Step 1: Create a “bootstrapped” dataset.

Original Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

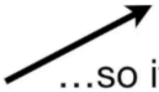
Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease

To create a bootstrapped dataset that is the same size as the original, we just randomly select samples from the original dataset.

Original Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes



…so it's the first sample in our bootstrapped dataset.

Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes

Step 2: Create a decision tree using the bootstrapped dataset, but only use a random subset of variables (or columns) at each step.

In this example, we will only consider 2 variables (columns) at each step.

Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

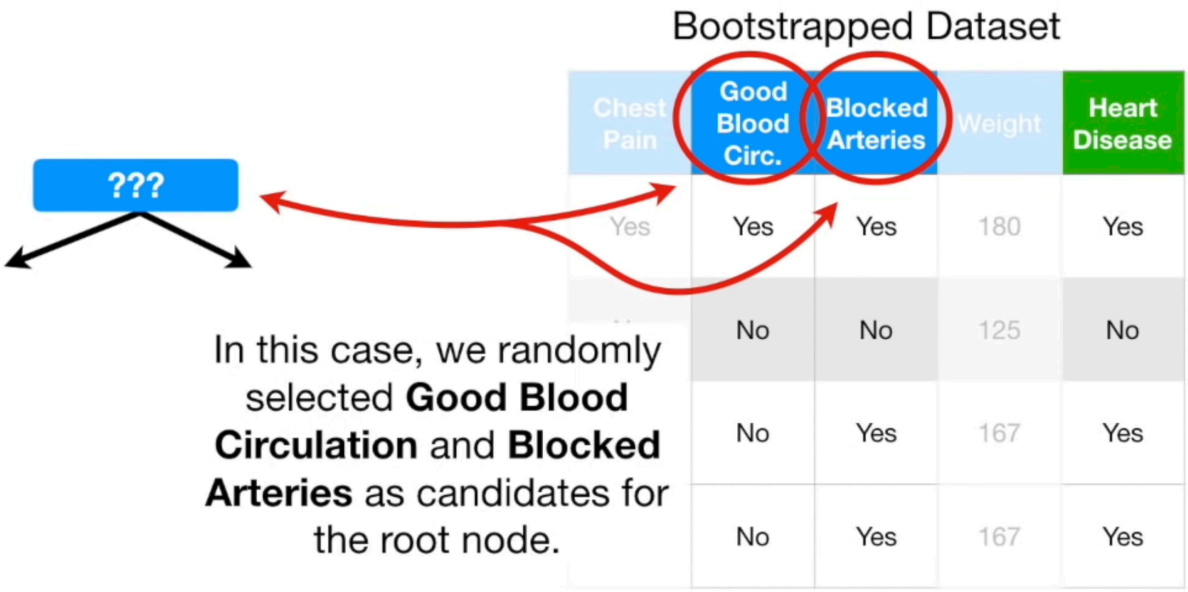
Step 2: Create a decision tree using the bootstrapped dataset, but only use a random subset of variables (or columns) at each step.

In this example, we will only consider 2 variables (columns) at each step.

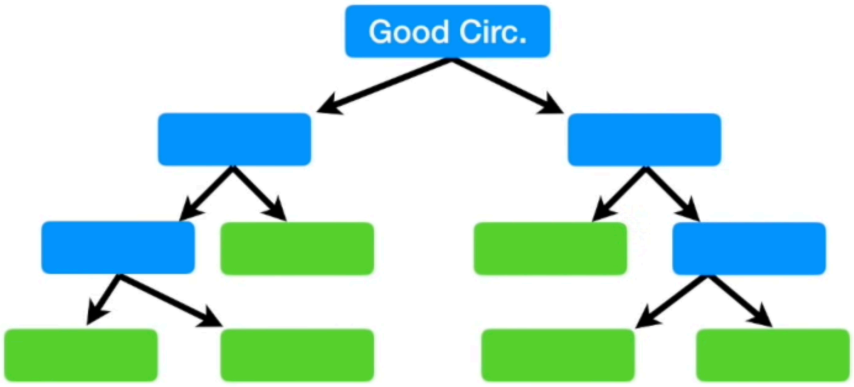
NOTE: We'll talk more about how to determine the optimal number of variables to consider later…

Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

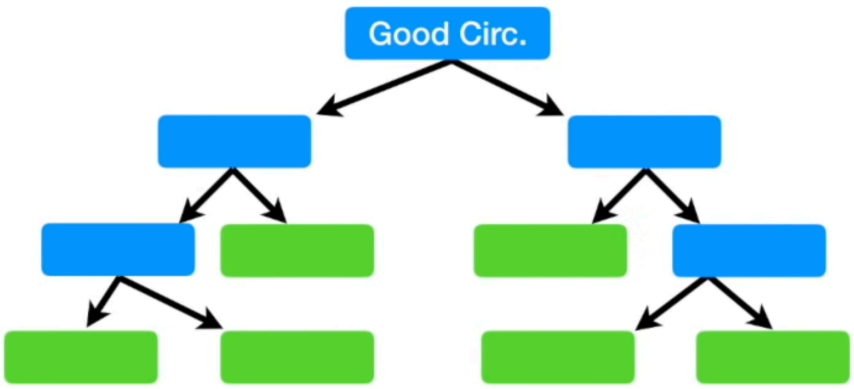

Bootstrapped Dataset
???
Chest Pain
Good Blood Circ.
Blocked Arteries
Weight
Heart Disease
Yes
Yes
Yes
180
Yes
No
No
125
No
No
Yes
167
Yes
No
Yes
167
Yes
In this case, we randomly selected **Good Blood Circulation** and **Blocked Arteries** as candidates for the root node.

Double Bam!!!



Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes



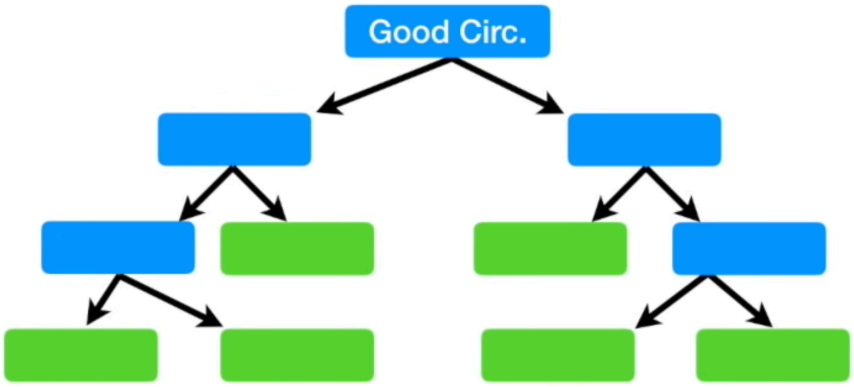
Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

We built a tree…

1) Using a bootstrapped dataset

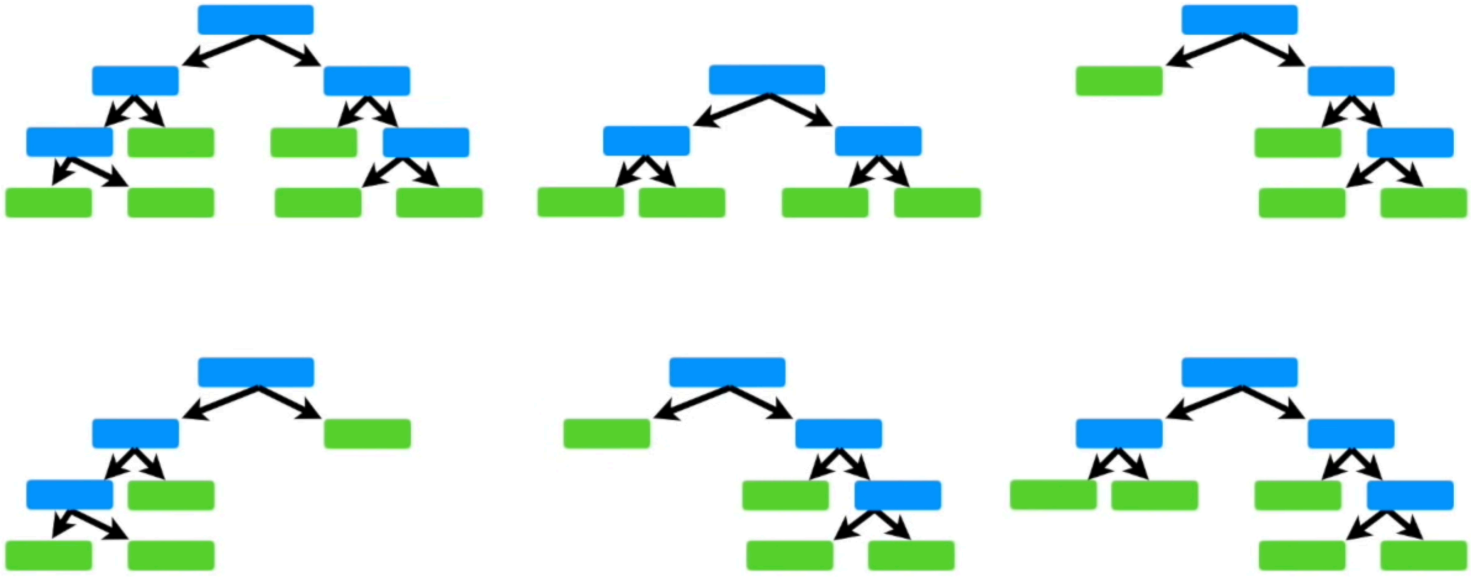
2) Only considering a random a subset of variables at each step.



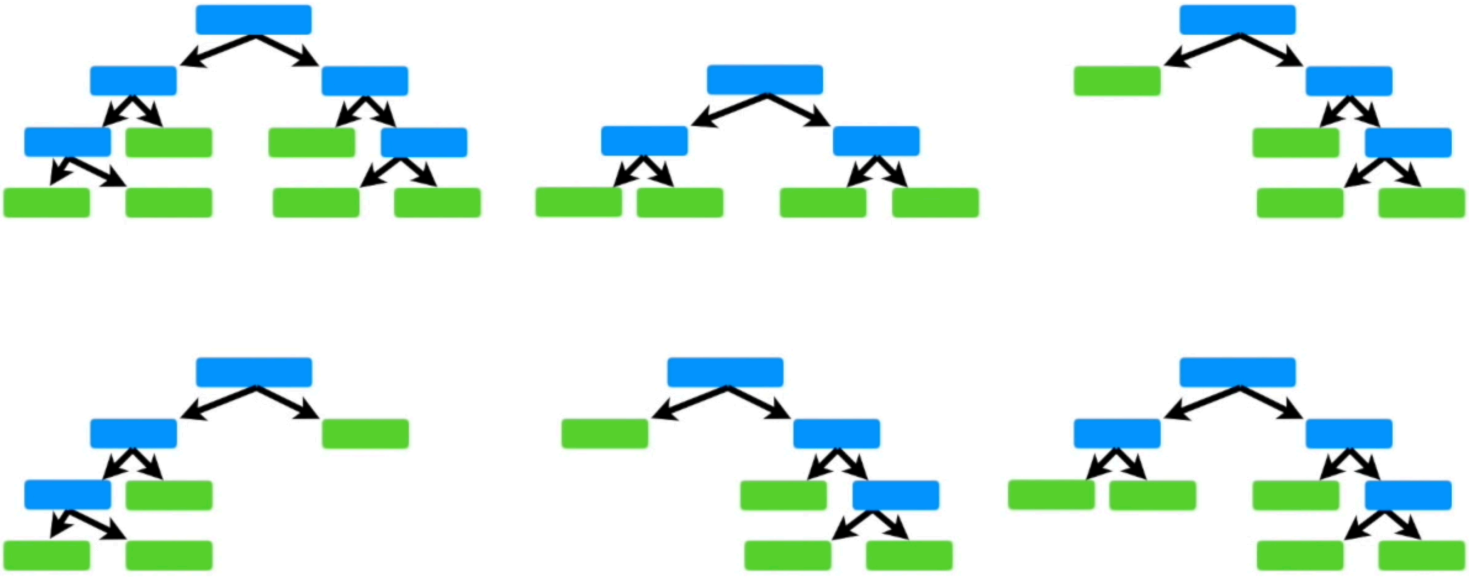
Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

Now go back to Step 1 and repeat: Make a new bootstrapped dataset and build a tree considering a subset of variables at each step.

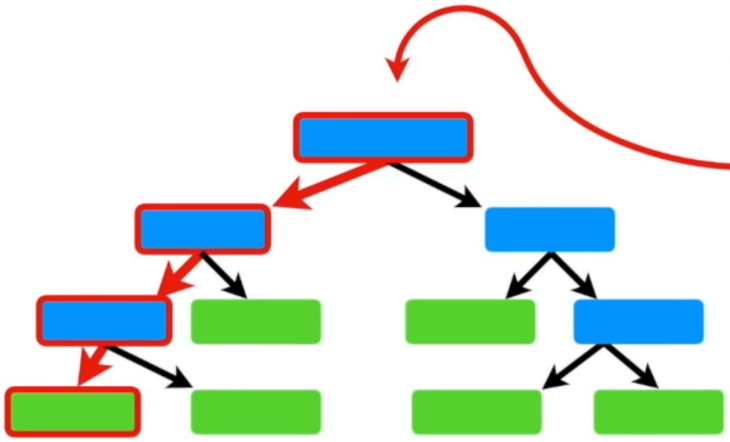


Using a bootstrapped sample and considering only a subset of the variables at each step results in a wide variety of trees.



Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	No	No	168	

…we've got all the measurements…



Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	No	No	168	

The first tree says "Yes"…

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	No	No	168	

After running the data down all of the trees in the random forest, we see which option received more votes.

Heart Disease	
Yes	No
5	1

Terminology Alert!!!

Bootstrapping the data plus using the **agg**regate to make a decision is called "**Bagging**"

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	No	No	168	**YES**

Heart Disease	
Yes	No
5	1

We allowed duplicate entries in the bootstrapped dataset…

Original Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

As a result, this entry was not included in the bootstrapped dataset.

Original Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

Typically, about 1/3 of the original data does not end up in the bootstrapped dataset.

Original Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

Original Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

Here is the entry that didn't end up in the bootstrapped dataset..

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	No	210	No

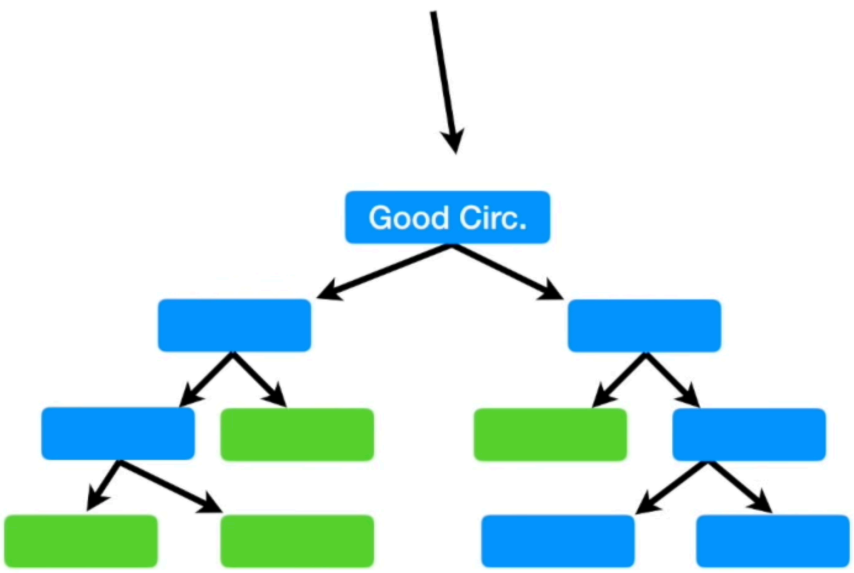
Original Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

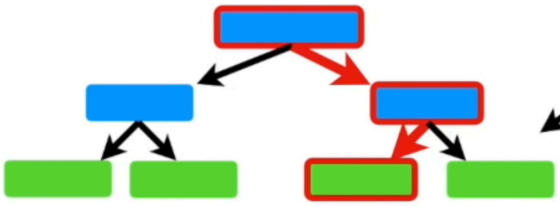
This is called the
"**Out-Of-Bag Dataset**"

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	No	210	No

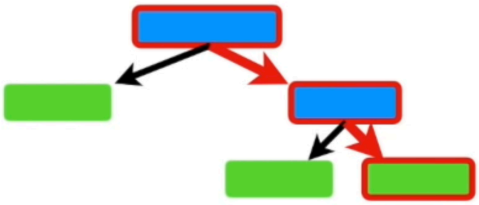
Since the Out-Of-Bag data was not used to create this tree…



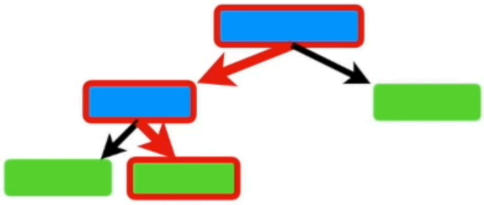
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	No	210	No



This tree incorrectly labeled the Out-of-Bag sample "**Yes**".



Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	No	210	No



Classification of the Out-Of-Bag sample	
Yes	No
1	3

Classification of the Out-Of-Bag sample	
Yes	No
4	0

Classification of the Out-Of-Bag sample	
Yes	No
3	1

etc… etc… etc…

Ultimately, we can measure how accurate our random forest is by the proportion of Out-Of-Bag samples that were correctly classified by the Random Forest.

Classification of the Out-Of-Bag sample	
Yes	No
1	3

Classification of the Out-Of-Bag sample	
Yes	No
4	0

Classification of the Out-Of-Bag sample	
Yes	No
3	1

etc… etc… etc…

Ultimately, we can measure how accurate our random forest is by the proportion of Out-Of-Bag samples that were correctly classified by the Random Forest.

The proportion of Out-Of-Bag samples that were *incorrectly* classified is the "**Out-Of-Bag Error**"

OK, we now know how to:

1) Build a Random Forest

2) Use a Random Forest

3) Estimate the accuracy of a Random Forest.

Now we can compare the Out-Of-Bag error for a random forest built using only 2 variables per step…

Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

…to random forest built using 3 variables per step…

…and we test a bunch of different settings and choose the most accurate random forest.

Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

In other words…

…change the number of variables used per step…

1) Build a Random Forest

2) Estimate the accuracy of a Random Forest.

In other words…

…change the number of variables used per step…

1) Build a Random Forest

2) Estimate the accuracy of a Random Forest.

Do this for a bunch of times and then choose the one that is most accurate.

In other words…

…change the number of variables used per step…

1) Build a Random Forest

2) Estimate the accuracy of a Random Forest.

Typically, we start by using the square of the number of variables and then try a a few settings above and below that value.

StatQuest:
Random Forests in R

```
> set.seed(42)
>
> data.imputed <- rfImpute(hd ~ ., data = data, iter=6)
ntree      OOB      1      2
  300:  17.49% 14.02% 21.58%
ntree      OOB      1      2
  300:  17.16% 13.41% 21.58%
ntree      OOB      1      2
  300:  17.49% 14.02% 21.58%
ntree      OOB      1      2
  300:  17.16% 13.41% 21.58%
ntree      OOB      1      2
  300:  17.16% 13.41% 21.58%
ntree      OOB      1      2
  300:  17.16% 13.41% 21.58%
> model <- randomForest(hd ~ ., data=data.imputed, proximity=TRUE)
```

Just like when we imputed values for the **NAs**, we want to predict **hd** (aka **h**eart **d**isease) using all of the other columns in the dataset.

```
> set.seed(42)
>
> data.imputed <- rfImpute(hd ~ ., data = data, iter=6)
ntree      OOB      1      2
  300:  17.49% 14.02% 21.58%
ntree      OOB      1      2
  300:  17.16% 13.41% 21.58%
ntree      OOB      1      2
  300:  17.49% 14.02% 21.58%
ntree      OOB      1      2
  300:  17.16% 13.41% 21.58%
ntree      OOB      1      2
  300:  17.16% 13.41% 21.58%
ntree      OOB      1      2
  300:  17.16% 13.41% 21.58%
> model <- randomForest(hd ~ ., data=data.imputed, proximity=TRUE)
```

We also want **randomForest()** to return the proximity matrix. We'll use this to cluster the samples at the end of the StatQuest.

```
> model

Call:
 randomForest(formula = hd ~ ., data = data.imputed, proximity = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 16.5%
Confusion matrix:
          Healthy Unhealthy class.error
Healthy       141        23   0.1402439
Unhealthy      27       112   0.1942446
```

Next we see that the random forest was built to classify samples.

If we had used the random forest to predict weight or height, it would say "**regression**".

```
> model

Call:
 randomForest(formula = hd ~ ., data = data.imputed, proximity = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 16.5%
Confusion matrix:
           Healthy Unhealthy class.error
Healthy        141        23   0.1402439
Unhealthy       27       112   0.1942446
```

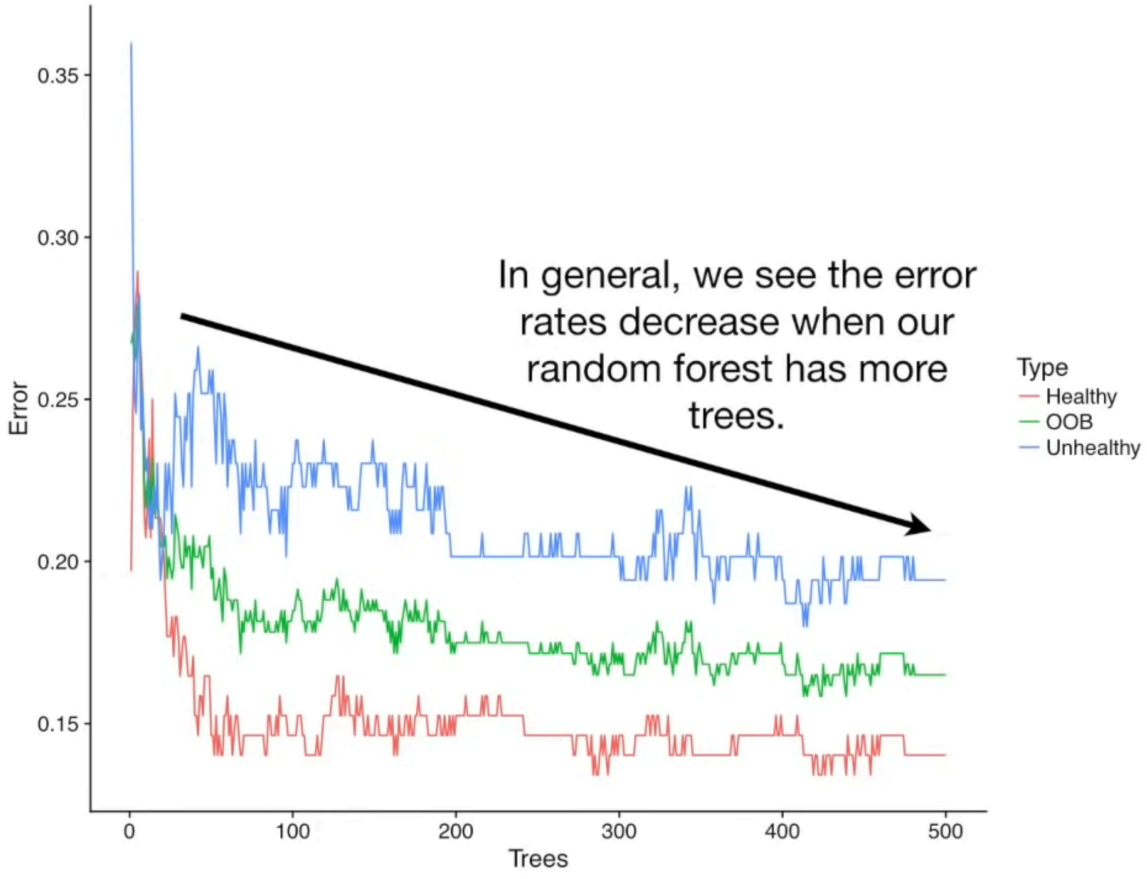
Here's the **O**ut-**o**f-**B**ag (**OOB**) error estimate. This means that 83.5% of the OOB samples were correctly classified by the random forest.

```
> model

Call:
 randomForest(formula = hd ~ ., data = data.imputed, proximity = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 16.5%
Confusion matrix:
          Healthy Unhealthy class.error
Healthy       141        23   0.1402439
Unhealthy      27       112   0.1942446
```

There were 27 unhealthy patients that were incorrectly classified “Healthy”…

```
> oob.error.data <- data.frame(
+   Trees=rep(1:nrow(model$err.rate), times=3),
+   Type=rep(c("OOB", "Healthy", "Unhealthy"), each=nrow(model$err.rate)),
+   Error=c(model$err.rate[,"OOB"],
+     model$err.rate[,"Healthy"],
+     model$err.rate[,"Unhealthy"]))
>
> ggplot(data=oob.error.data, aes(x=Trees, y=Error)) +
+   geom_line(aes(color=Type))
```

To see if 500 trees is enough for optimal classification, we can plot the error rates.


0.35
0.30
0.25
0.20
0.15
Error
0
100
200
300
400
500
Trees
In general, we see the error rates decrease when our random forest has more trees.
Type
Healthy
OOB
Unhealthy

```
> model <- randomForest(hd ~ ., data=data.imputed, ntree=1000, proximity=TRUE)
> model

Call:
 randomForest(formula = hd ~ ., data = data.imputed, ntree = 1000,      proximity
= TRUE)
               Type of random forest: classification
                     Number of trees: 1000
No. of variables tried at each split: 3

        OOB estimate of  error rate: 16.5%
Confusion matrix:
          Healthy Unhealthy class.error
Healthy       142        22   0.1341463
Unhealthy      28       111   0.2014388
```

The OOB error rate is the same as before…