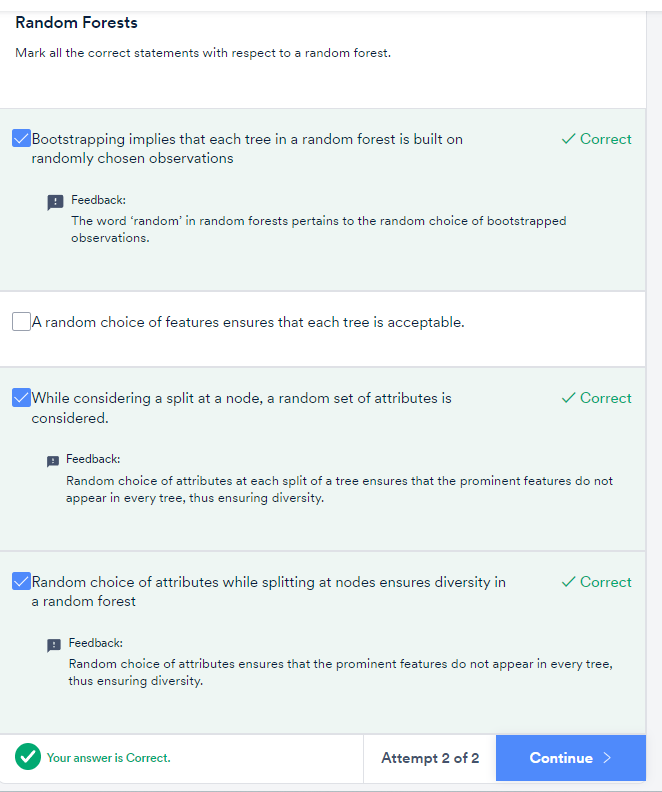
**Immunity to the curse of dimensionality**: Since each tree does not consider all the features, the feature space (the number of features that a model has to consider) reduces. This makes an algorithm immune to the curse of dimensionality. Also, a large feature space causes computational and complexity issues.

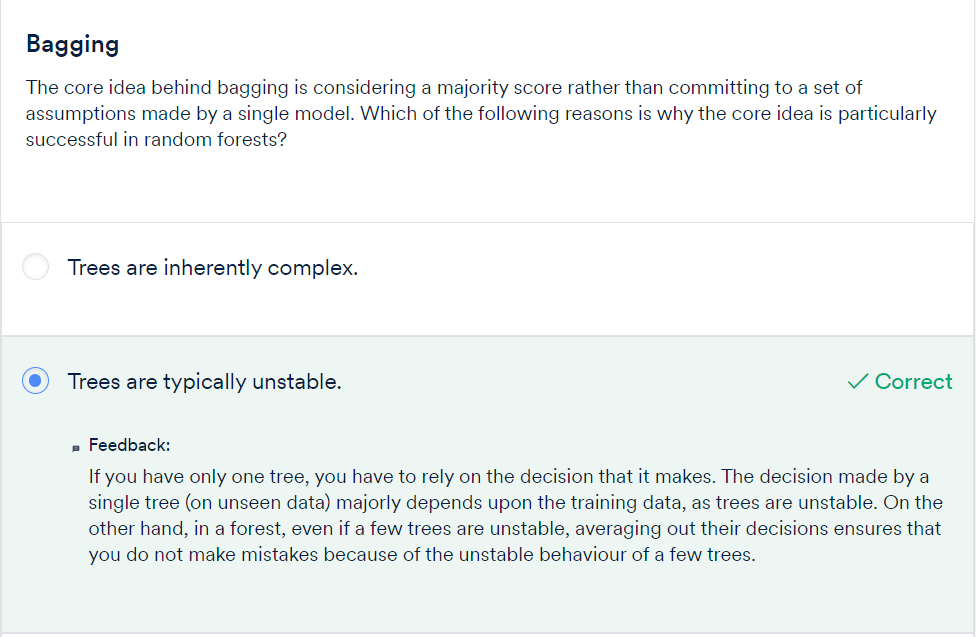
**Parallelization**: You need a number of trees to make a forest. Since two trees are independently built on different data and attributes, they can be built separately. This implies that you can make full use of your multi-core CPU to build random forests. Suppose there are 4 cores and 100 trees to be built; each core can build 25 trees to make a forest.

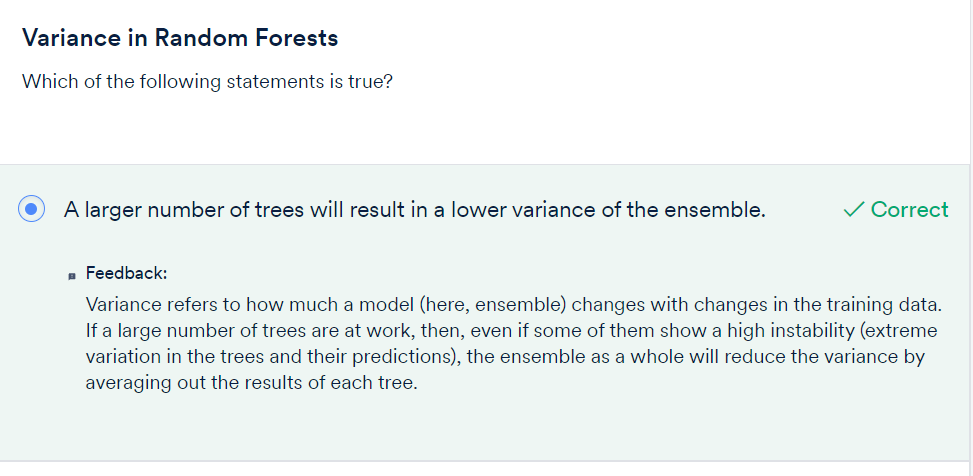
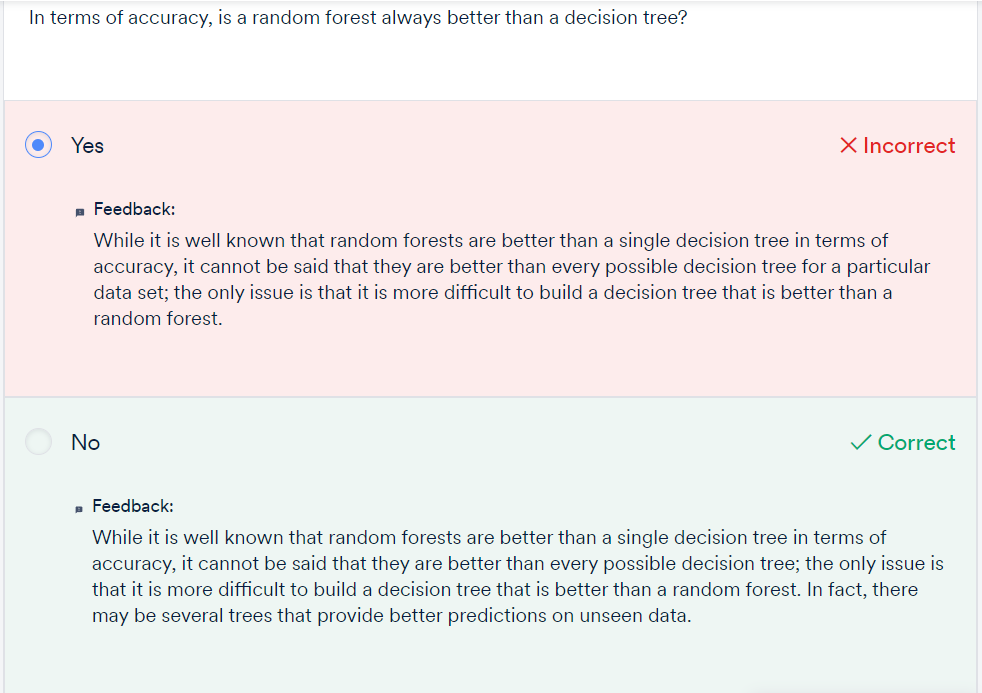
**Testing/training data and the OOB (out-of-bag) error**: You should always avoid violating the fundamental tenet of learning: 'Not testing a model on what it has been trained on’. While building individual trees, you can choose a random subset of the observations to train them. If you have 10,000 observations, each tree may only be built from 7,000 (70%) randomly chosen observations. OOB is the mean prediction error on each training sample xᵢ, using only the trees that do not have xᵢ in their bootstrap sample used for building the model. This is very similar to a cross-validation (CV) error. In a CV error, you can measure the performance on the subset of data that the model has not seen before.

In fact, it has been proven that using an OOB estimate is as accurate as using a test data set of a size equal to the training set.

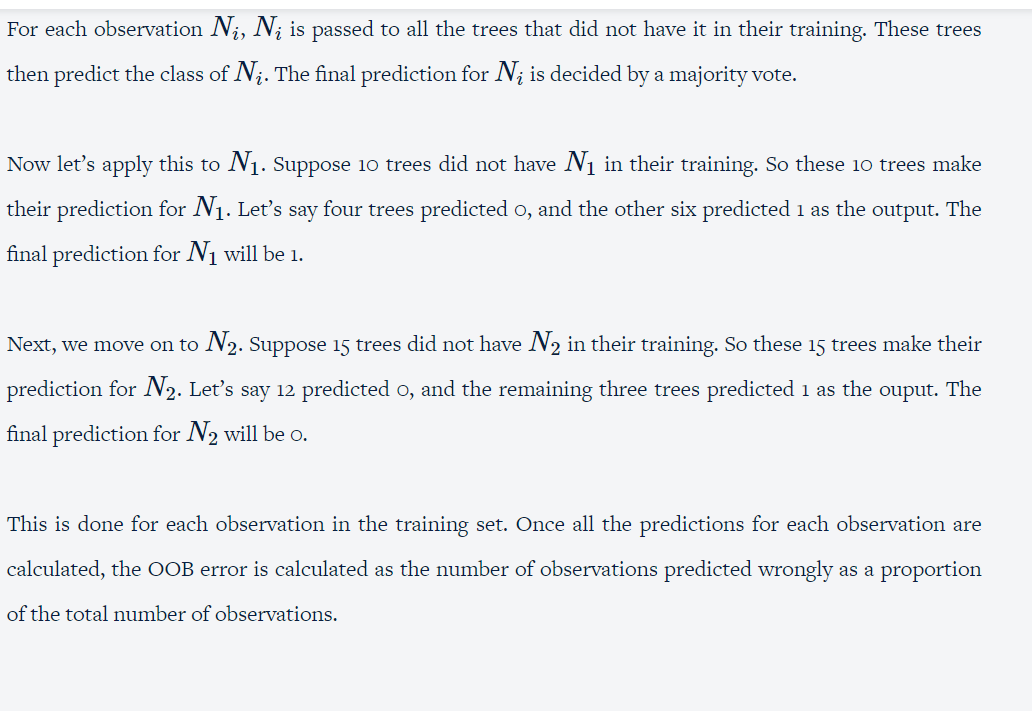
Thus, the OOB error omits the need for set-aside test data (though you can still work with test data like you have been doing, at the cost of eating into the training data).

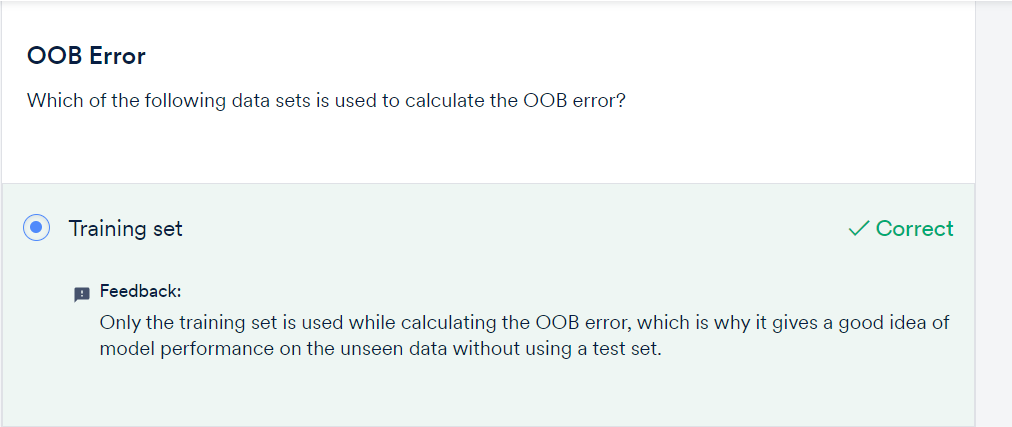


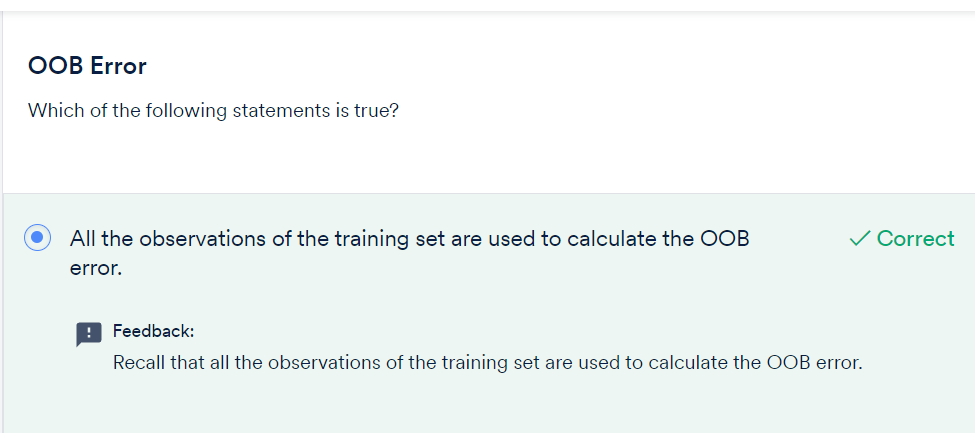




# Comprehension - OOB (Out-of-Bag) Error:







# Feature Importance in Random Forests :

The importance of features in random forests, sometimes called ‘**Gini importance**’ or ‘**mean decrease impurity**’, is defined as the **total decrease in node impurity** (it is weighted by the probability of reaching that node (which is approximated by the proportion of samples reaching that node)) **averaged**over all the trees of the ensemble.

To summarise, you learnt how to build a random forest in sklearn. Apart from the hyperparameters that you have in a decision tree, there are two more hyperparameters in random forests: **max\_features** and **n\_estimators**. The effects of both the hyperparameters are briefly summarised below.

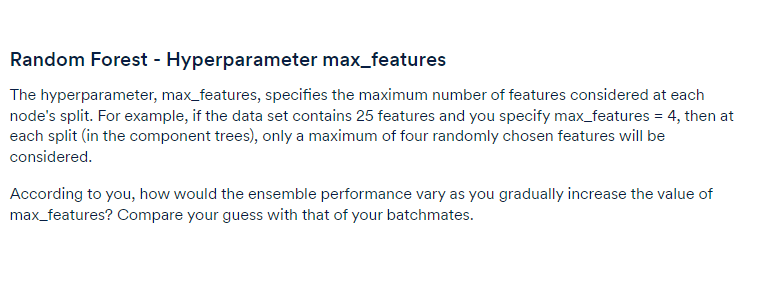
**The effect of max\_features**

You learnt that there is an optimal value of max\_features, i.e, at very low values, the component trees are too simple to learn about anything useful, while at extremely high values, the component trees become similar to each other (and violate the 'diversity' criterion).

**The effect of n\_estimators**

When you observe the plot of n\_estimators and training and test accuracies, you will see that as you increase the value of n\_estimators, the accuracies of both the training and test sets gradually increase. More importantly, the model does not overfit even when its complexity is increasing. This is an important benefit of random forests: You can increase the number of trees as much as you like without worrying about overfitting (only if your computational resources allow).

Also, as you saw, since there were a lot of models to fit, the time taken was quite high. If you want to gain a better understanding of the time taken to build random forests, you can go through [this](https://learn.upgrad.com/course/3615/segment/33940/201104/618672/3149639) optional segment.





# Random Forest Regression in Python :

