COSC 3337 : Data Science I

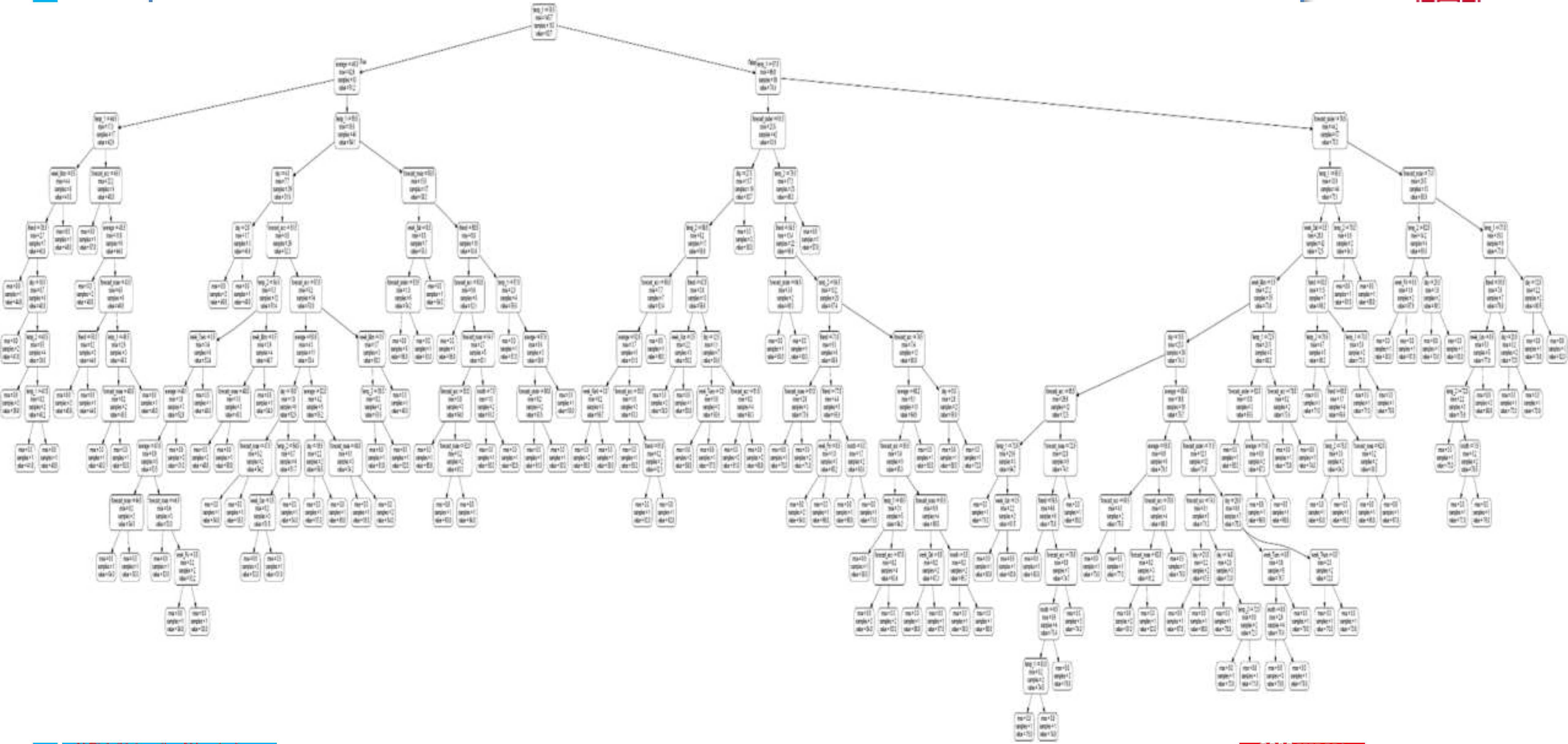
N. Rizk

College of Natural and Applied Sciences

Department of Computer Science

University of Houston

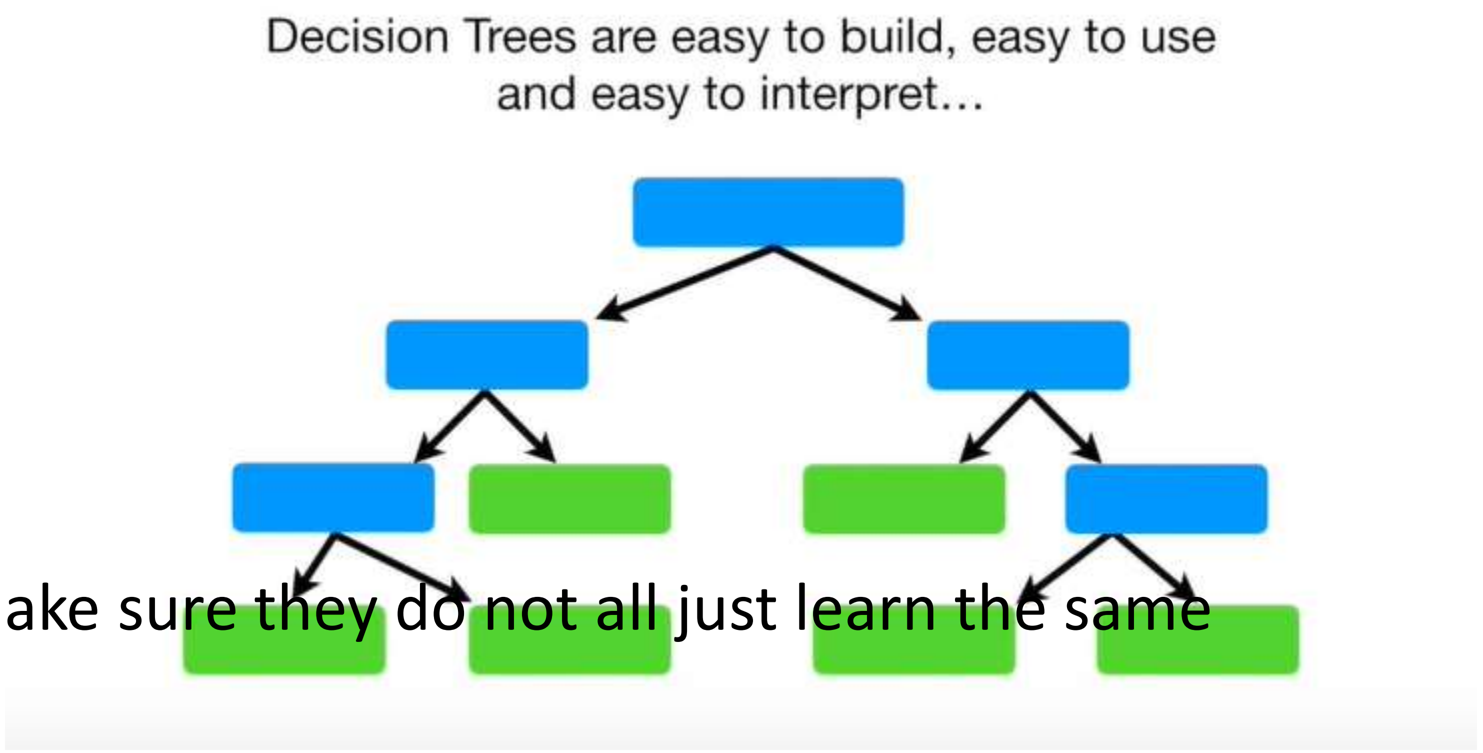
Random Forests



Random_Forest

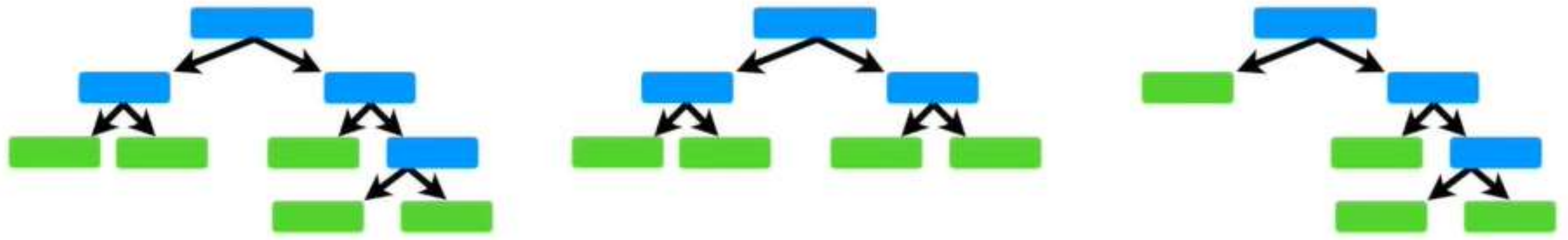
A single decision tree does not perform well

- Inaccurate and not flexible in classifying new samples. But, it is super fast
- What if we learn multiple trees?


Decision Trees are easy to build, easy to use and easy to interpret…

- We need to make sure they do not all just learn the same

The good news is that **Random Forests** combine the simplicity of decision trees with flexibility resulting in a vast improvement in accuracy.



How to build and evaluate a random forest?

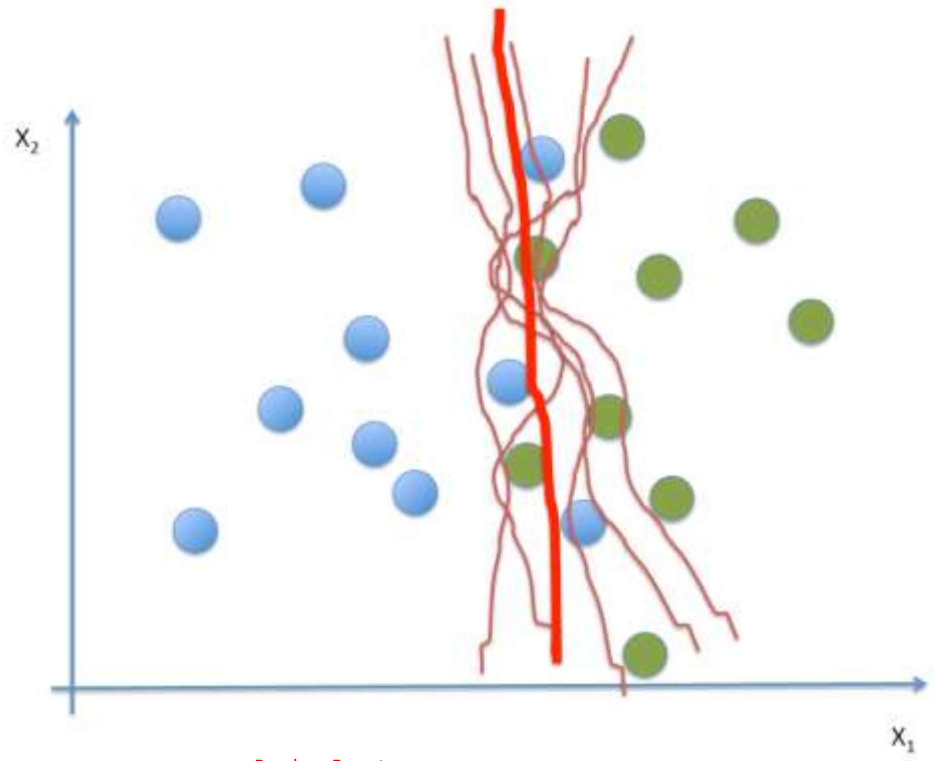
1- create a Boostraped data set from the original data set

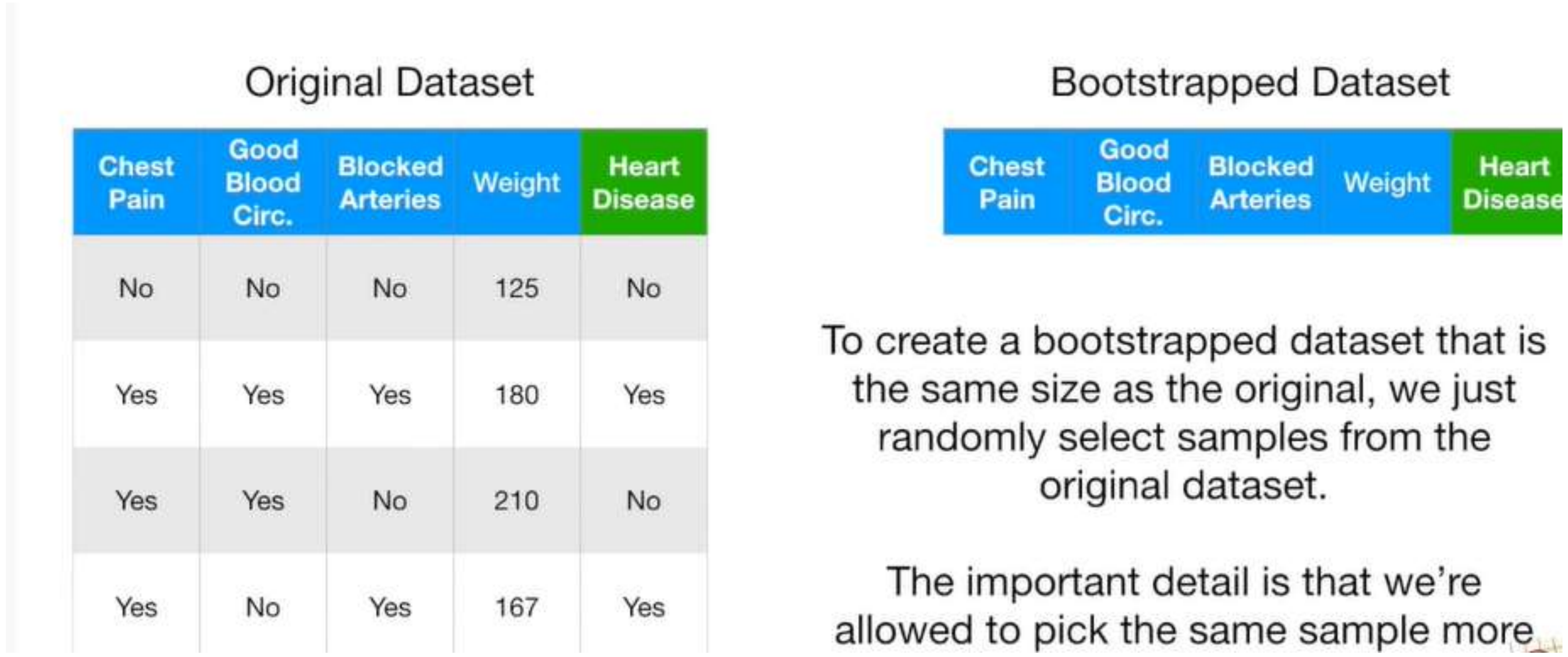
2- build trees

3-run data along each tree

Bootstrap

- Construct B (hundreds) of trees (no pruning)
- Learn a classifier for each bootstrap sample and average them
- Very effective





Original Dataset

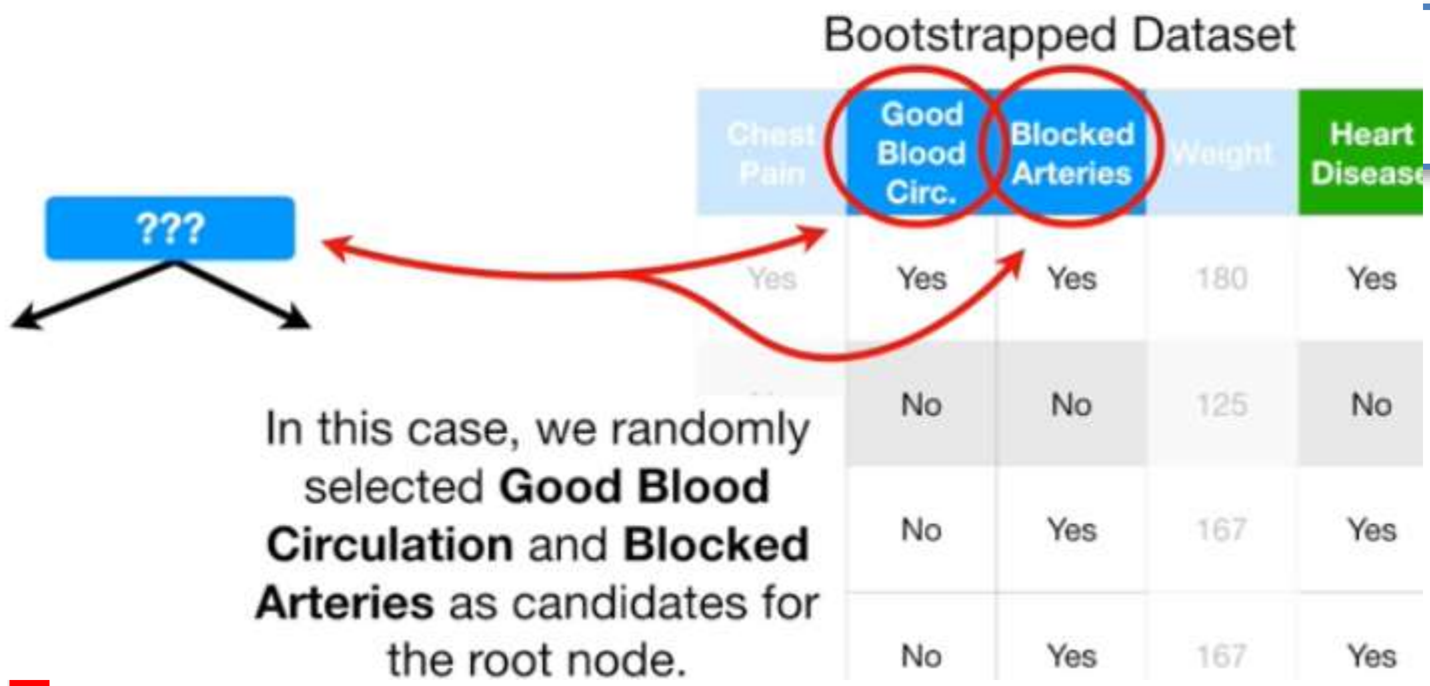
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

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.

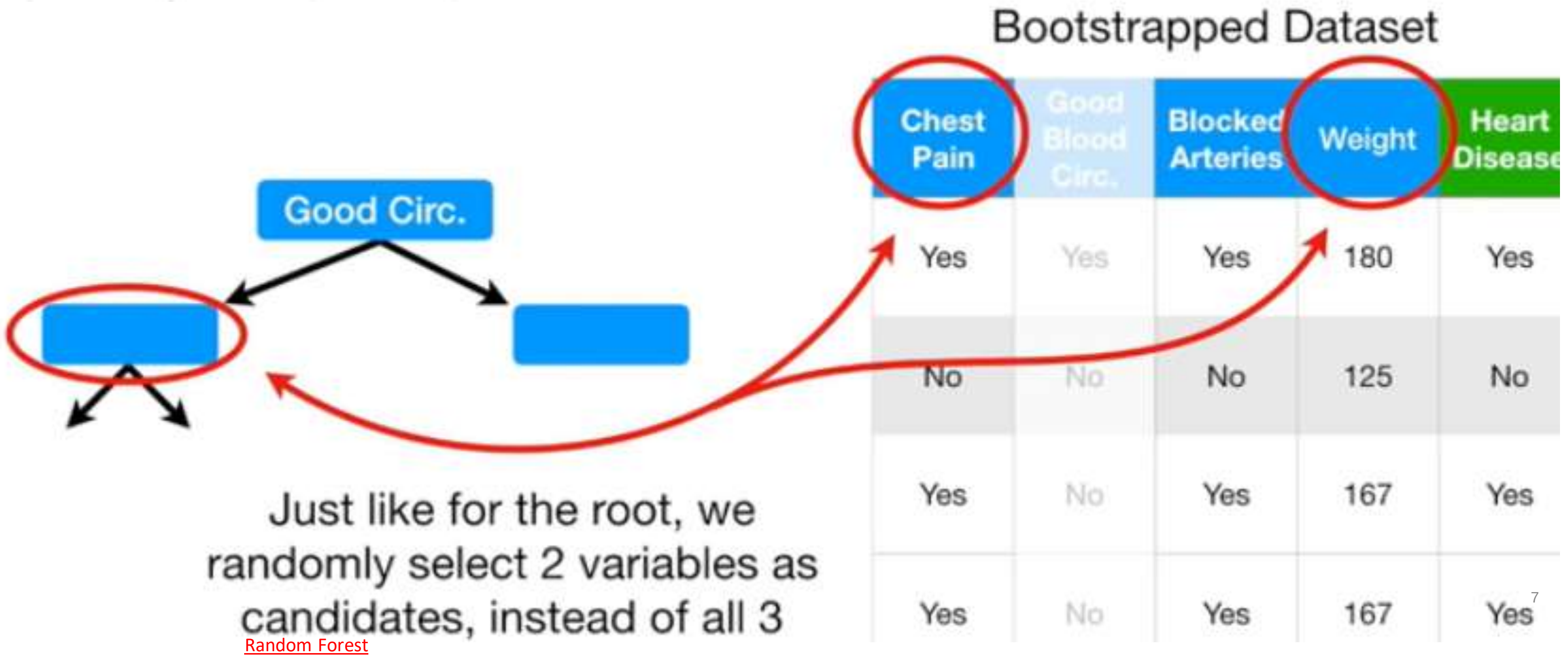
The important detail is that we're allowed to pick the same sample more



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

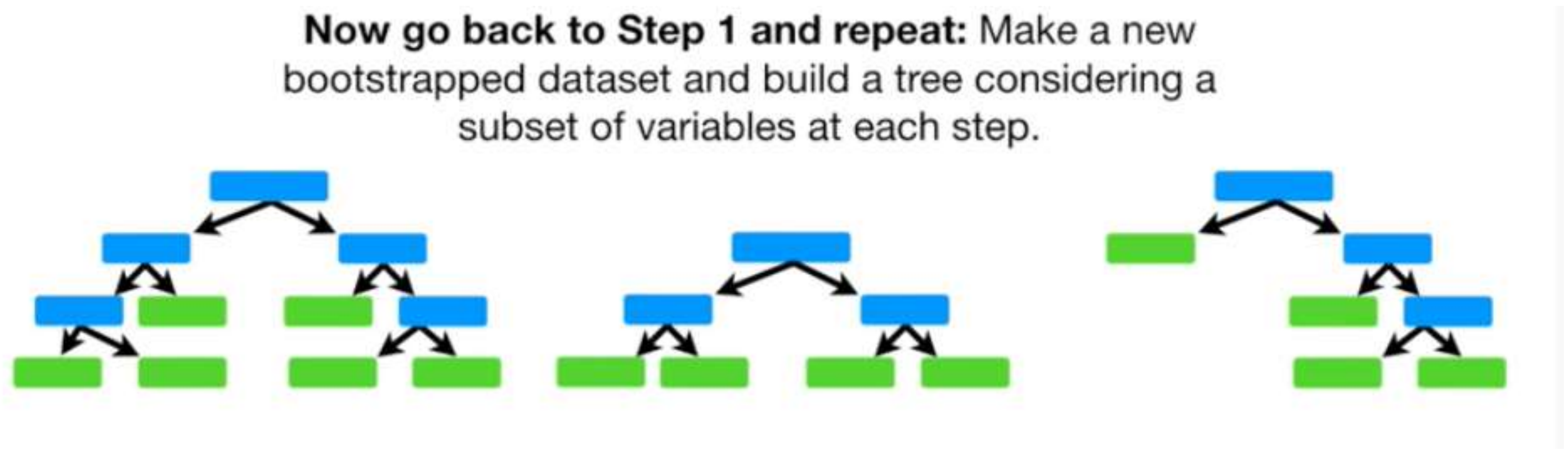
In this case, we randomly selected **Good Blood Circulation** and **Blocked Arteries** as candidates for the root node.



Bootstrapped Dataset

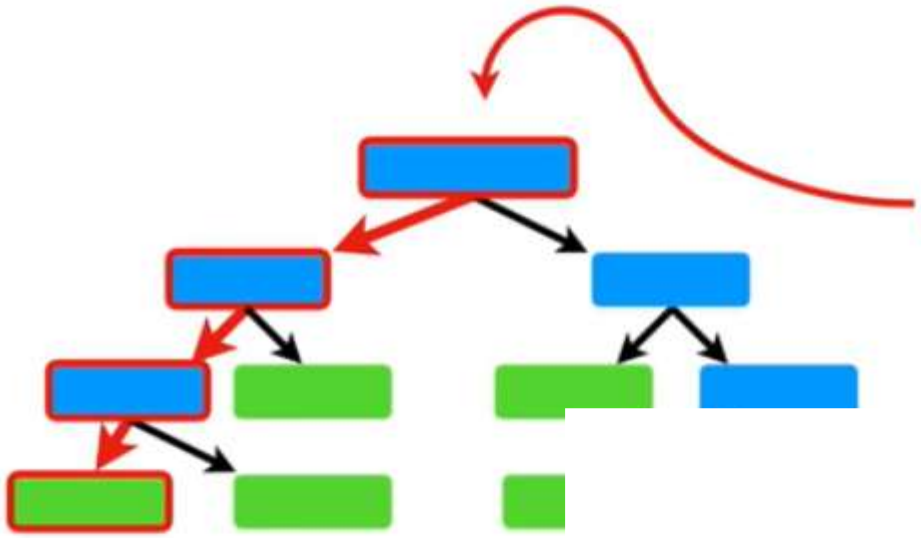
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

Just like for the root, we randomly select 2 variables as candidates, instead of all 3



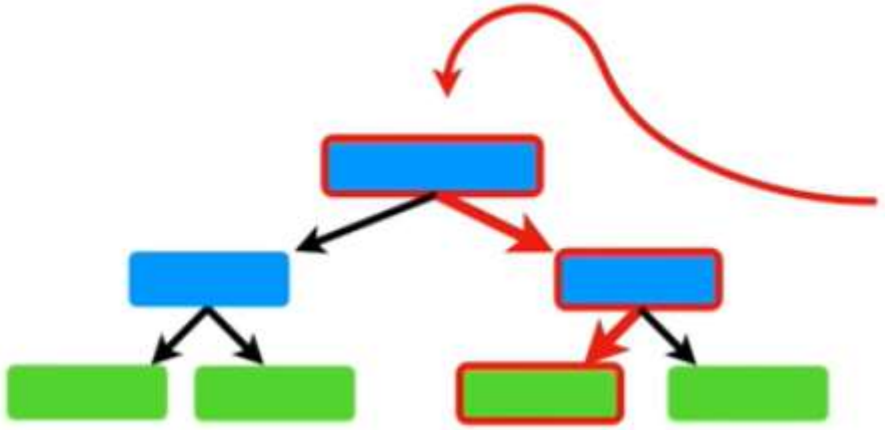
Now go back to Step 1 and repeat: Make a new bootstrapped dataset and build a tree considering a subset of variables at each step.

Bootstrapping the data using the aggregate to make decisions is called Bagging



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

Now we run the data down the second tree that we made…



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

Heart Disease	
Yes	No
1	0

Bagging

- If we split the data in random different ways, decision trees give different results, high variance.
- Bagging: **B**ootstrap **agg**regat**ing** is a method that result in low variance.
- If we had multiple realizations of the data (or multiple samples), we could calculate the predictions multiple times and take the average of the fact that averaging multiple onerous estimations produce less uncertain results

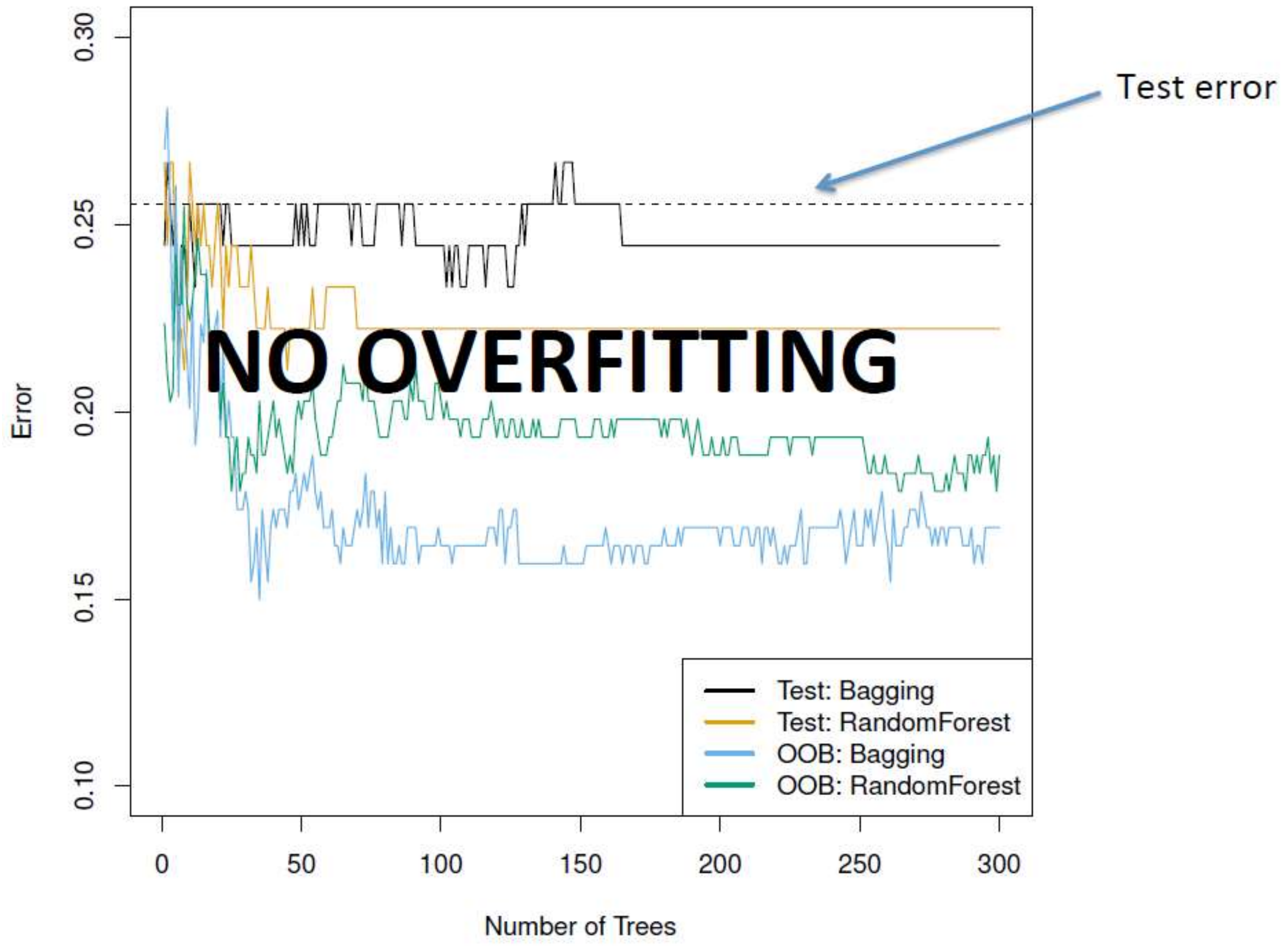
Bagging

- Say for each sample *b*, we calculate $f^b(x)$, then:

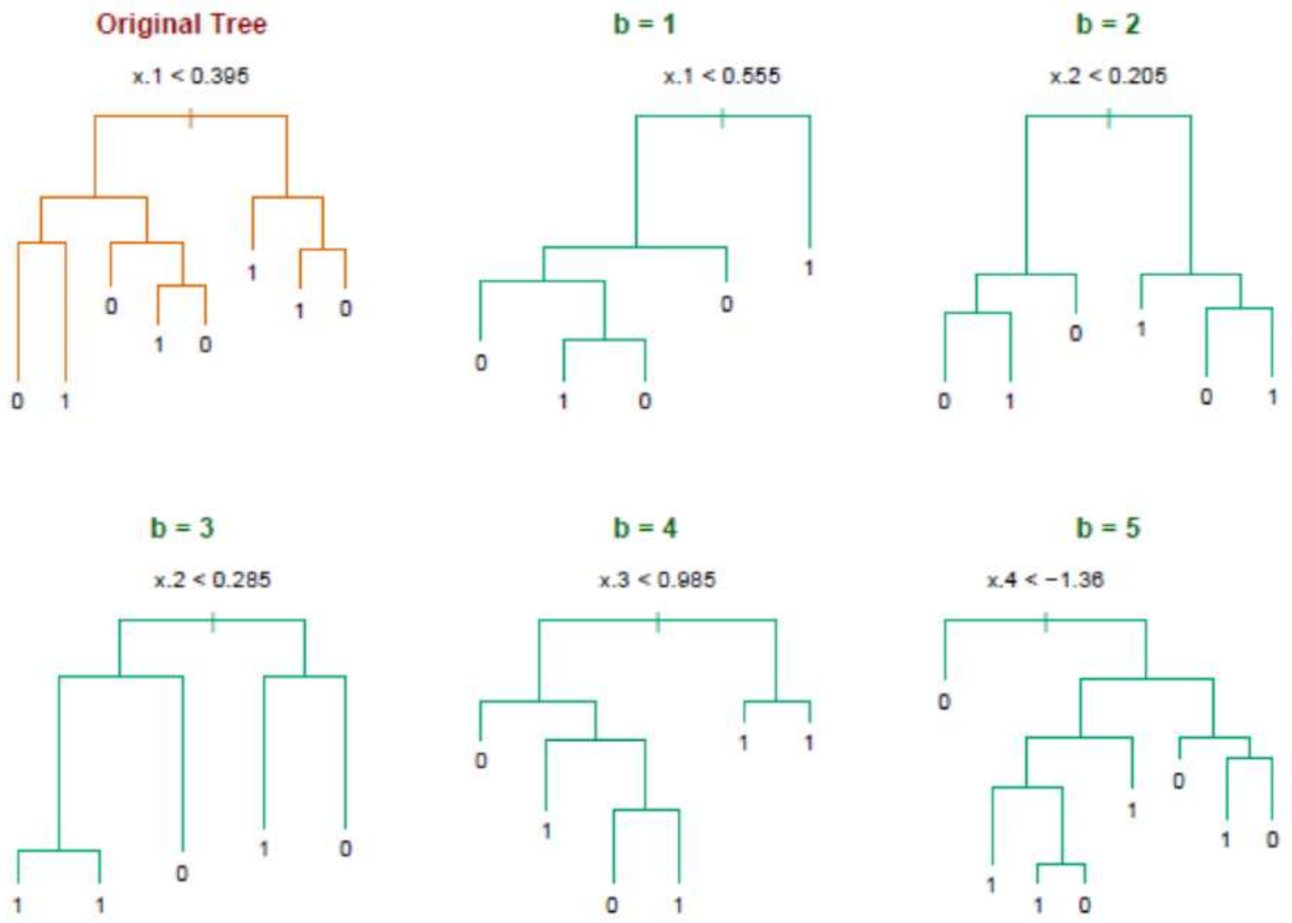
$$\hat{f}_{ave}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_x^b$$

How? $\longrightarrow$ $\boxed{\text{Bootstrap}}$

Bagging for classification: Majority vote



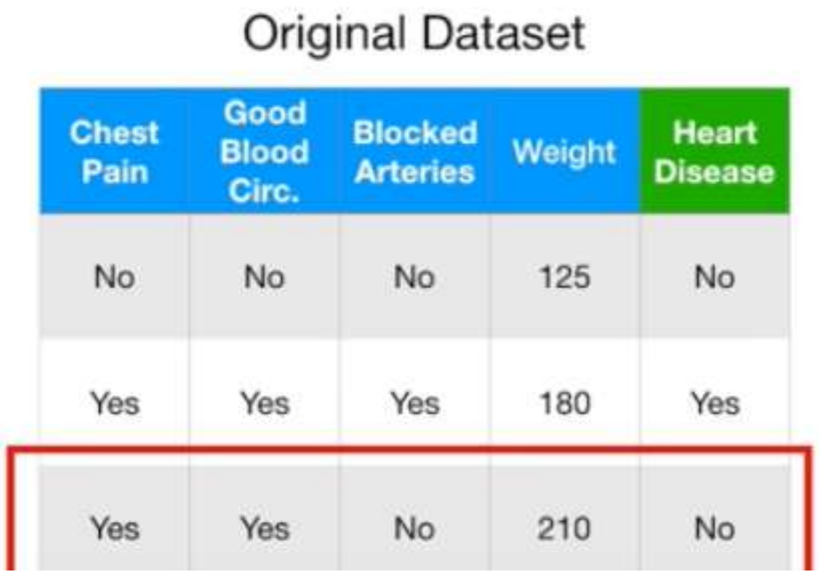
Bagging decision trees



Hastie et al.,"The Elements of Statistical Learning: Data Mining, Inference, and Prediction", Springer (2009)

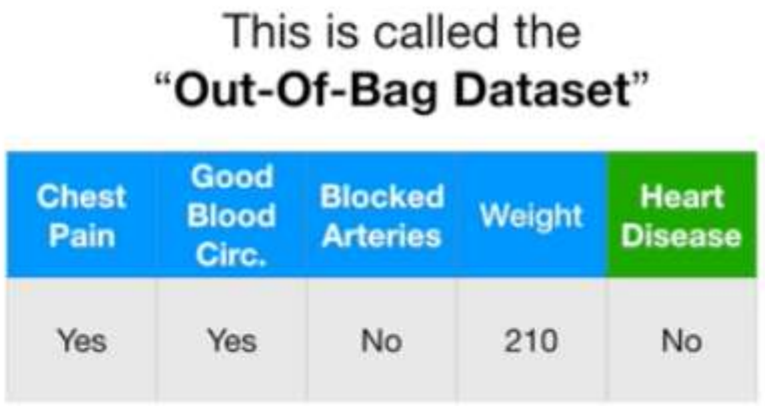
Evaluating the random forest

- 1-create
- 2- use random forest for prediction
- 3-evaluating ……………………..



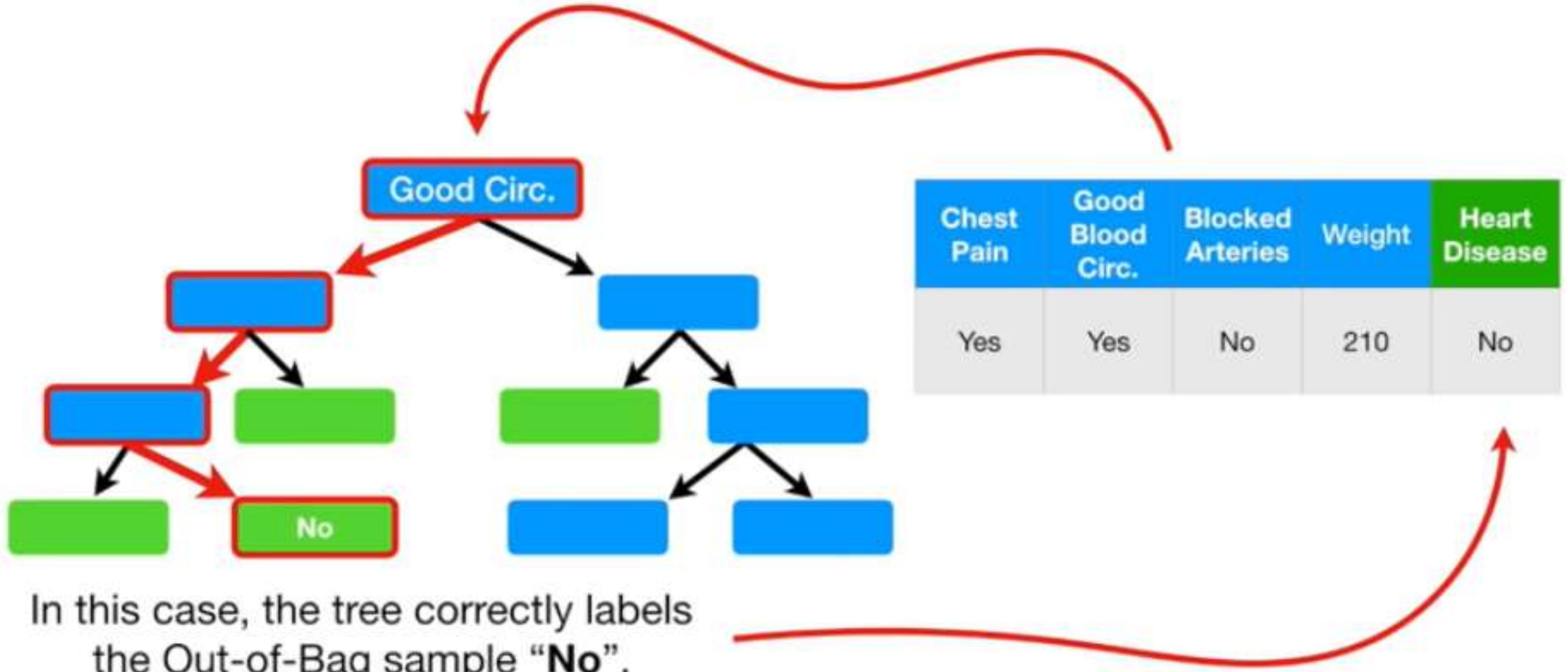
Original Dataset

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



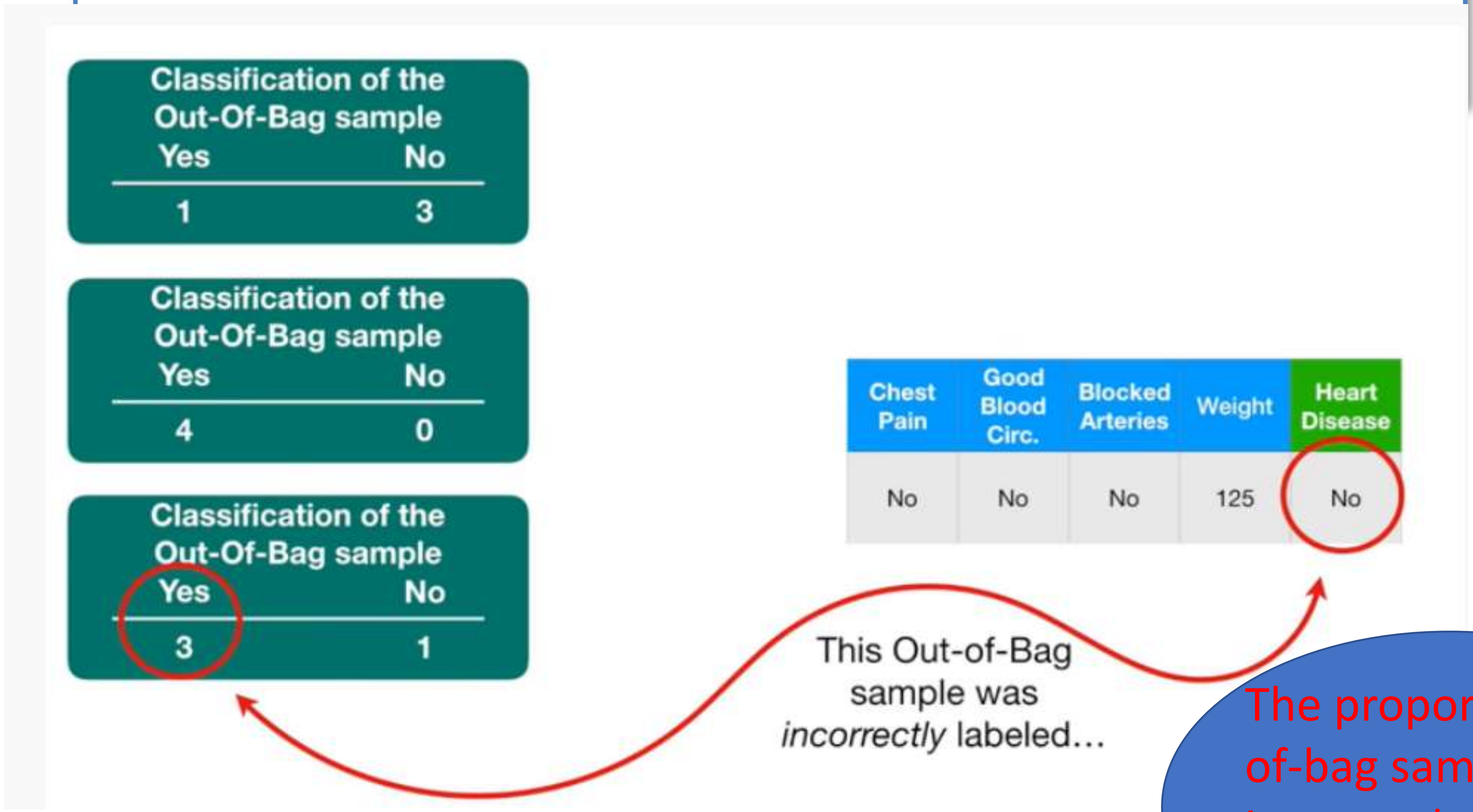
This is called the **"Out-Of-Bag Dataset"**

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



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

In this case, the tree correctly labels the Out-of-Bag sample "**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

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

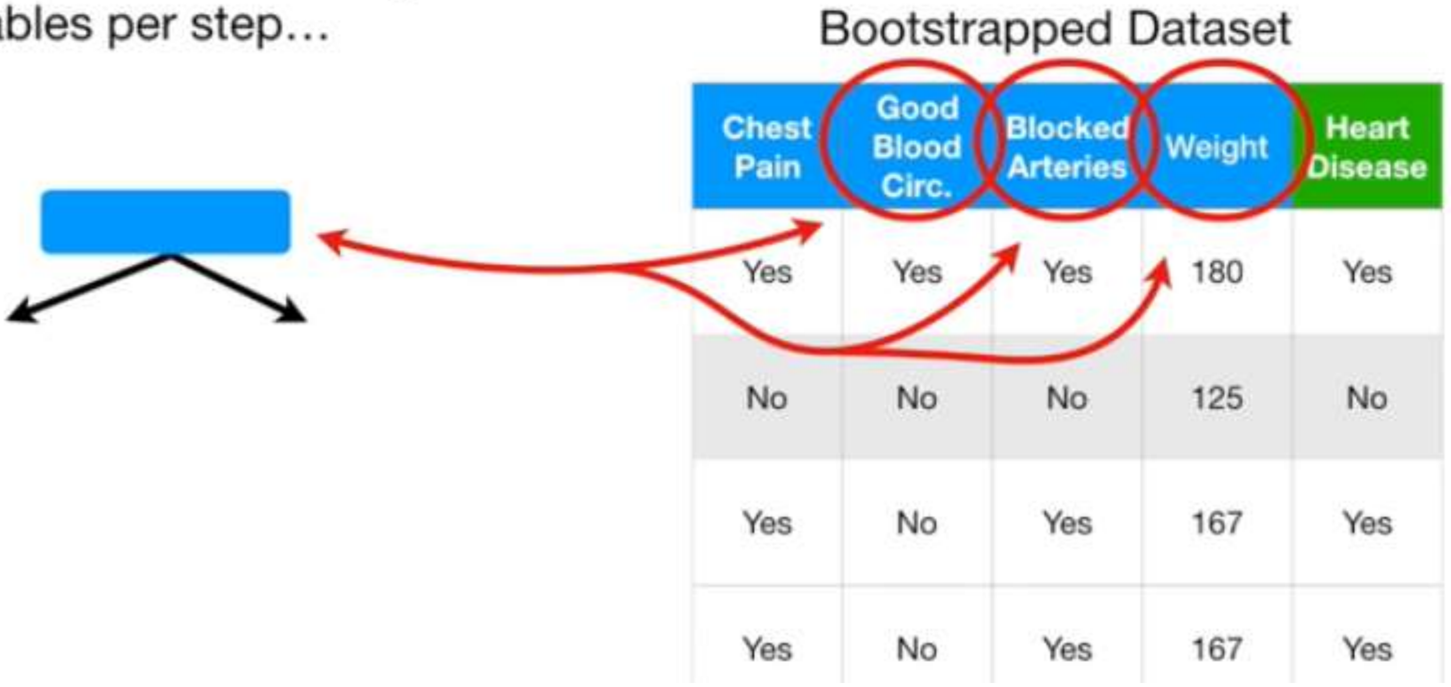
This Out-of-Bag sample was *incorrectly* labeled...

The proportion of out-of-bag samples incorrectly classified is called Out-of-Bag Error

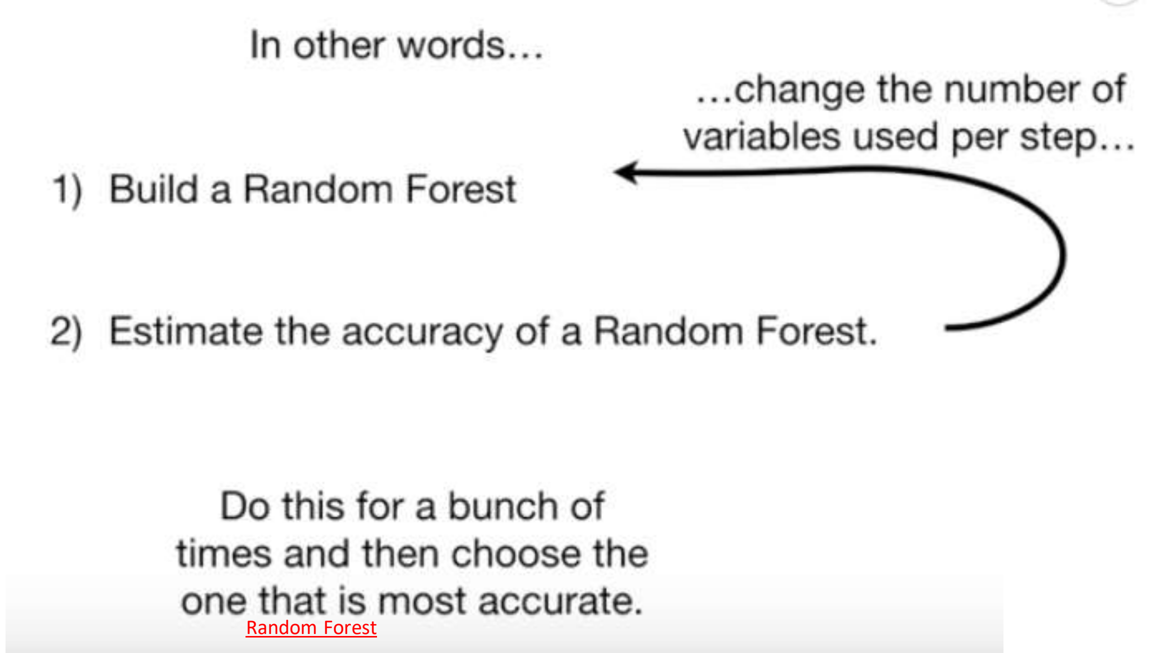
Out-of-Bag Error Estimation

- No cross validation?
- Remember, in bootstrapping we sample with replacement, and therefore not all observations are used for each bootstrap sample. On average 1/3 of them are not used!
- We call them out-of-bag samples (OOB)
- We can predict the response for the i-th observation using each of the trees in which that observation was OOB and do this for n observations
- Calculate overall OOB MSE or classification error

to random forest built using 3
variables per step…



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.

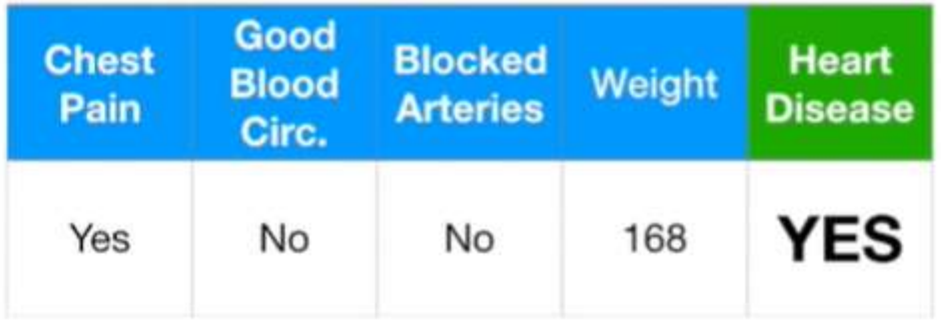
Do this for a bunch of
times and then choose the
one that is most accurate.

Random Forest

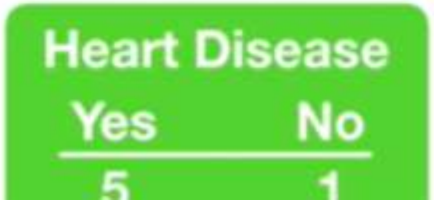
Bagging

- Reduces overfitting (variance)
- Normally uses one type of classifier
- Decision trees are popular
- Easy to parallelize

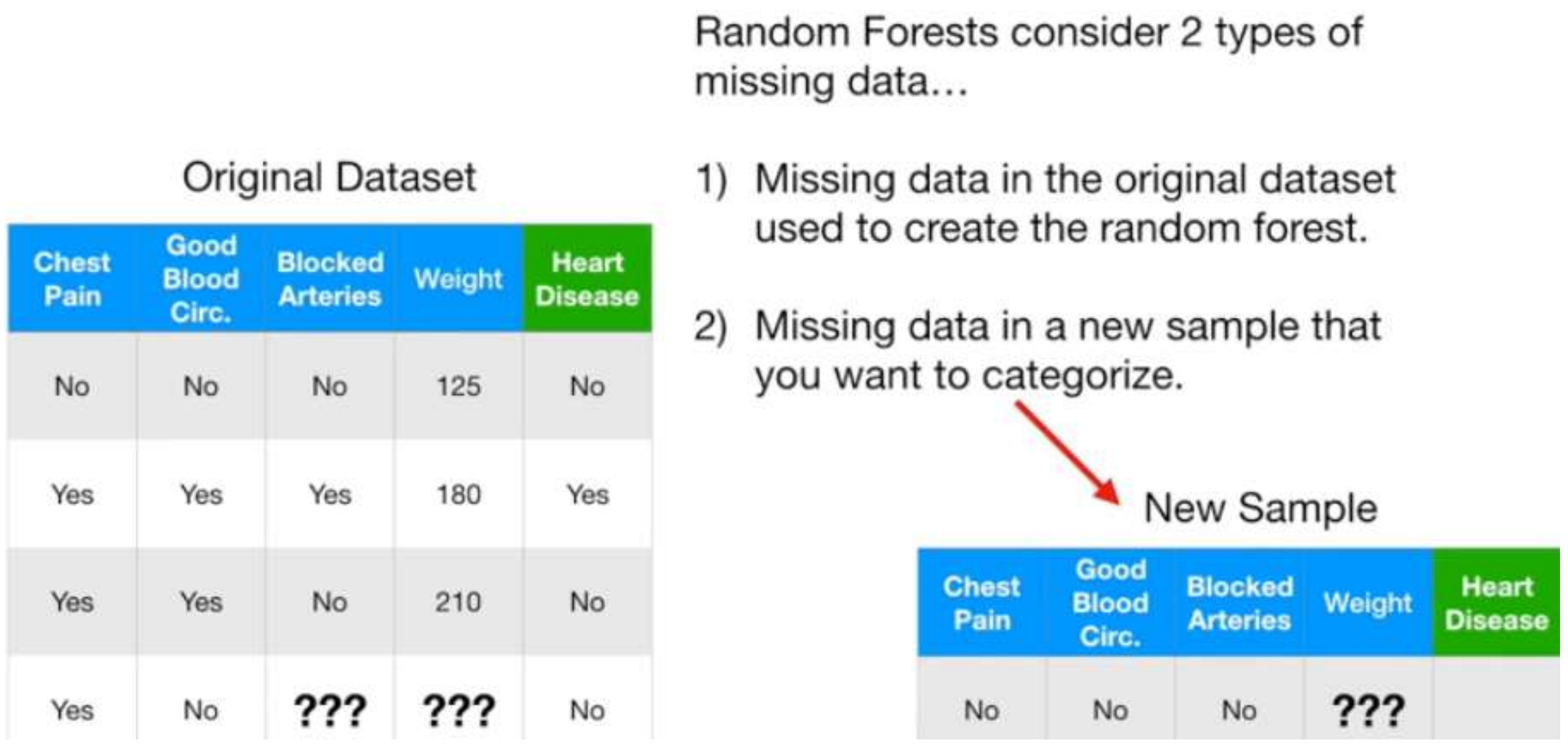
Bootstrapping the data plus using the **agg**regate to make a decision is called “**Bagging**”



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



Heart Disease	
Yes	No
5	1



Random Forests consider 2 types of missing data…

1) Missing data in the original dataset used to create the random forest.

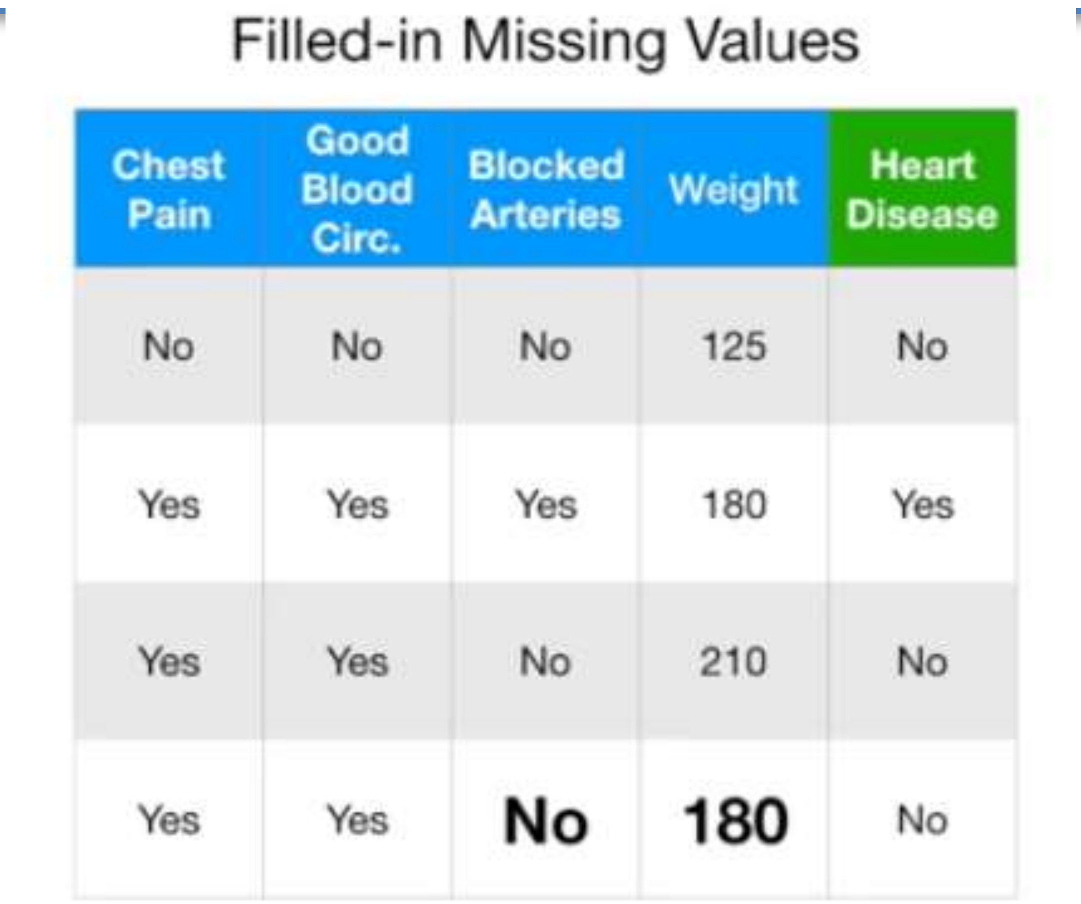
2) Missing data in a new sample that you want to categorize.

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	???	???	No

New Sample

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



Filled-in Missing Values

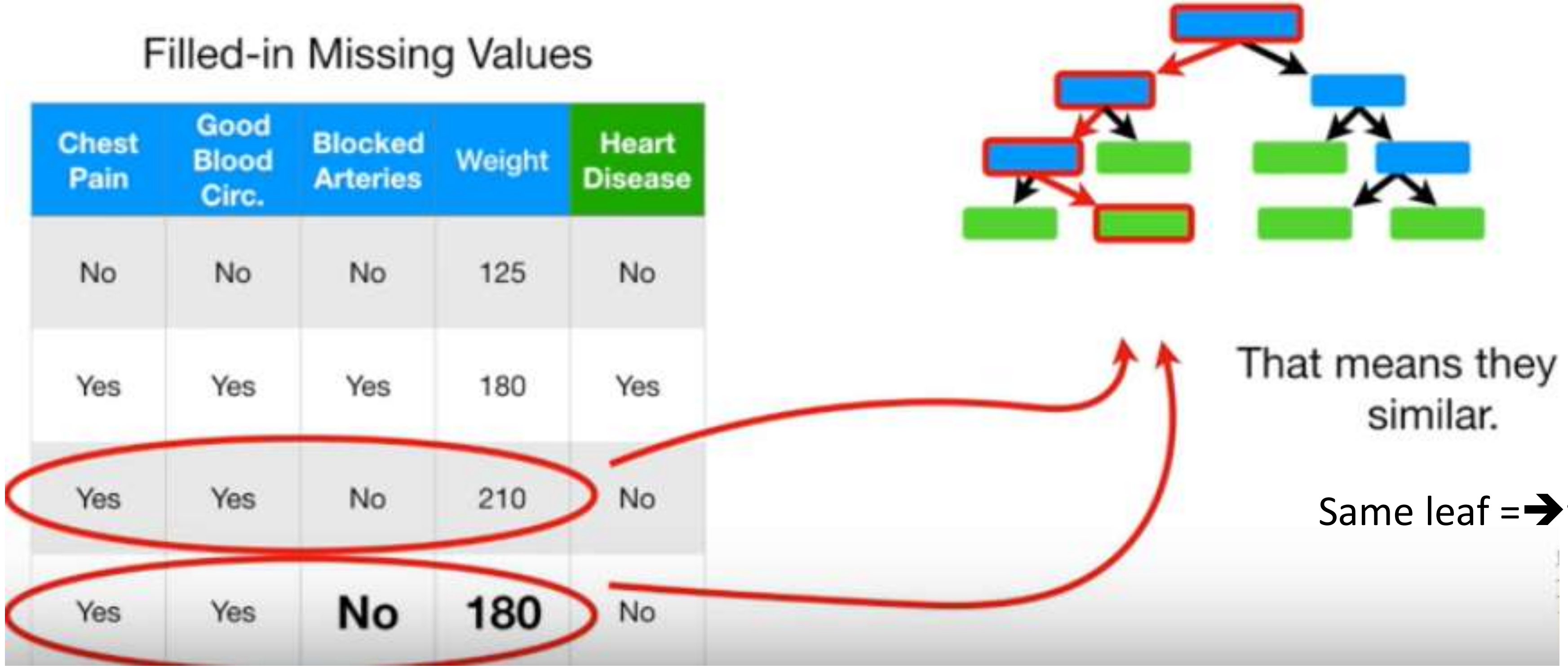
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	Yes	**No**	**180**	No

Categorical fill by voting

Numerical fill by average

Refine guesses ?

1- find similar records to the missing data ➔build all trees
2- run all data on all trees



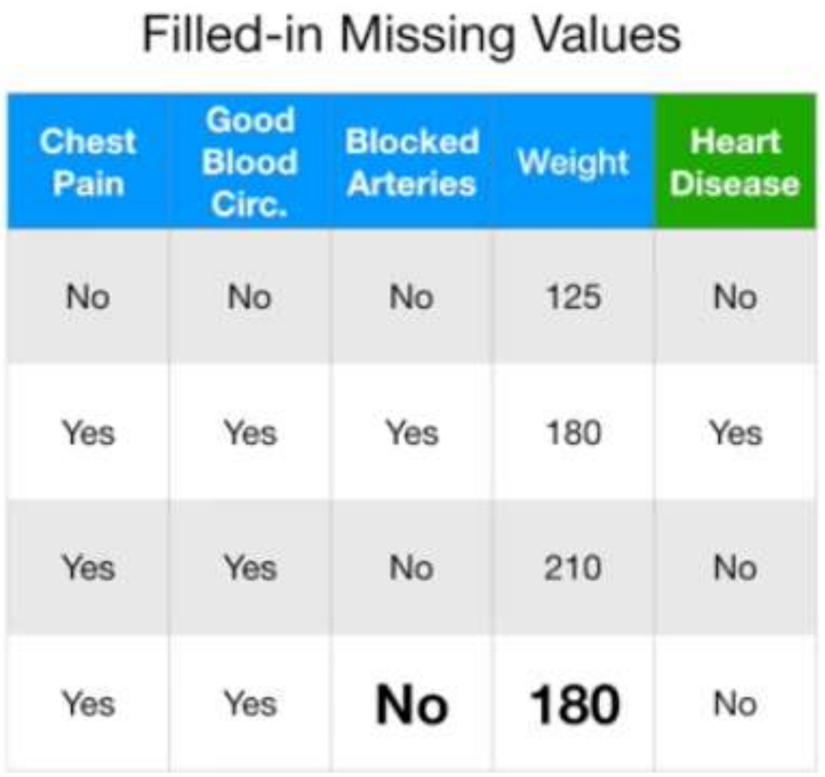
Filled-in Missing Values

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	Yes	**No**	**180**	No

That means they similar.

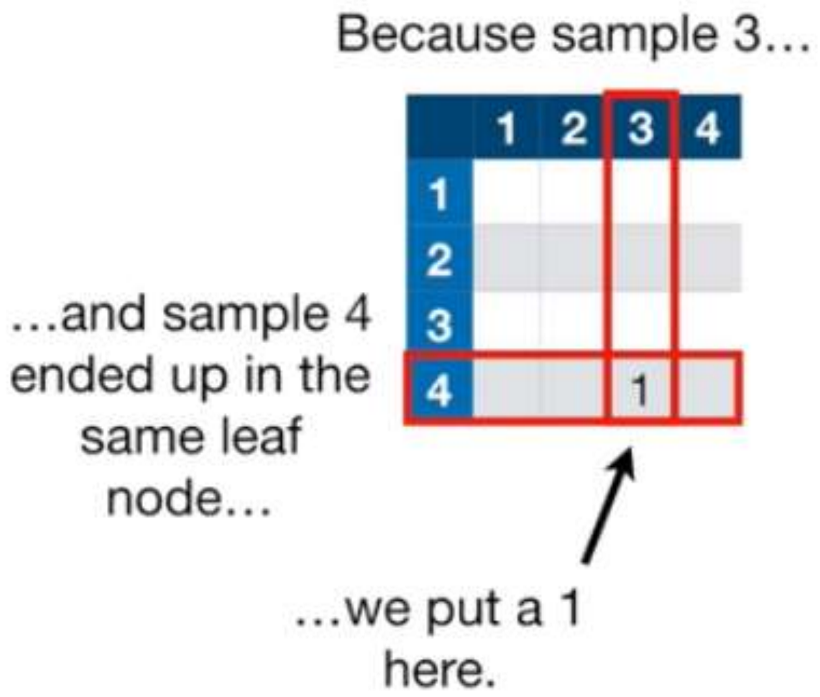
Same leaf =➔they are similar

Proximity Matrix



Filled-in Missing Values

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	Yes	**No**	**180**	No



Because sample 3…

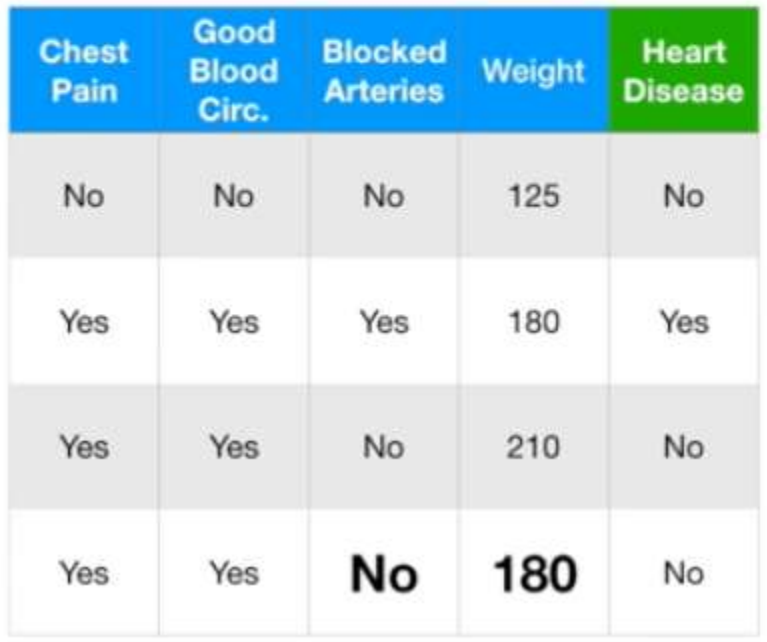
	1	2	3	4
1				
2				
3				
4			1	

…and sample 4 ended up in the same leaf node…

…we put a 1 here.

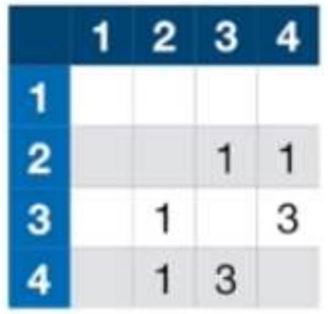
Running tree 1 , tree 2 , tree 3 (3 &4 are similar in the third tree)

Filled-in Missing Values



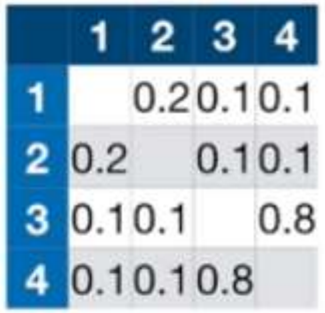
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	Yes	**No**	**180**	No

…and here's the updated proximity matrix.

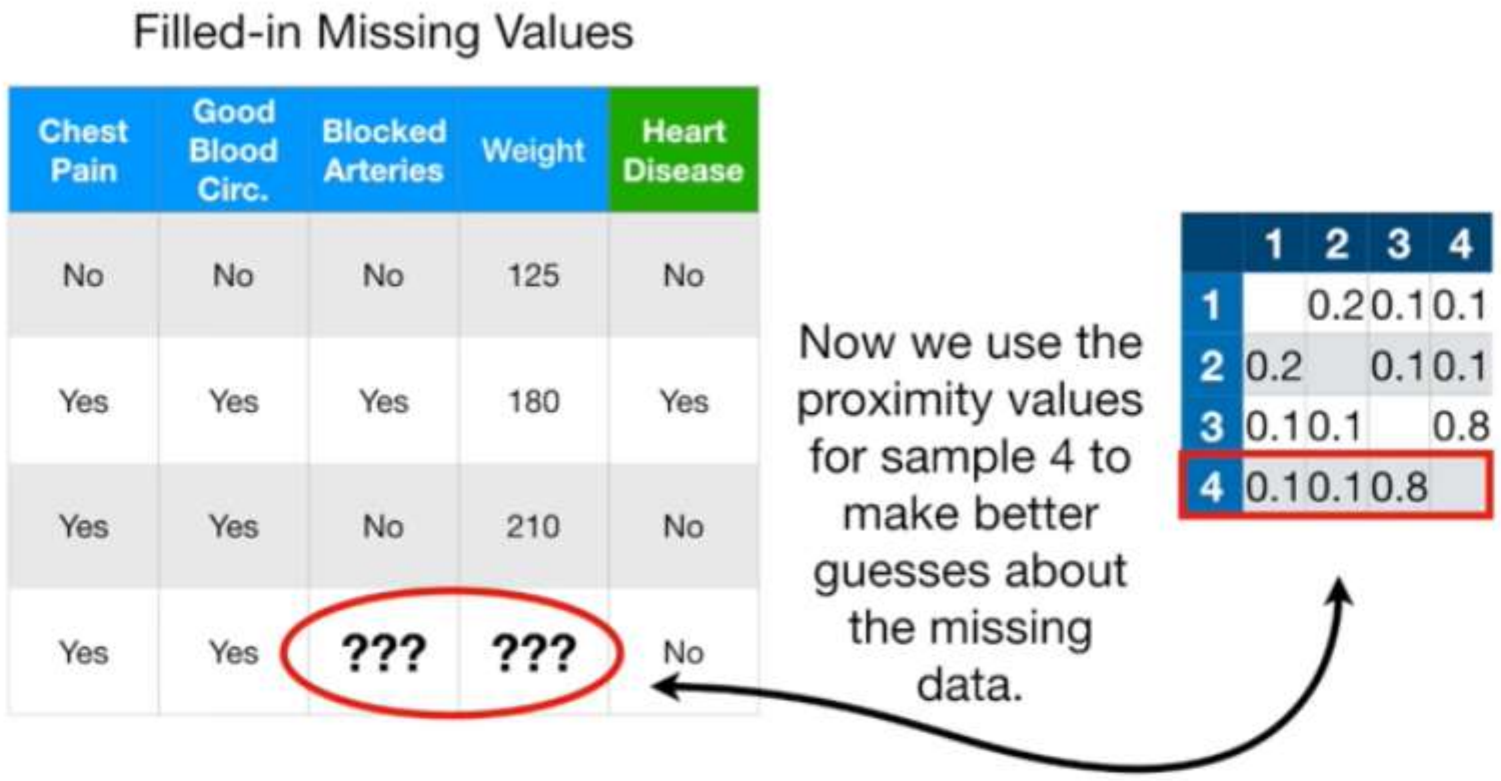


	1	2	3	4
1				
2			1	1
3		1		3
4		1	3	

…..Then divide by total number of trees (e.g 10 trees)



	1	2	3	4
1		0.2	0.1	0.1
2	0.2		0.1	0.1
3	0.1	0.1		0.8
4	0.1	0.1	0.8	



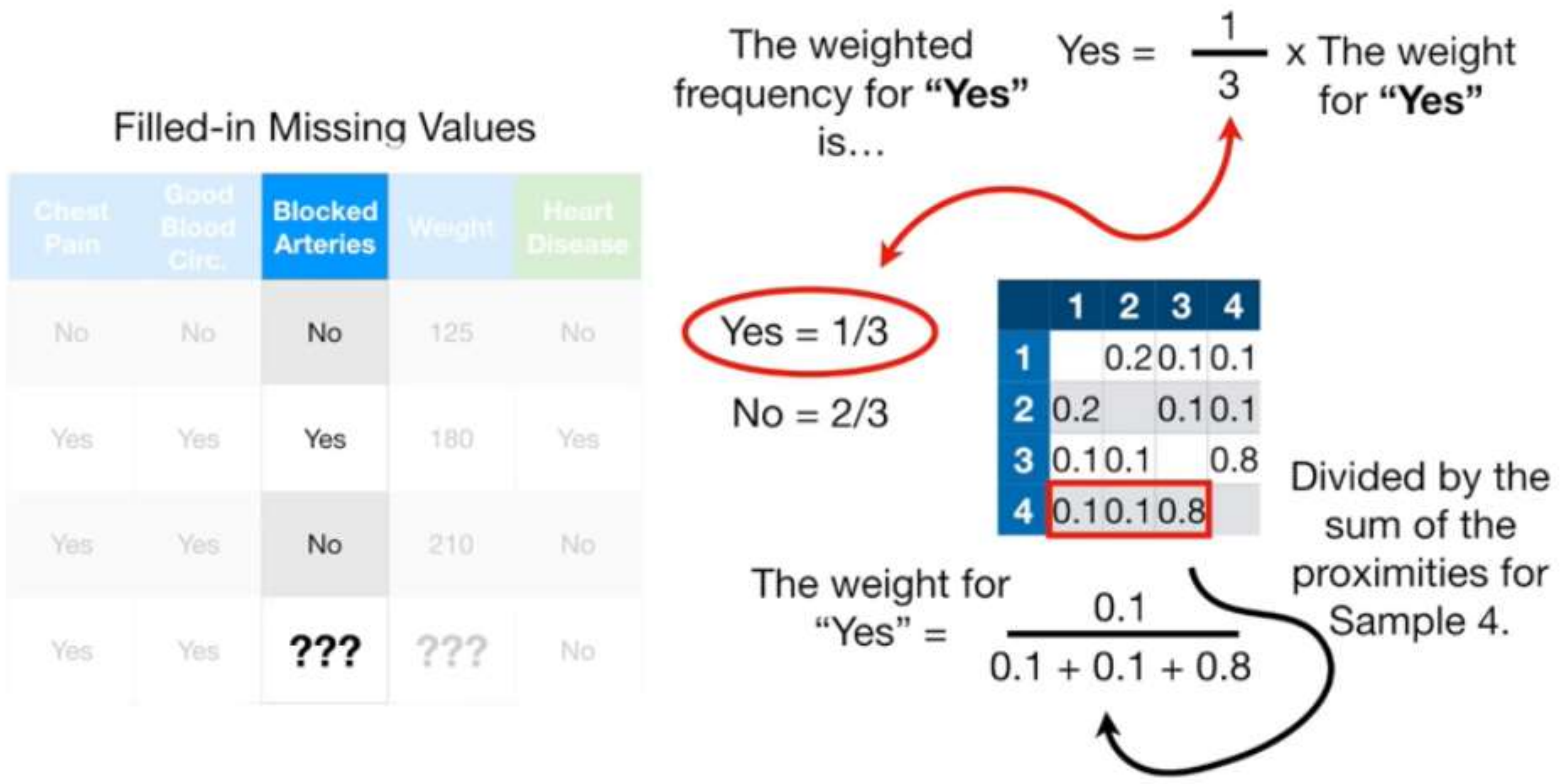
Filled-in Missing Values

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	Yes	???	???	No

Now we use the proximity values for sample 4 to make better guesses about the missing data.

	1	2	3	4
1		0.2	0.1	0.1
2	0.2		0.1	0.1
3	0.1	0.1		0.8
4	0.1	0.1	0.8	

Frequency of Yes



Filled-in Missing Values

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	Yes	???	???	No

The weighted frequency for **"Yes"** is… Yes = $\frac{1}{3}$ x The weight for **"Yes"**

Yes = 1/3

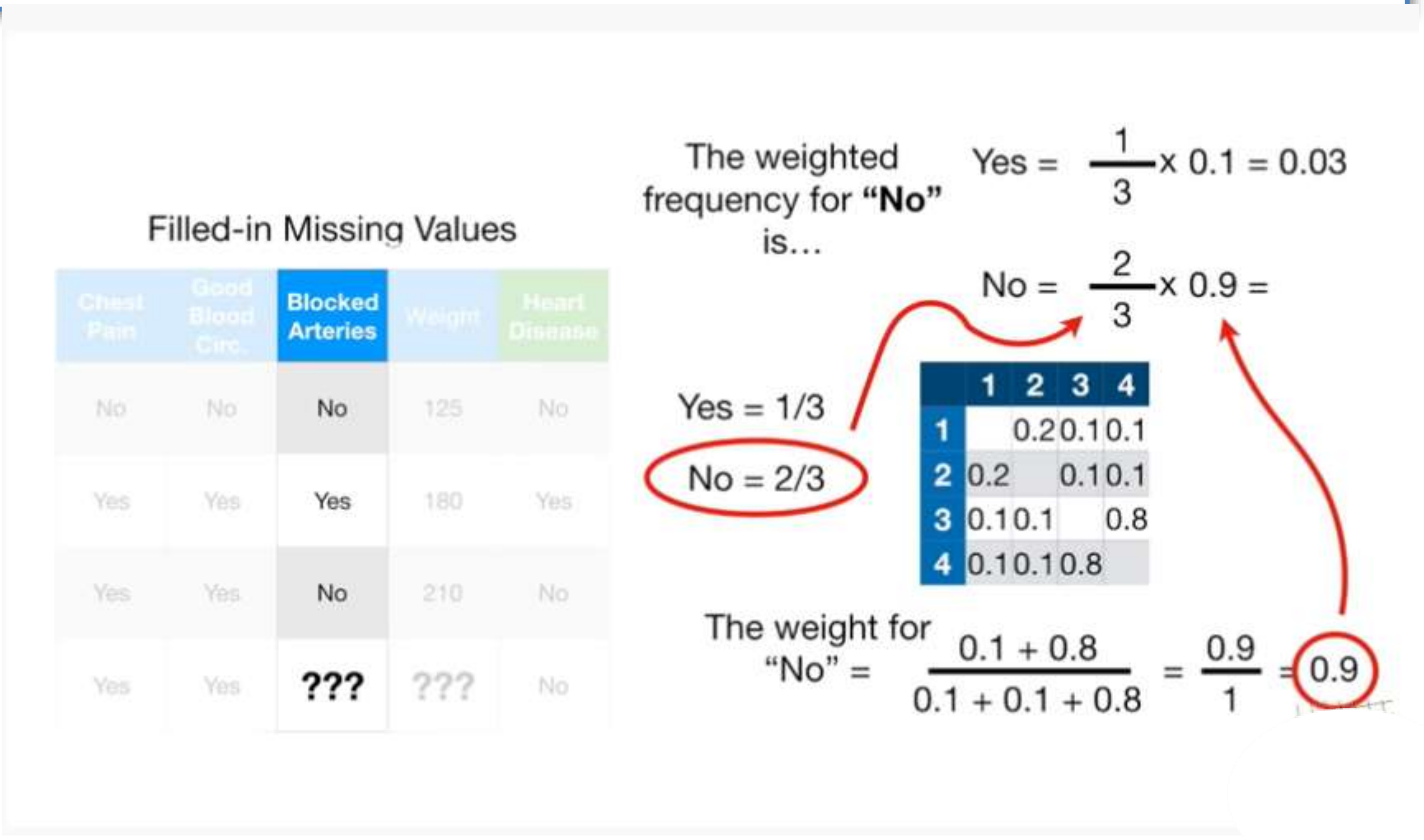
No = 2/3

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & & 0.2 & 0.1 & 0.1 \\ 2 & 0.2 & & 0.1 & 0.1 \\ 3 & 0.1 & 0.1 & & 0.8 \\ 4 & 0.1 & 0.1 & 0.8 & \end{array}$$

The weight for "Yes" = $\frac{0.1}{0.1 + 0.1 + 0.8}$

Divided by the sum of the proximities for Sample 4.

Frequency of No



Filled-in Missing Values

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	Yes	???	???	No

The weighted frequency for **"No"** is...

$$\text{Yes} = \frac{1}{3} \times 0.1 = 0.03$$

$$\text{No} = \frac{2}{3} \times 0.9 =$$

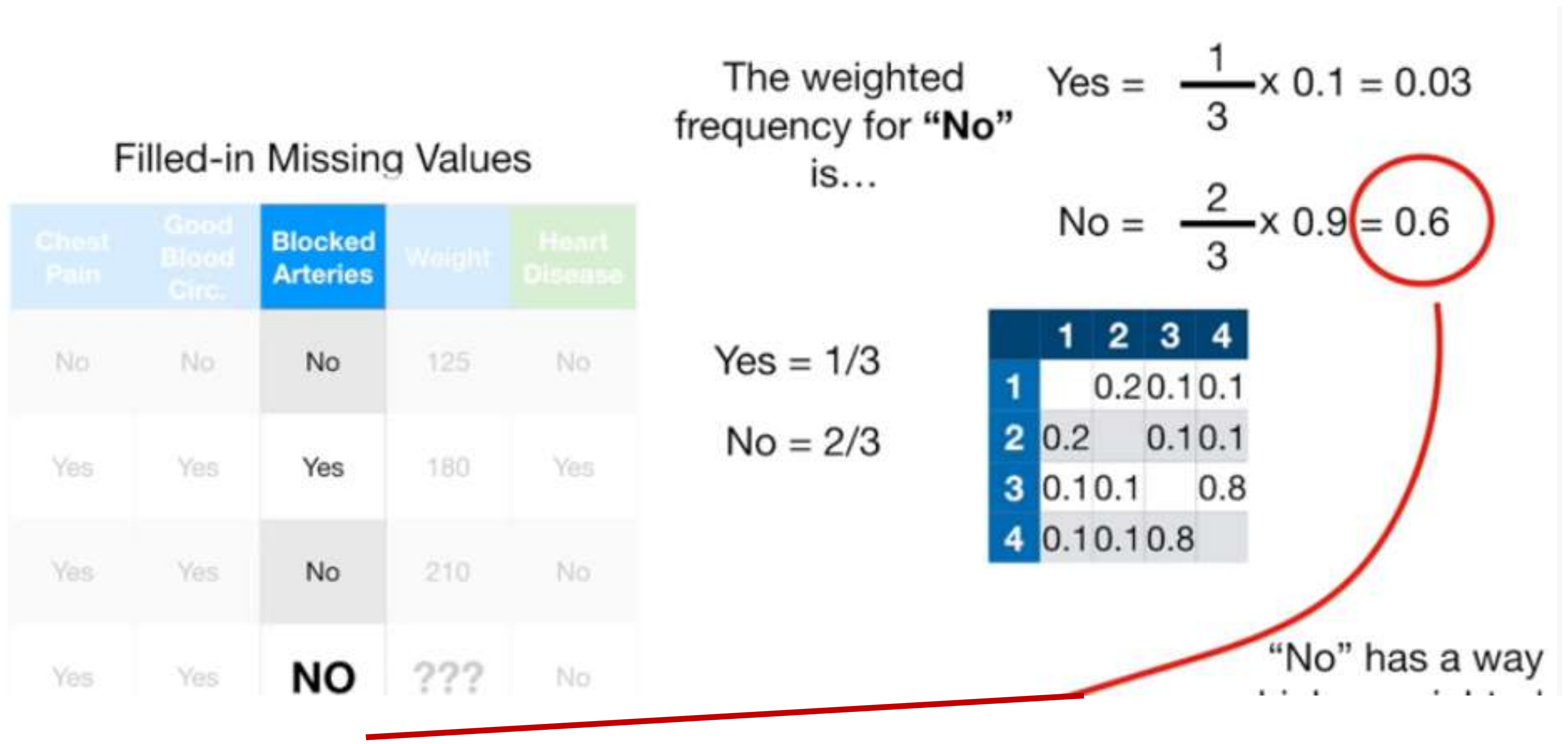
Yes = 1/3

No = 2/3

	1	2	3	4
1		0.2	0.1	0.1
2	0.2		0.1	0.1
3	0.1	0.1		0.8
4	0.1	0.1	0.8	

$$\text{The weight for "No"} = \frac{0.1 + 0.8}{0.1 + 0.1 + 0.8} = \frac{0.9}{1} = \boxed{0.9}$$

Conclusion:
No has a way bigger value



Filled-in Missing Values

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	Yes	NO	???	No

The weighted frequency for **“No”** is…

$$\text{Yes} = \frac{1}{3} \times 0.1 = 0.03$$

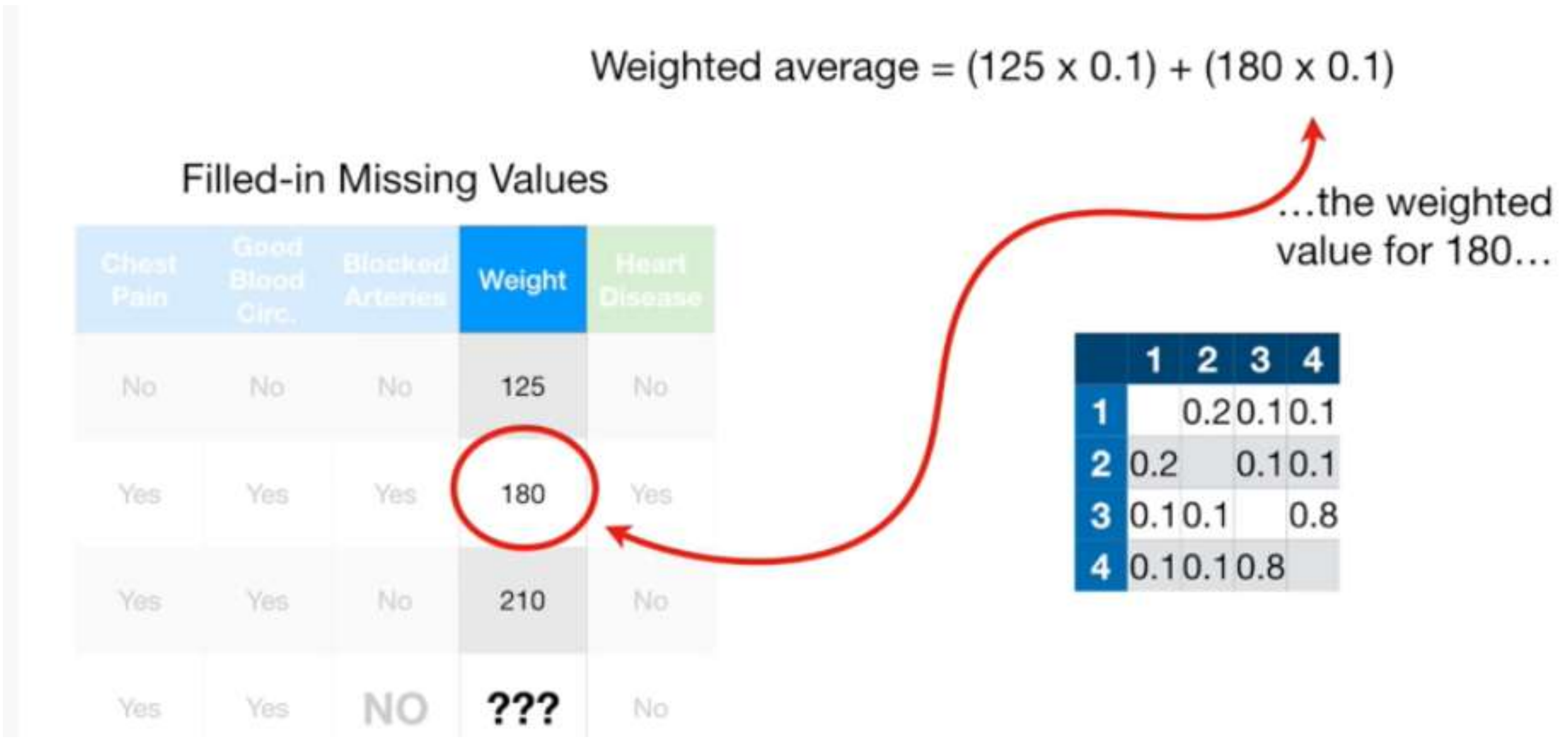
$$\text{No} = \frac{2}{3} \times 0.9 = \boxed{0.6}$$

Yes = 1/3

No = 2/3

	1	2	3	4
1		0.2	0.1	0.1
2	0.2		0.1	0.1
3	0.1	0.1		0.8
4	0.1	0.1	0.8	

“No” has a way



Weighted average = (125 x 0.1) + (180 x 0.1)

Filled-in Missing Values

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	Yes	NO	???	No

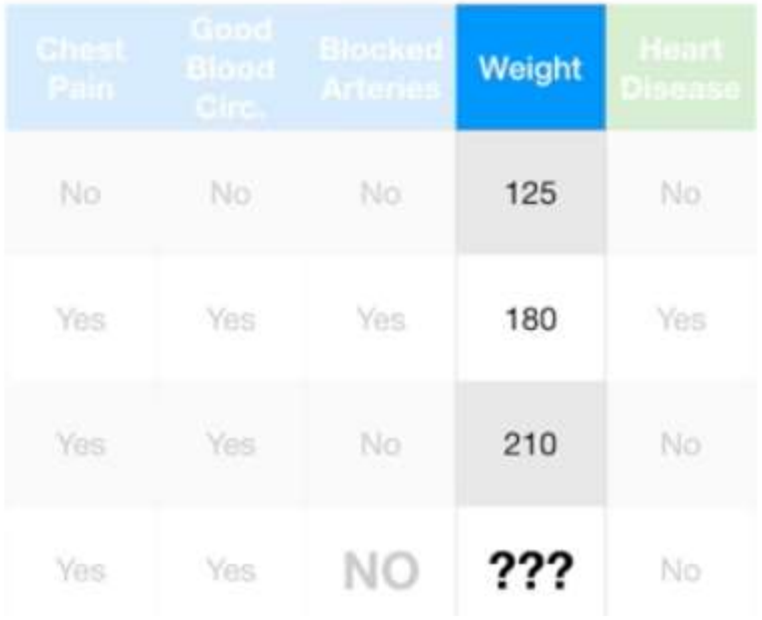
…the weighted value for 180…

	1	2	3	4
1		0.2	0.1	0.1
2	0.2		0.1	0.1
3	0.1	0.1		0.8
4	0.1	0.1	0.8	

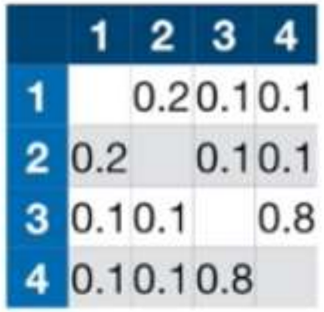
Weighted average = (125 x 0.1) + (180 x 0.1) + (210 x 0.8)

= 198.5

Filled-in Missing Values

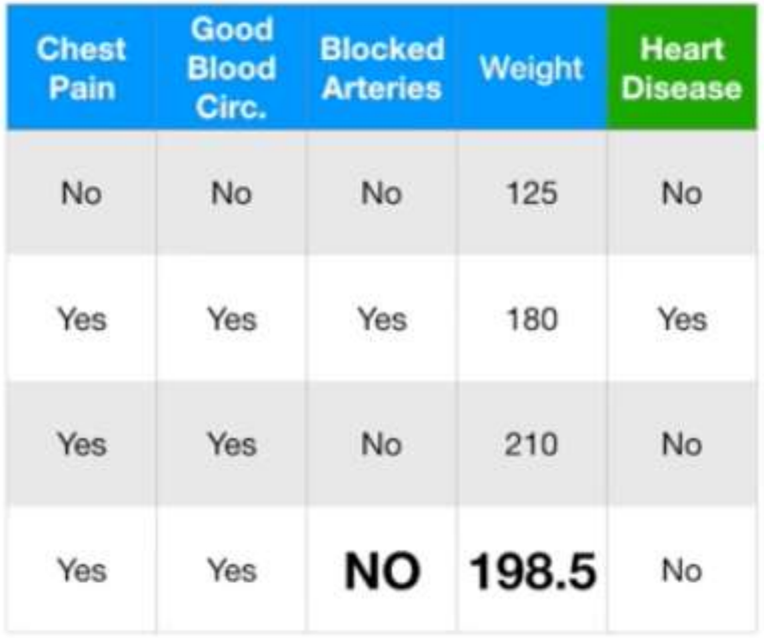


Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	Yes	NO	???	No



	1	2	3	4
1		0.2	0.1	0.1
2	0.2		0.1	0.1
3	0.1	0.1		0.8
4	0.1	0.1	0.8	

New Guesses !



Filled-in Missing Values

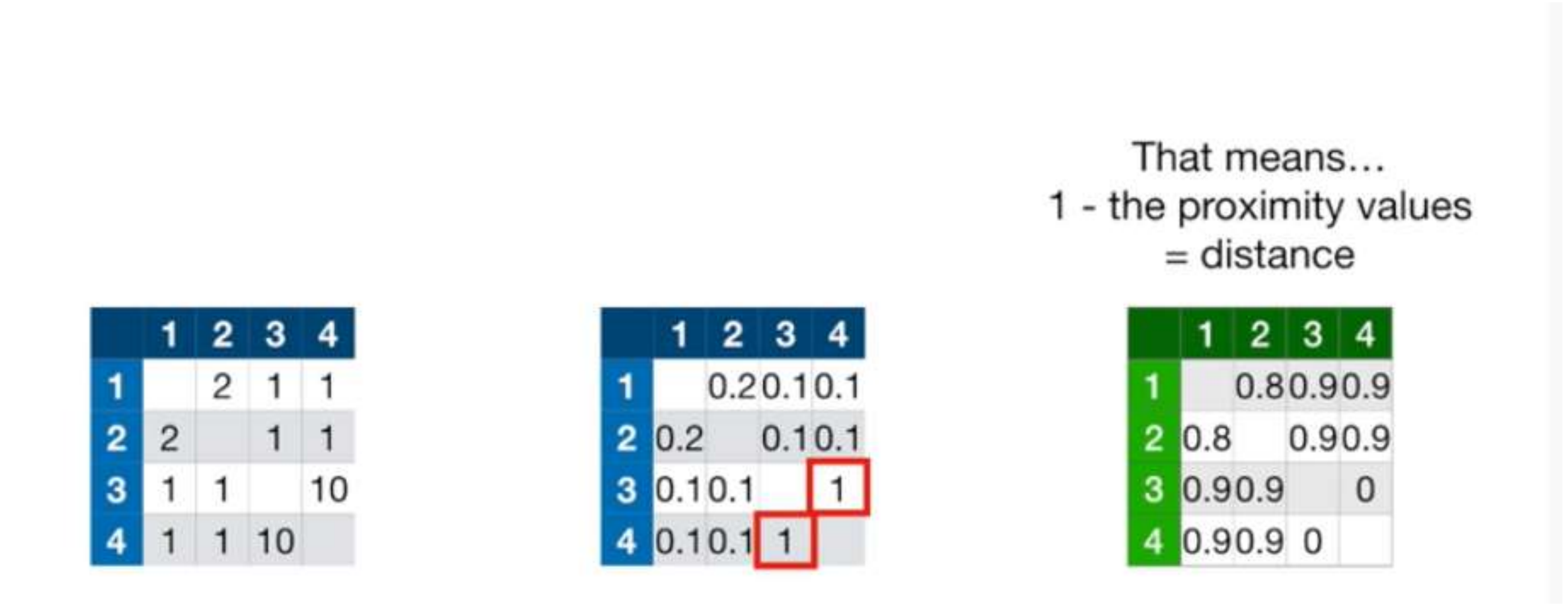
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	Yes	**NO**	**198.5**	No

Now that we've revised our guesses a little bit, we do the whole thing over again…

We build a random forest, run the data through the trees, recalculate the proximities and recalculate the missing values.

We do this 6 or 7 times until the missing values converge (i.e. no longer change each time we recalculate).

Proximity matrix removes the complexity of data types



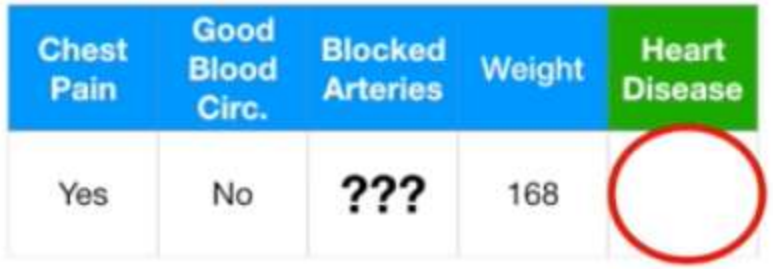
That means…
1 - the proximity values
= distance

	1	2	3	4
1		2	1	1
2	2		1	1
3	1	1		10
4	1	1	10	

	1	2	3	4
1		0.2	0.1	0.1
2	0.2		0.1	0.1
3	0.1	0.1		1
4	0.1	0.1	1	

	1	2	3	4
1		0.8	0.9	0.9
2	0.8		0.9	0.9
3	0.9	0.9		0
4	0.9	0.9	0	

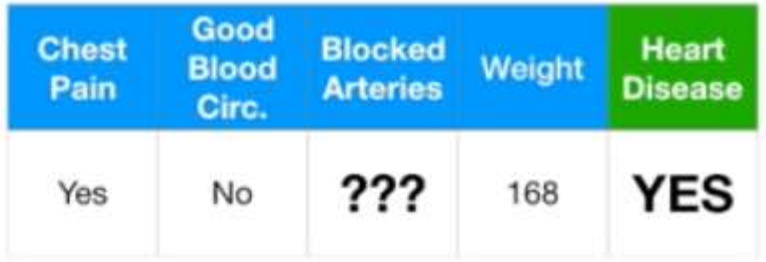
Missing data in the sample that we want to classify !!



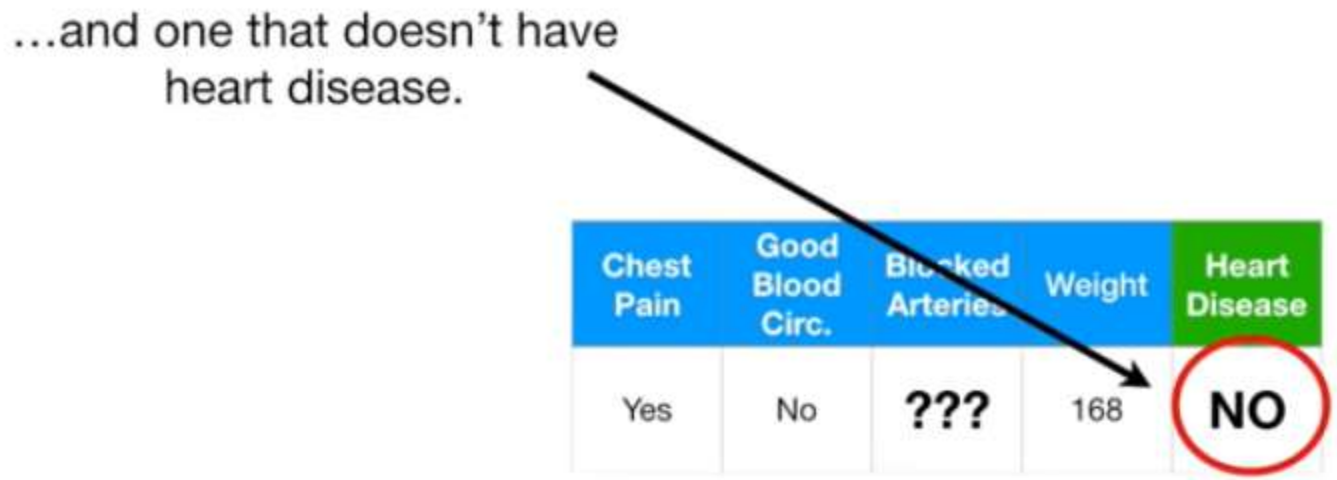
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	No	???	168	

So we want to know if they have heart disease or not…

Create two instances one yes and one no



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



...and one that doesn't have heart disease.

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

Try the guess again....................................

Variable Importance Measures

- Bagging results in improved accuracy over prediction using a single tree
- Unfortunately, difficult to interpret the resulting model. Bagging improves prediction accuracy at the expense of interpretability.
- Calculate the total amount that the RSS or Gini index is decreased due to splits over a given predictor, averaged over all B trees.

Bagging

- Each tree is identically distributed (i.d.)

$\rightarrow$ the expectation of the average of B such trees is the same as the expectation of any one of them

$\rightarrow$ the bias of bagged trees is the same as that of the individual trees

- i.d. and not i.i.d

Why does bagging generate correlated trees?

- Suppose that there is one very strong predictor in the data set, along with a number of other moderately strong predictors.
- Then all bagged trees will select the strong predictor at the top of the tree and therefore all trees will look similar.
- How do we avoid this?
- What if we consider only a subset of the predictors at each split?
- We will still get correlated trees unless …. we randomly select the subset !

Random Forest, Ensemble Model

- The random forest (Breiman, 2001) is an ensemble approach that can also be thought of as a form of nearest neighbor predictor.
- Ensembles are a divide-and-conquer approach used to improve performance. The main principle behind ensemble methods is that a group of "weak learners" can come together to form a "strong learner".

Trees and Forests

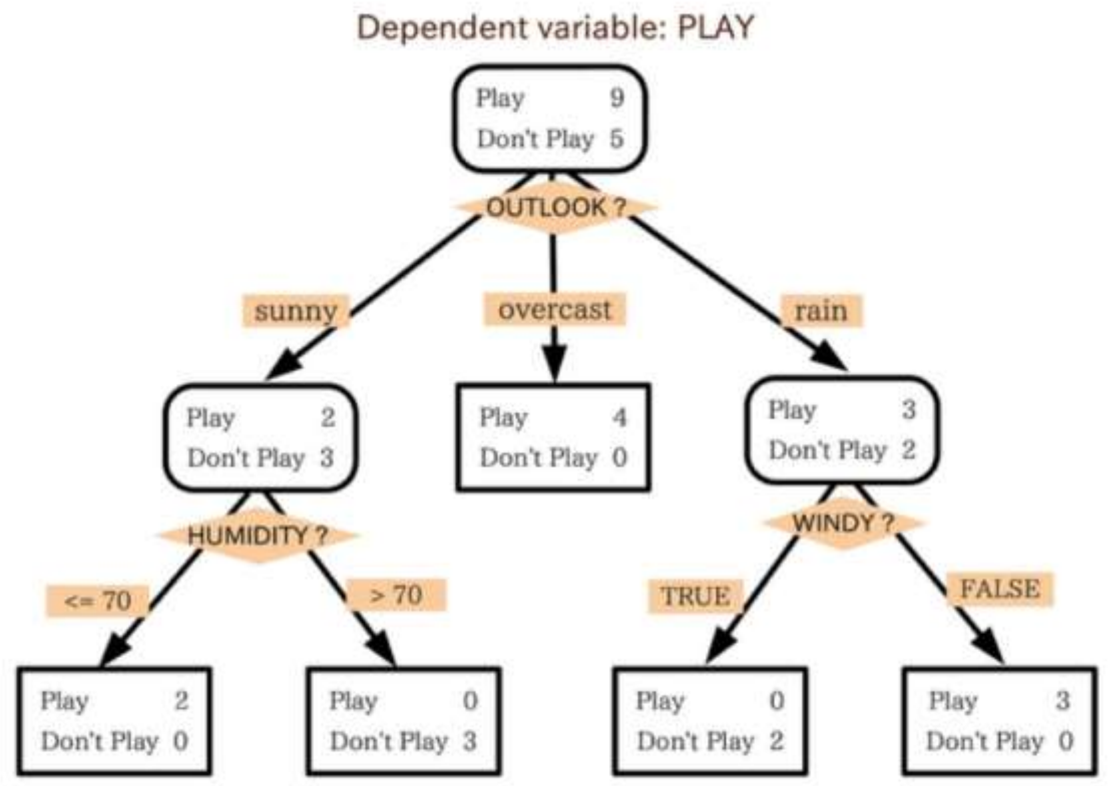
- The random forest starts with a standard machine learning technique called a “decision tree” which, in ensemble terms, corresponds to our weak learner. In a decision tree, an input is entered at the top and as it traverses down the tree the data gets bucketed into smaller and smaller sets.

Random Forest

- As in bagging, we build a number of decision trees on bootstrapped training samples each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors.
- Note that if m = p, then this is bagging.

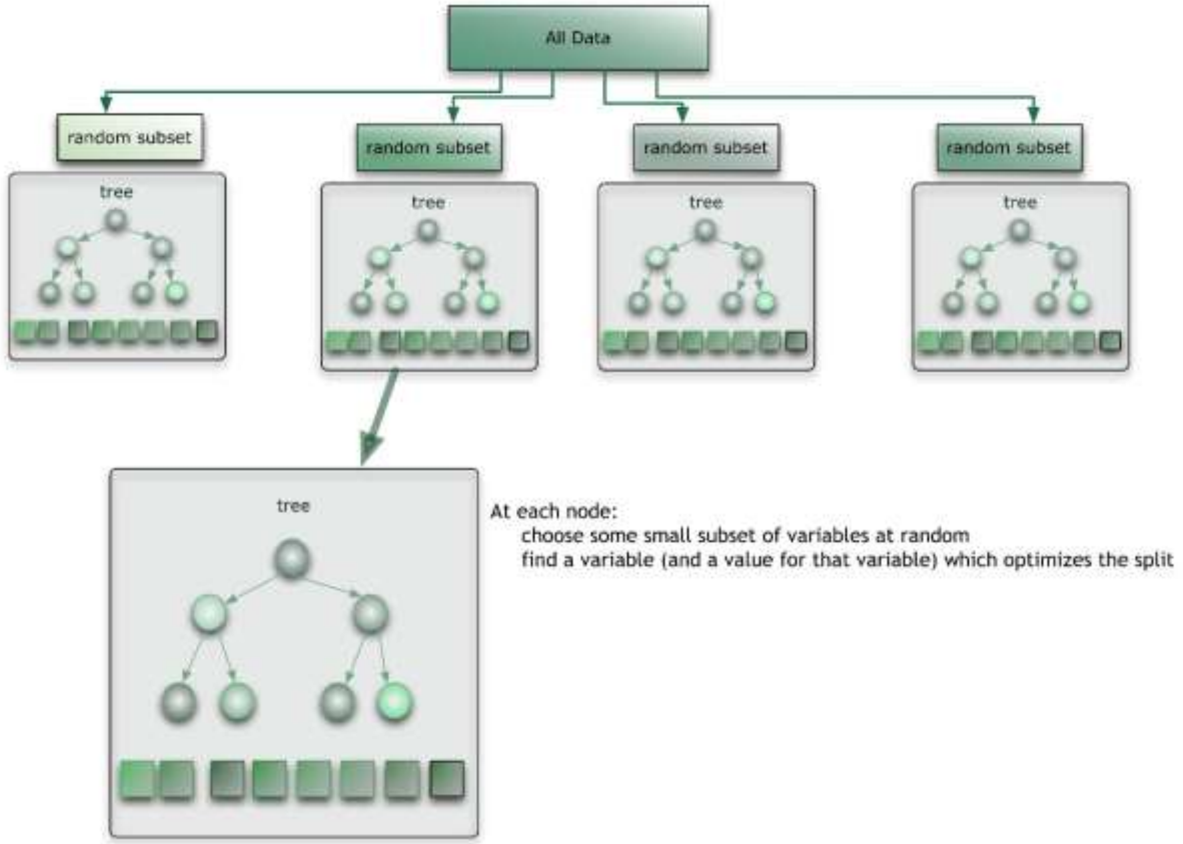
Trees and Forests

- In this example, the tree advises us, based upon weather conditions, whether to play ball. For example, if the outlook is sunny and the humidity is less than or equal to 70, then it's probably OK to play.



Trees and Forests

- The random forest takes this notion to the next level by combining trees with the notion of an ensemble. Thus, in ensemble terms, the trees are weak learners and the random forest is a strong learner.


All Data
random subset
random subset
random subset
random subset
tree
tree
tree
tree
tree
At each node:
choose some small subset of variables at random
find a variable (and a value for that variable) which optimizes the split

Random Forest

Random Forest Algorithm

- For b = 1 to B:

(a) Draw a bootstrap sample Z^* of size N from the training data.

(b) Grow a random-forest tree to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.

i. Select m variables at random from the p variables.

ii. Pick the best variable/split-point among the m.

iii. Split the node into two daughter nodes. Output the ensemble of trees.

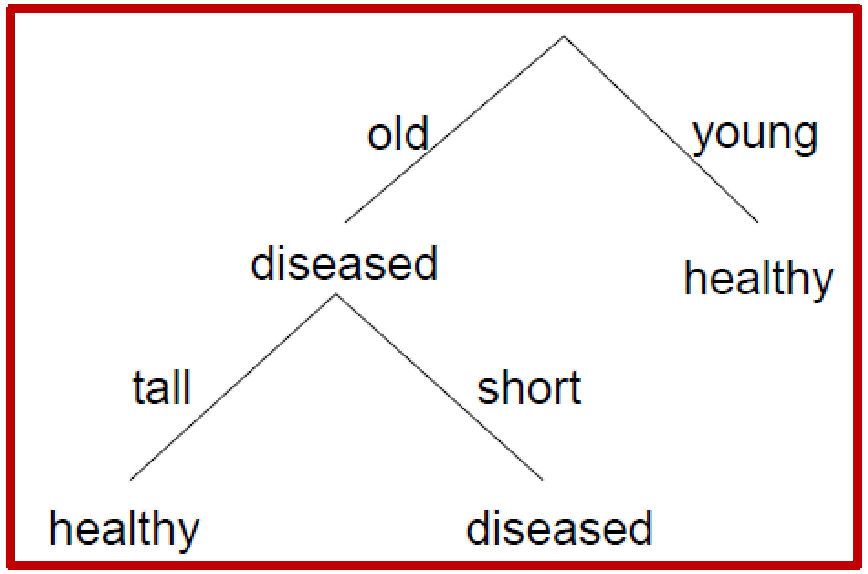
Random Forest Algorithm

- To make a prediction at a new point x we do:

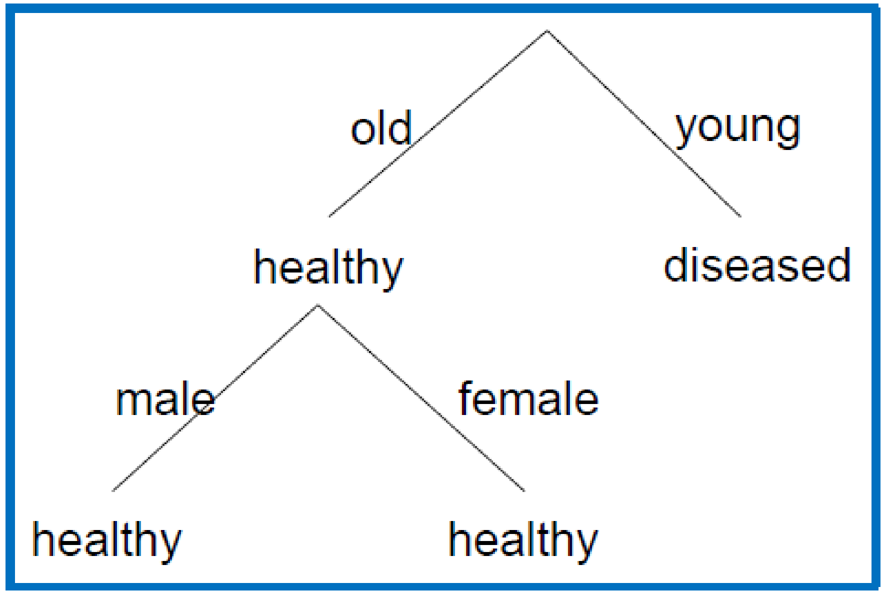
$\rightarrow$ For regression: average the results

$\rightarrow$ For classification: majority vote

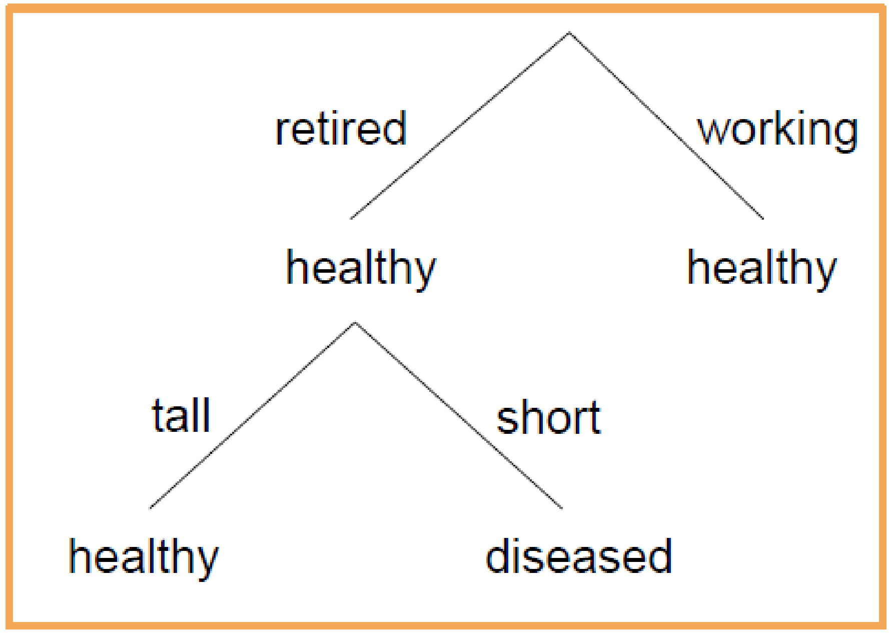
Tree 1



Tree 2



Tree 3



New sample:

old, retired, male, short

Tree predictions:

diseased, healthy, diseased

Majority rule:

diseased

Training the algorithm

- For some number of trees *T*:
- Sample *N* cases at random with replacement to create a subset of the data. The subset should be about 66% of the total set.
- At each node:
    - For some number *m* (see below), *m* predictor variables are selected at random from all the predictor variables.
    - The predictor variable that provides the best split, according to some objective function, is used to do a binary split on that node.
    - At the next node, choose another *m* variables at random from all predictor variables and do the same.
- Depending upon the value of *m*, there are three slightly different systems:
- Random splitter selection: *m* =1
- Breiman's bagger: *m* = total number of predictor variables
- Random forest: *m* << number of predictor variables. Breiman suggests three possible values for m: ½√*m*, √*m*, and 2√*m*

Running a Random Forest

- When a new input is entered into the system, it is run down all of the trees. The result may either be an average or weighted average of all of the terminal nodes that are reached, or, in the case of categorical variables, a voting majority.

Note that:

- With a large number of predictors, the eligible predictor set will be quite different from node to node.
- The greater the inter-tree correlation, the greater the random forest error rate, so one pressure on the model is to have the trees as uncorrelated as possible.
- As *m* goes down, both inter-tree correlation and the strength of individual trees go down. So some optimal value of *m* must be discovered.

Differences to standard tree

- Train each tree on Bootstrap **Resample** of data (Bootstrap resample of data set with N samples: Make new data set by drawing **with Replacement N samples**; i.e., some samples will probably occur multiple times in new data set)
- For each split, consider only m randomly selected variables
- Don’t prune
- Fit B trees in such a way and use average or majority voting to aggregate results

Random Forests Tuning

- The inventors make the following recommendations:

$\rightarrow$ For classification, the default value for m is √p and the minimum node size is one.

$\rightarrow$ For regression, the default value for m is p/3 and the minimum node size is five.

- In practice the best values for these parameters will depend on the problem, and they should be treated as tuning parameters.
- Like with Bagging, we can use OOB and therefore RF can be fit in one sequence, with cross-validation being performed along the way. Once the OOB error stabilizes, the training can be terminated.

Why Random Forests works:

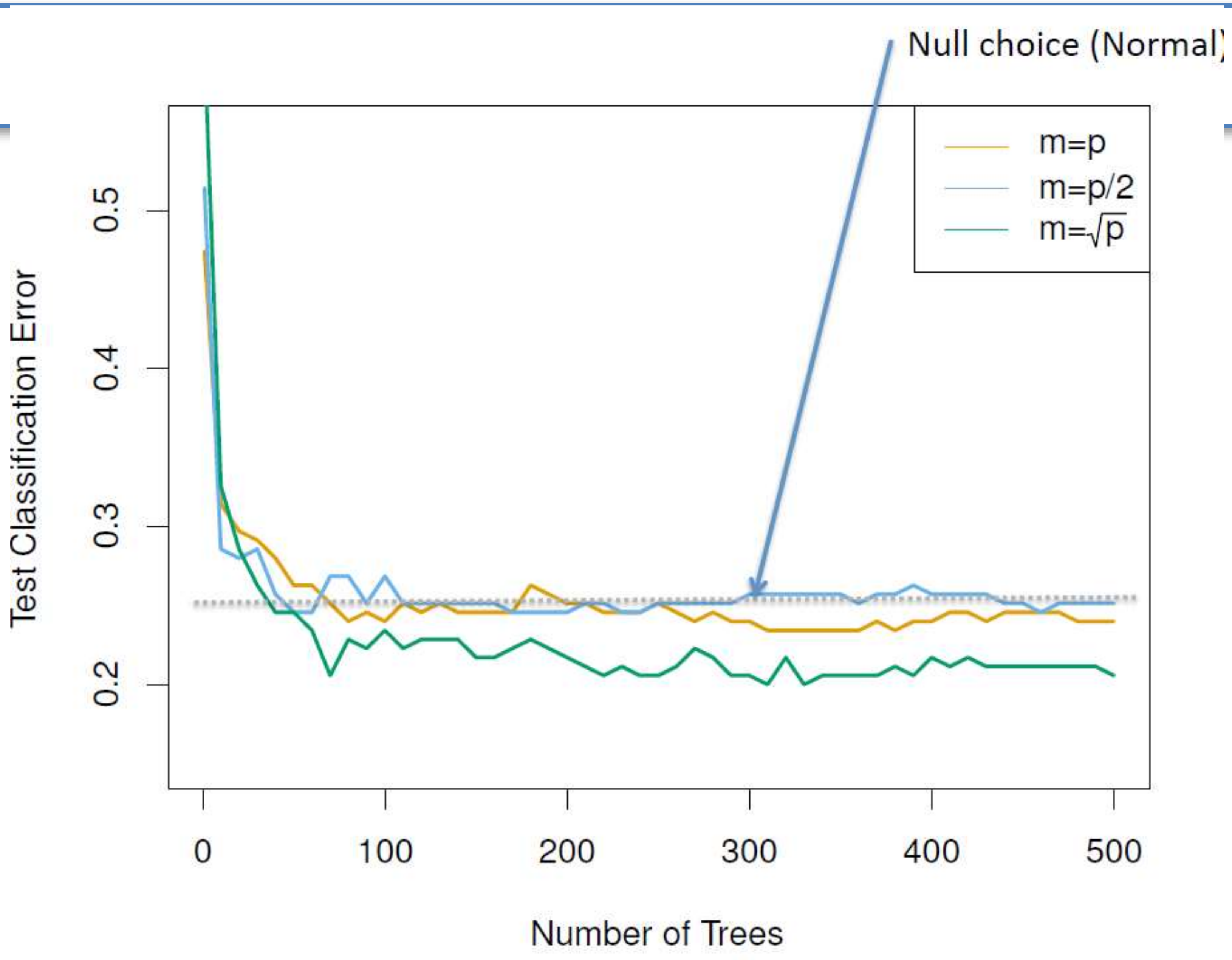
- Mean Squared Error = Variance + Bias^2
- If trees are sufficiently deep, they have very small bias
- How could we improve the variance over that of a single tree?
- The variance is $\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$. Hence, if the number of trees increases, the variance decreases.

Advantages of Random Forest

- No need for pruning trees
- Accuracy and variable importance generated automatically
- Overfitting is not a problem
- Not very sensitive to outliers in training data
- Easy to set parameters
- Good performance

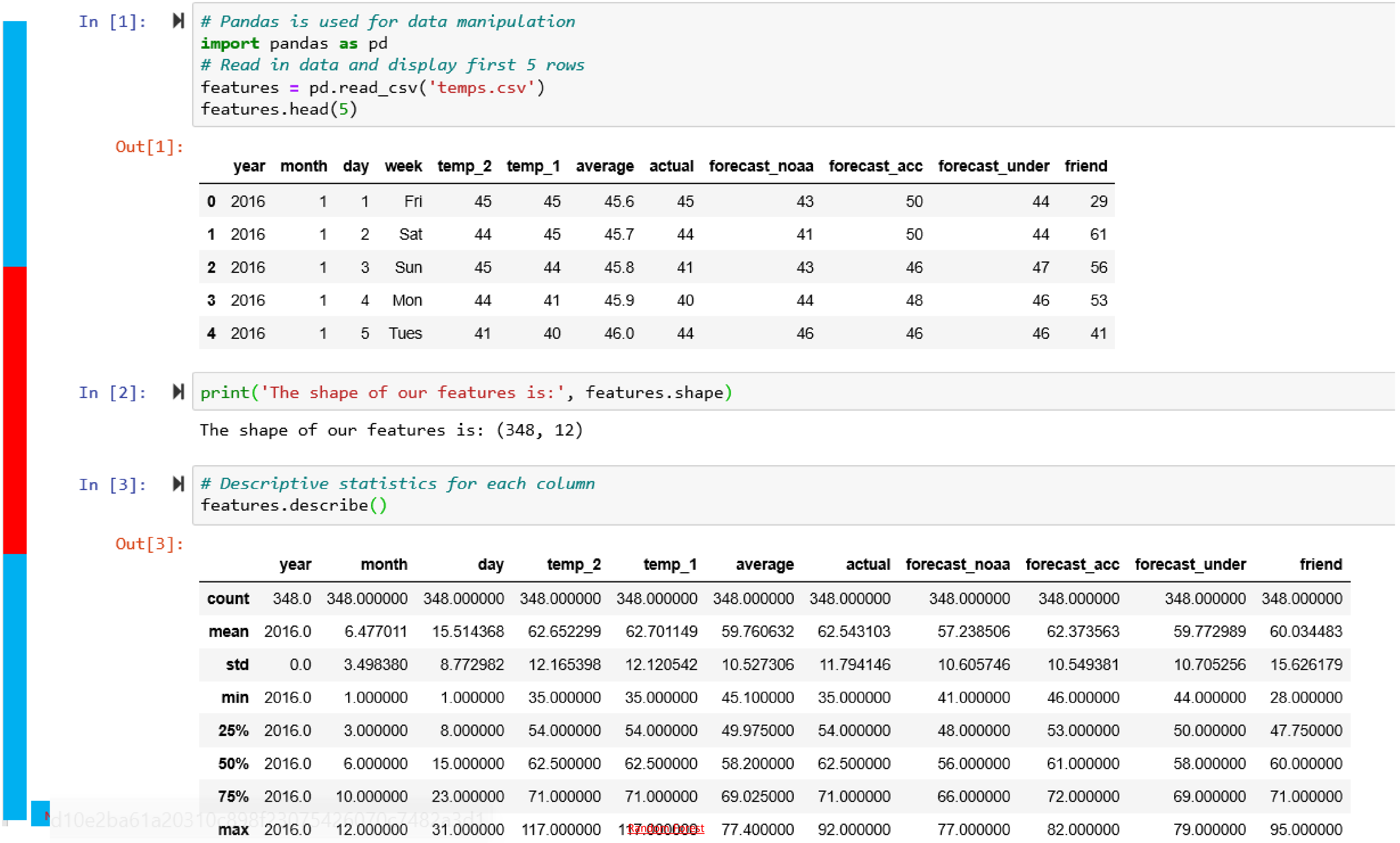
Example

- 4,718 genes measured on tissue samples from 349 patients.
- Each gene has different expression
- Each of the patient samples has a qualitative label with 15 different levels: either normal or 1 of 14 different types of cancer.
- Use random forests to predict cancer type based on the 500 genes that have the largest variance in the training set.


Null choice (Normal)
m=p
m=p/2
m=√p
Test Classification Error
0.2
0.3
0.4
0.5
0
100
200
300
400
500
Number of Trees

Roadmap:

1. **State the question and determine required data**
2. **Acquire the data in an accessible format**
3. **Identify and correct missing data points/anomalies as required**
4. **Prepare the data for the machine learning model**
5. **Establish a baseline model that you aim to exceed**
6. **Train the model on the training data**
7. **Make predictions on the test data**
8. **Compare predictions to the known test set targets and calculate performance metrics**
9. **If performance is not satisfactory, adjust the model, acquire more data, or try a different modeling technique**



In [1]:
```python
# Pandas is used for data manipulation
import pandas as pd
# Read in data and display first 5 rows
features = pd.read_csv('temps.csv')
features.head(5)
```

Out[1]:

	year	month	day	week	temp_2	temp_1	average	actual	forecast_noaa	forecast_acc	forecast_under	friend
0	2016	1	1	Fri	45	45	45.6	45	43	50	44	29
1	2016	1	2	Sat	44	45	45.7	44	41	50	44	61
2	2016	1	3	Sun	45	44	45.8	41	43	46	47	56
3	2016	1	4	Mon	44	41	45.9	40	44	48	46	53
4	2016	1	5	Tues	41	40	46.0	44	46	46	46	41

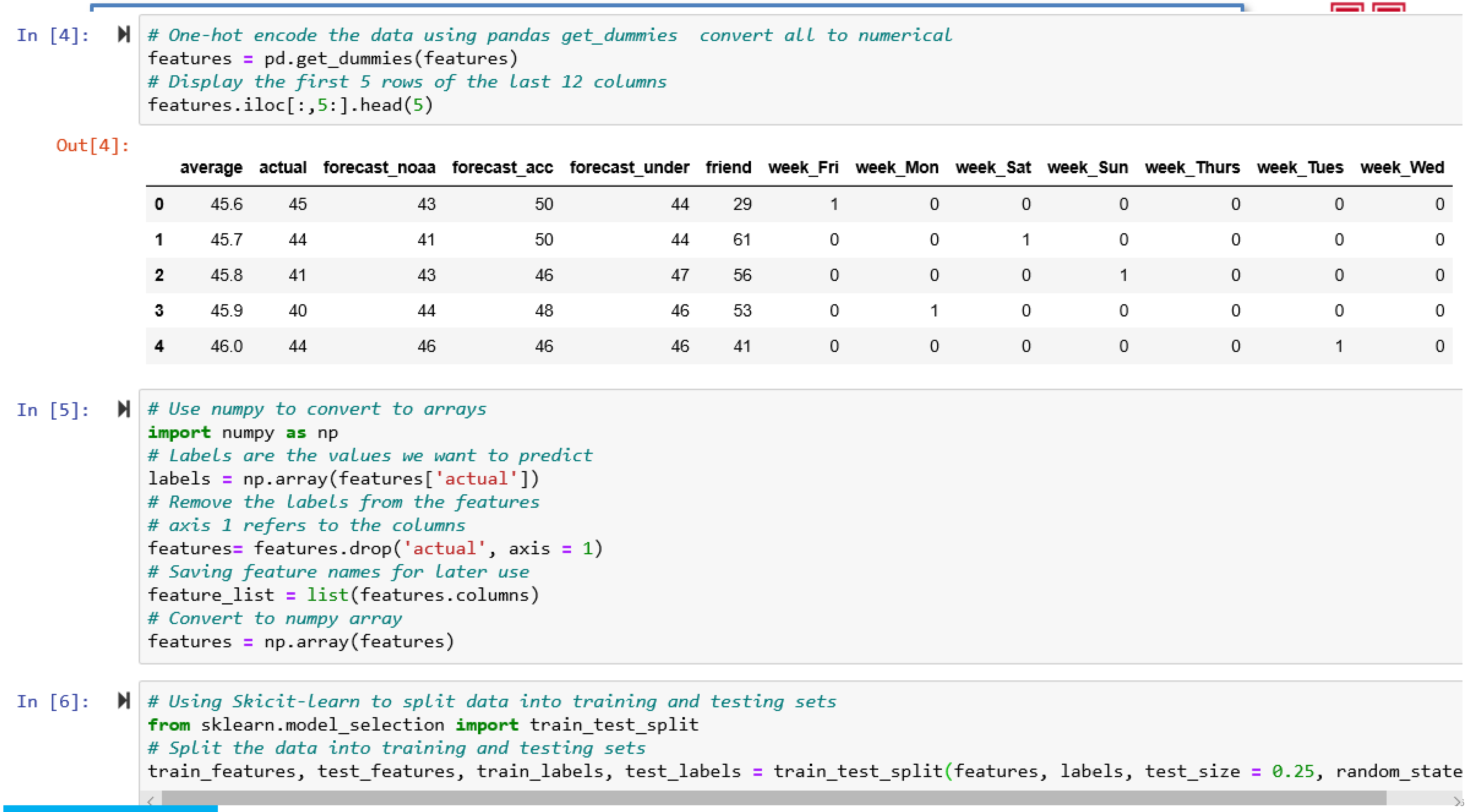
In [2]:
```python
print('The shape of our features is:', features.shape)
```

```
The shape of our features is: (348, 12)
```

In [3]:
```python
# Descriptive statistics for each column
features.describe()
```

Out[3]:

	year	month	day	temp_2	temp_1	average	actual	forecast_noaa	forecast_acc	forecast_under	friend
count	348.0	348.000000	348.000000	348.000000	348.000000	348.000000	348.000000	348.000000	348.000000	348.000000	348.000000
mean	2016.0	6.477011	15.514368	62.652299	62.701149	59.760632	62.543103	57.238506	62.373563	59.772989	60.034483
std	0.0	3.498380	8.772982	12.165398	12.120542	10.527306	11.794146	10.605746	10.549381	10.705256	15.626179
min	2016.0	1.000000	1.000000	35.000000	35.000000	45.100000	35.000000	41.000000	46.000000	44.000000	28.000000
25%	2016.0	3.000000	8.000000	54.000000	54.000000	49.975000	54.000000	48.000000	53.000000	50.000000	47.750000
50%	2016.0	6.000000	15.000000	62.500000	62.500000	58.200000	62.500000	56.000000	61.000000	58.000000	60.000000
75%	2016.0	10.000000	23.000000	71.000000	71.000000	69.025000	71.000000	66.000000	72.000000	69.000000	71.000000
max	2016.0	12.000000	31.000000	117.000000	117.000000	77.400000	92.000000	77.000000	82.000000	79.000000	95.000000

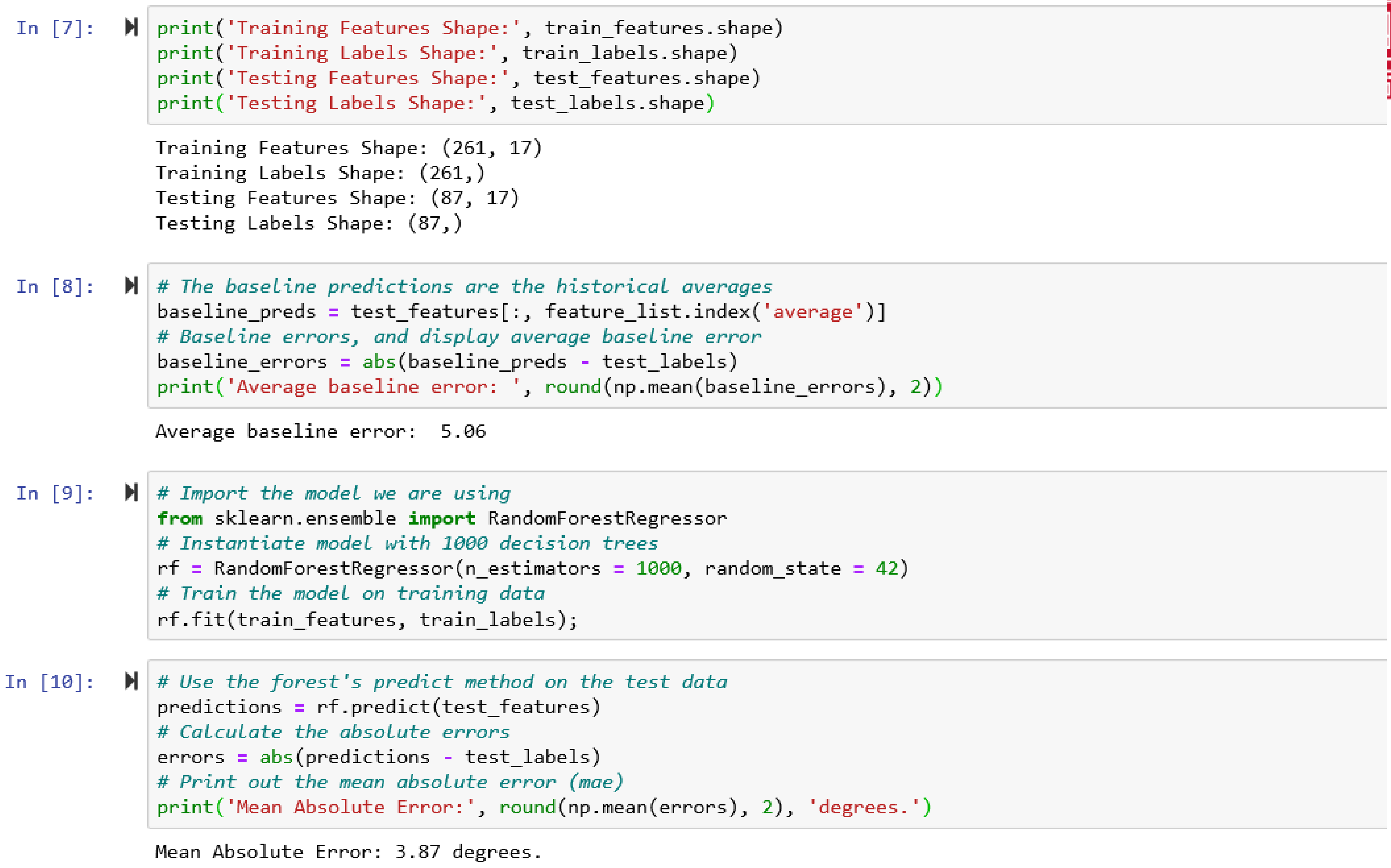


In [4]:
```python
# One-hot encode the data using pandas get_dummies  convert all to numerical
features = pd.get_dummies(features)
# Display the first 5 rows of the last 12 columns
features.iloc[:,5:].head(5)
```

Out[4]:
```
   average  actual  forecast_noaa  forecast_acc  forecast_under  friend  week_Fri  week_Mon  week_Sat  week_Sun  week_Thurs  week_Tues  week_Wed
0     45.6      45             43            50              44      29         1         0         0         0           0          0         0
1     45.7      44             41            50              44      61         0         0         1         0           0          0         0
2     45.8      41             43            46              47      56         0         0         0         1           0          0         0
3     45.9      40             44            48              46      53         0         1         0         0           0          0         0
4     46.0      44             46            46              46      41         0         0         0         0           0          1         0
```

In [5]:
```python
# Use numpy to convert to arrays
import numpy as np
# Labels are the values we want to predict
labels = np.array(features['actual'])
# Remove the labels from the features
# axis 1 refers to the columns
features= features.drop('actual', axis = 1)
# Saving feature names for later use
feature_list = list(features.columns)
# Convert to numpy array
features = np.array(features)
```

In [6]:
```python
# Using Skicit-learn to split data into training and testing sets
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets
train_features, test_features, train_labels, test_labels = train_test_split(features, labels, test_size = 0.25, random_state
```



```
In [7]:  print('Training Features Shape:', train_features.shape)
          print('Training Labels Shape:', train_labels.shape)
          print('Testing Features Shape:', test_features.shape)
          print('Testing Labels Shape:', test_labels.shape)

          Training Features Shape: (261, 17)
          Training Labels Shape: (261,)
          Testing Features Shape: (87, 17)
          Testing Labels Shape: (87,)
```

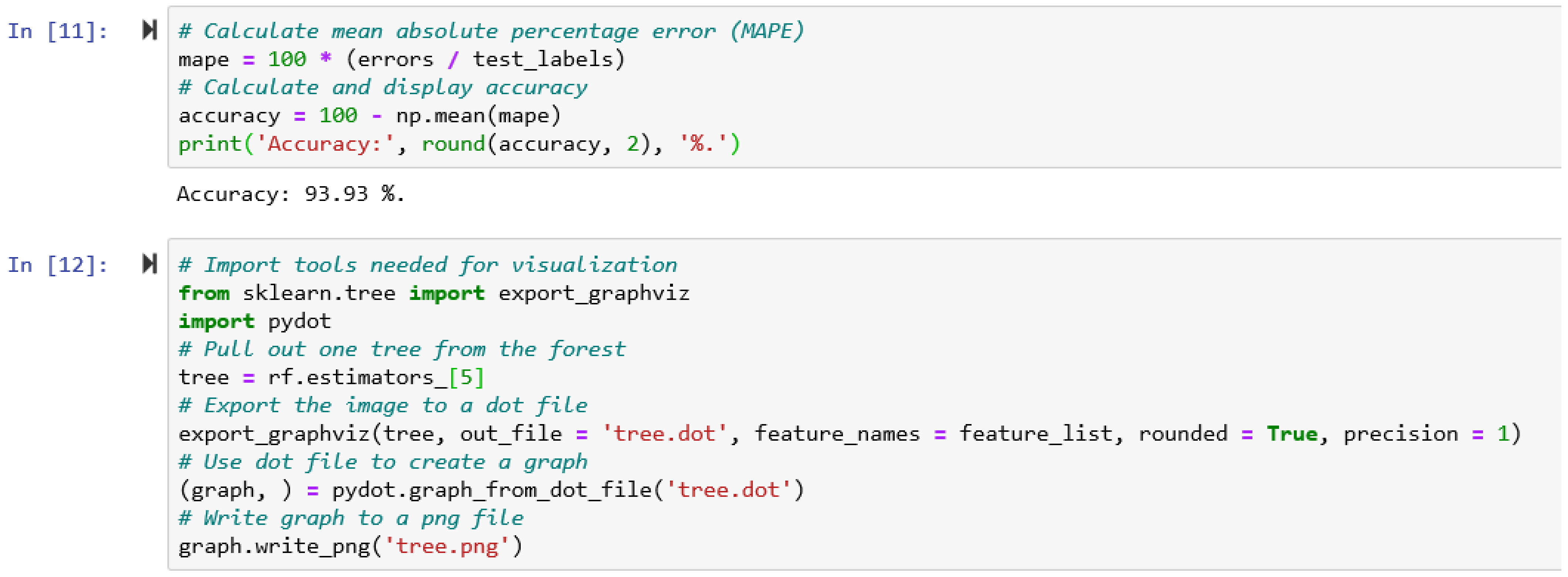
```
In [8]:  # The baseline predictions are the historical averages
          baseline_preds = test_features[:, feature_list.index('average')]
          # Baseline errors, and display average baseline error
          baseline_errors = abs(baseline_preds - test_labels)
          print('Average baseline error: ', round(np.mean(baseline_errors), 2))

          Average baseline error:  5.06
```

```
In [9]:  # Import the model we are using
          from sklearn.ensemble import RandomForestRegressor
          # Instantiate model with 1000 decision trees
          rf = RandomForestRegressor(n_estimators = 1000, random_state = 42)
          # Train the model on training data
          rf.fit(train_features, train_labels);
```

```
In [10]: # Use the forest's predict method on the test data
          predictions = rf.predict(test_features)
          # Calculate the absolute errors
          errors = abs(predictions - test_labels)
          # Print out the mean absolute error (mae)
          print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')

          Mean Absolute Error: 3.87 degrees.
```



```
In [11]:  # Calculate mean absolute percentage error (MAPE)
          mape = 100 * (errors / test_labels)
          # Calculate and display accuracy
          accuracy = 100 - np.mean(mape)
          print('Accuracy:', round(accuracy, 2), '%.')

          Accuracy: 93.93 %.

In [12]:  # Import tools needed for visualization
          from sklearn.tree import export_graphviz
          import pydot
          # Pull out one tree from the forest
          tree = rf.estimators_[5]
          # Export the image to a dot file
          export_graphviz(tree, out_file = 'tree.dot', feature_names = feature_list, rounded = True, precision = 1)
          # Use dot file to create a graph
          (graph, ) = pydot.graph_from_dot_file('tree.dot')
          # Write graph to a png file
          graph.write_png('tree.png')
```