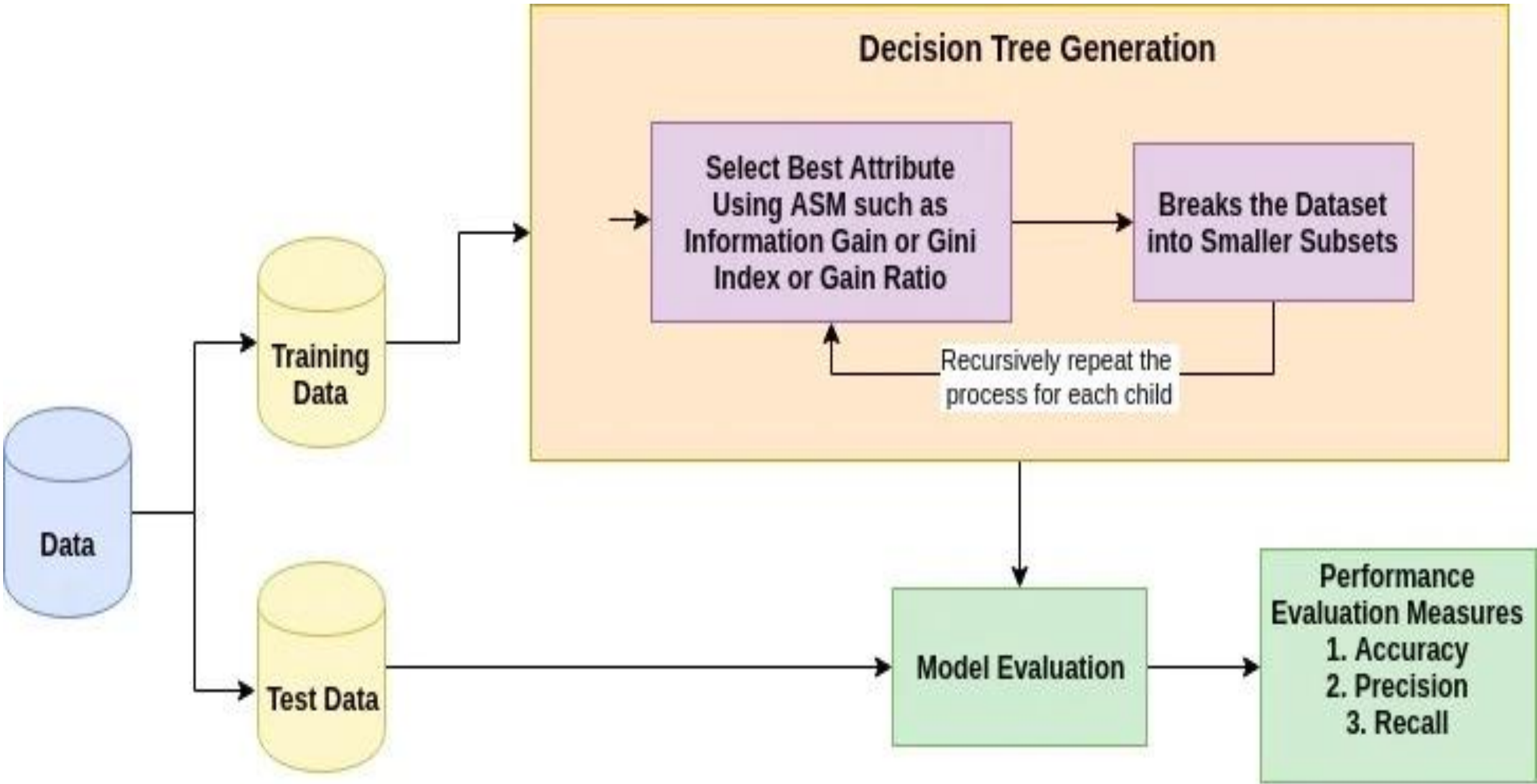




name	age	apple_pie?	potato_salad?	sushi?	midwest?
Jeff	32	0	1	1	1
Pete	25	1	1	0	1
Anne	33	1	1	0	1
Natalie	26	0	0	1	0
Stella	30	1	1	1	1
Rob	25	1	0	0	1
Joe	42	1	1	0	1
Jim	38	1	1	0	1
Lisa	36	1	1	0	0
Sarah	29	1	0	1	0
David	35	1	0	0	1
Eric	28	1	1	1	0
Mike	20	0	1	0	1
Karen	38	1	0	0	1
Megan	31	0	0	1	0

name	age	apple_pie?	potato_salad?	sushi?	midwest?
Jeff	32	0	1	1	1
Pete	25	1	1	0	1
Anne	33	1	1	0	1
Natalie	26	0	0	1	0
Stella	30	1	1	1	1
Rob	25	1	0	0	1
Joe	42	1	1	0	1
Jim	38	1	1	0	1
Lisa	36	1	1	0	0
Sarah	29	1	0	1	0
David	35	1	0	0	1
Eric	28	1	1	1	0
Mike	20	0	1	0	1
Karen	38	1	0	0	1
Megan	31	0	0	1	0

In data science, entropy is used as a way to measure how “mixed” a column is.

Specifically, entropy is used to measure disorder.

$$- \sum_{i=1}^c P(x_i) \log_b P(x_i)$$

name	age	apple_pie?	potato_salad?	sushi?	midwest?
Jeff	32	0	1	1	1
Pete	25	1	1	0	1
Anne	33	1	1	0	1
Natalie	26	0	0	1	0
Stella	30	1	1	1	1
Rob	25	1	0	0	1
Joe	42	1	1	0	1
Jim	38	1	1	0	1
Lisa	36	1	1	0	0
Sarah	29	1	0	1	0
David	35	1	0	0	1
Eric	28	1	1	1	0
Mike	20	0	1	0	1
Karen	38	1	0	0	1
Megan	31	0	0	1	0

Entropy of Midwest

$$-\sum_{i=1}^c P(x_i) \log_b P(x_i)$$

$$\begin{aligned}
&-(10/15 \cdot \log_2(10/15) + 5/15 \cdot \log_2(5/15)) \\
&\quad -(-.389975 + -.528308) \\
&\quad \quad -(-.918278) \\
&\quad \quad .918278
\end{aligned}$$

$$\begin{aligned}
& -\left(\frac{10}{15} \cdot \log_2\left(\frac{10}{15}\right) + \frac{5}{15} \cdot \log_2\left(\frac{5}{15}\right)\right) \\
& \quad -(-.389975 + -.528308) \\
& \quad \quad -(-.918278) \\
& \quad \quad .918278
\end{aligned}$$

name	age	apple_pie?	potato_salad?	sushi?	midwest?
Jeff	32	0	1	1	1
Pete	25	1	1	0	1
Anne	33	1	1	0	1
Natalie	26	0	0	1	0
Stella	30	1	1	1	1
Rob	25	1	0	0	1
Joe	42	1	1	0	1
Jim	38	1	1	0	1
Lisa	36	1	1	0	0
Sarah	29	1	0	1	0
David	35	1	0	0	1
Eric	28	1	1	1	0
Mike	20	0	1	0	1
Karen	38	1	0	0	1
Megan	31	0	0	1	0

In information theory, a bit is thought of as a binary number representing 0 for no information and 1 for a full bit of information. Suppose there's an equal probability of it raining tomorrow (1) or not raining(0). If I tell you that it will rain tomorrow, I've given you one bit of information.

We can also think of entropy as information. Suppose we have a loaded six-sided die which always lands on (3). Each time we roll the die, we know upfront that the result will be (3). We gain no new information by rolling the die, so entropy is 0. On the other hand, if the die is fair and we roll a (3) there was a 1/6 chance in rolling the (3). Now we have gained information. Thus, rolling the die gives us one bit of information — which side the number landed on.

$$\begin{aligned}
& - (10/15 \cdot \log_2(10/15) + 5/15 \cdot \log_2(5/15)) \\
& \quad - (-.389975 + -.528308) \\
& \quad \quad - (-.918278) \\
& \quad \quad .918278
\end{aligned}$$

name	age	apple_pie?	potato_salad?	sushi?	midwest?
Jeff	32	0	1	1	1
Pete	25	1	1	0	1
Anne	33	1	1	0	1
Natalie	26	0	0	1	0
Stella	30	1	1	1	1
Rob	25	1	0	0	1
Joe	42	1	1	0	1
Jim	38	1	1	0	1
Lisa	36	1	1	0	0
Sarah	29	1	0	1	0
David	35	1	0	0	1
Eric	28	1	1	1	0
Mike	20	0	1	0	1
Karen	38	1	0	0	1
Megan	31	0	0	1	0

We get less than one “bit” of information — only .918278 — because there are more (1)s in the “midwest?” column than (0)s.

This means that if we were predicting a new value, we could guess that the answer is (1) and be right more often than wrong (because there’s a 2/3 probability of the answer being 1). Due to this prior knowledge, we gain less than a full “bit” of information when we observe a new value.

$$\begin{aligned}
& - (10/15 \cdot \log_2(10/15) + 5/15 \cdot \log_2(5/15)) \\
& - (-.389975 + -.528308) \\
& - (-.918278) \\
& .918278
\end{aligned}$$

Our goal is to find the best variable(s)/column(s) to split on when building a decision tree.

Eventually, we want to keep splitting the variables/columns until our mixed target column is no longer mixed.

name	age	apple_pie?	potato_salad?	sushi?	midwest?
Jeff	32	0	1	1	1
Pete	25	1	1	0	1
Anne	33	1	1	0	1
Natalie	26	0	0	1	0
Stella	30	1	1	1	1
Rob	25	1	0	0	1
Joe	42	1	1	0	1
Jim	38	1	1	0	1
Lisa	36	1	1	0	0
Sarah	29	1	0	1	0
David	35	1	0	0	1
Eric	28	1	1	1	0
Mike	20	0	1	0	1
Karen	38	1	0	0	1
Megan	31	0	0	1	0

$$IG(T, A) = Entropy(T) - \sum_{V \in A} \frac{|T_V|}{T} \cdot Entropy(T_V)$$

name	age	apple_pie?	potato_salad?	sushi?	midwest?
Jeff	32	0	1	1	1
Pete	25	1	1	0	1
Anne	33	1	1	0	1
Natalie	26	0	0	1	0
Stella	30	1	1	1	1
Rob	25	1	0	0	1
Joe	42	1	1	0	1
Jim	38	1	1	0	1
Lisa	36	1	1	0	0
Sarah	29	1	0	1	0
David	35	1	0	0	1
Eric	28	1	1	1	0
Mike	20	0	1	0	1
Karen	38	1	0	0	1
Megan	31	0	0	1	0

$$IG(T, A) = Entropy(T) - \sum_{v \in A} \frac{|T_v|}{T} \cdot Entropy(T_v)$$

- T = Target, our “midwest?” column
- A = the variable(column) we are testing, “potato_salad?”
- v = each value in A, each value in the “potato_salad?” column

1. First, we’ll calculate the original entropy for (T) before the split ,
.918278
2. Then, for each unique value (v) in variable (A), we compute the number of rows in which (A) takes on the value (v), and divide it by the total number of rows. For the “potato_salad?” column we get **9/15** for the unique value of (1) and **6/15** for the unique value of (0).
3. Next, we multiply the results by the entropy of the rows where (A) is (v). For the left split(split on 1 for “potato_salad?”) we get **9/15 * .764204**. For the right side of the split (split on 0 for “potato_salad?”) we get **6/15 * 1**.
4. We add all of these subset products together, **9/15*.764204 + 6/15 = .8585224**.
5. We then subtract from the overall entropy to get information gain, **.918278 -.8585224 = .059754**

name	age	apple_pie?	potato_salad?	sushi?	midwest?
Jeff	32	0	1	1	1
Pete	25	1	1	0	1
Anne	33	1	1	0	1
Natalie	26	0	0	1	0
Stella	30	1	1	1	1
Rob	25	1	0	0	1
Joe	42	1	1	0	1
Jim	38	1	1	0	1
Lisa	36	1	1	0	0
Sarah	29	1	0	1	0
David	35	1	0	0	1
Eric	28	1	1	1	0
Mike	20	0	1	0	1
Karen	38	1	0	0	1
Megan	31	0	0	1	0

$$IG(T, A) = Entropy(T) - \sum_{v \in A} \frac{|T_v|}{T} \cdot Entropy(T_v)$$

We end up with .059754, which means that we gain .059754 bits of information by splitting our data set on the “potato_salad?” variable/column.

Our information gain is low, but it’s still positive which is because we lowered the entropy on the left side of the split.

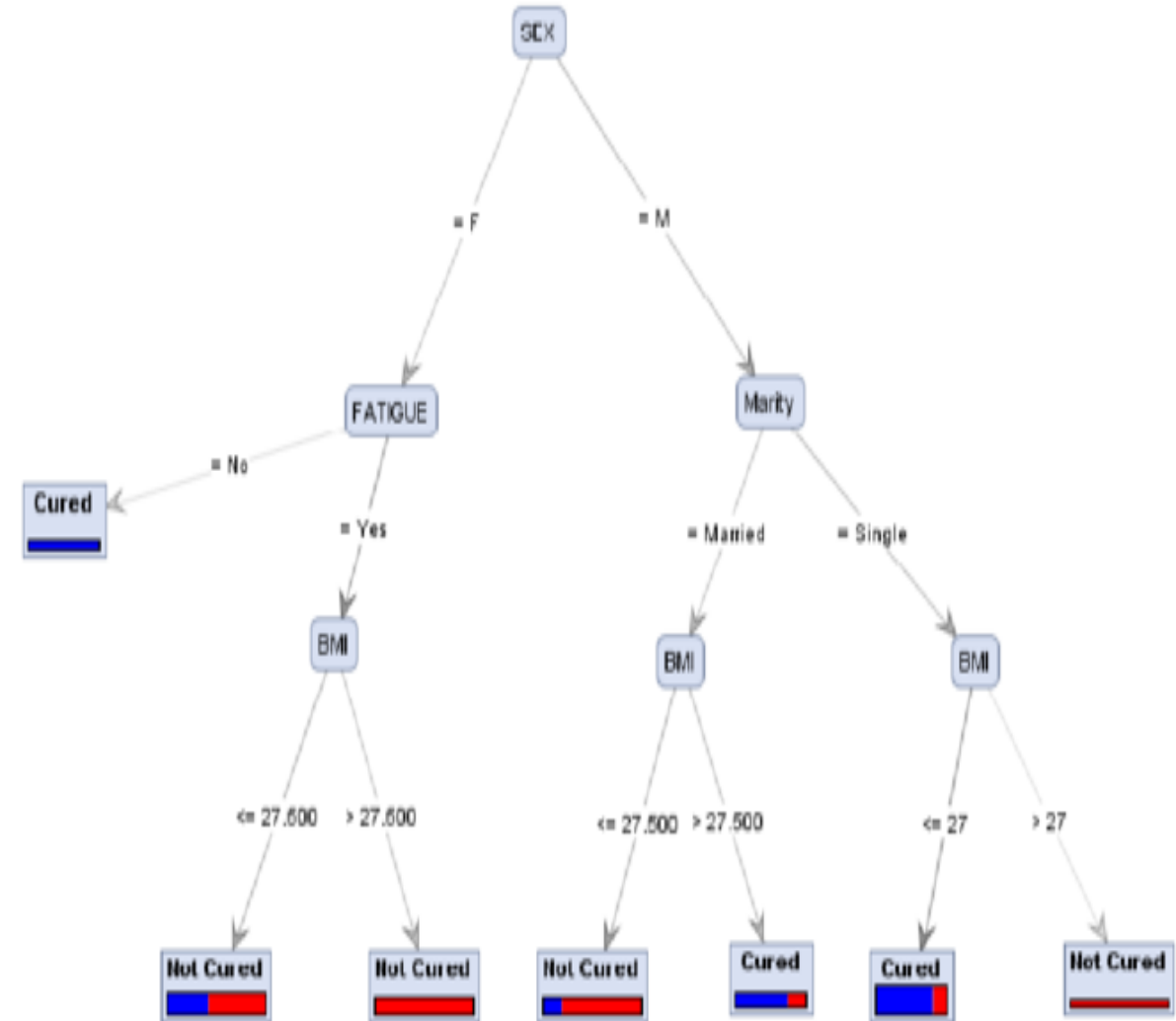
Gini index or Gini impurity measures the degree or probability of a particular variable being wrongly classified when it is randomly chosen.

If all the elements belong to a single class, then it can be called pure.

The degree of Gini index varies between 0 and 1, where 0 denotes that all elements belong to a certain class or if there exists only one class, and 1 denotes that the elements are randomly distributed across various classes.

A Gini Index of 0.5 denotes equally distributed elements into some classes.

$$\text{Gini} = 1 - \sum_{i=1}^n (p_i)^2$$



Past Trend	Open Interest	Trading Volume	Return
Positive	Low	High	Up
Negative	High	Low	Down
Positive	Low	High	Up
Positive	High	High	Up
Negative	Low	High	Down
Positive	Low	Low	Down
Negative	High	High	Down
Negative	Low	High	Down
Positive	Low	Low	Down
Positive	High	High	Up

Past Trend	Open Interest	Trading Volume	Return
Positive	Low	High	Up
Negative	High	Low	Down
Positive	Low	High	Up
Positive	High	High	Up
Negative	Low	High	Down
Positive	Low	Low	Down
Negative	High	High	Down
Negative	Low	High	Down
Positive	Low	Low	Down
Positive	High	High	Up

$P(\text{Past Trend}=\text{Positive}): 6/10$

$P(\text{Past Trend}=\text{Negative}): 4/10$

If (Past Trend = Positive & Return = Up), probability = $4/6$

If (Past Trend = Positive & Return = Down), probability = $2/6$

Gini index = $1 - ((4/6)^2 + (2/6)^2) = 0.45$

If (Past Trend = Negative & Return = Up), probability = 0

If (Past Trend = Negative & Return = Down), probability = $4/4$

Gini index = $1 - ((0)^2 + (4/4)^2) = 0$

Weighted sum of the Gini Indices can be calculated as follows:

Gini Index for Past Trend = $(6/10)0.45 + (4/10)0 = 0.27$

Past Trend	Open Interest	Trading Volume	Return
Positive	Low	High	Up
Negative	High	Low	Down
Positive	Low	High	Up
Positive	High	High	Up
Negative	Low	High	Down
Positive	Low	Low	Down
Negative	High	High	Down
Negative	Low	High	Down
Positive	Low	Low	Down
Positive	High	High	Up

P(Open Interest=High): 4/10

P(Open Interest=Low): 6/10

If (Open Interest = High & Return = Up), probability = 2/4

If (Open Interest = High & Return = Down), probability = 2/4

Gini index = $1 - ((2/4)^2 + (2/4)^2) = 0.5$

If (Open Interest = Low & Return = Up), probability = 2/6

If (Open Interest = Low & Return = Down), probability = 4/6

Gini index = $1 - ((2/6)^2 + (4/6)^2) = 0.45$

Weighted sum of the Gini Indices can be calculated as follows:

Gini Index for Open Interest = $(4/10)0.5 + (6/10)0.45 = 0.47$

Past Trend	Open Interest	Trading Volume	Return
Positive	Low	High	Up
Negative	High	Low	Down
Positive	Low	High	Up
Positive	High	High	Up
Negative	Low	High	Down
Positive	Low	Low	Down
Negative	High	High	Down
Negative	Low	High	Down
Positive	Low	Low	Down
Positive	High	High	Up

$P(\text{Trading Volume}=\text{High}): 7/10$

$P(\text{Trading Volume}=\text{Low}): 3/10$

If (Trading Volume = High & Return = Up), probability = $4/7$

If (Trading Volume = High & Return = Down), probability = $3/7$

Gini index = $1 - ((4/7)^2 + (3/7)^2) = 0.49$

If (Trading Volume = Low & Return = Up), probability = 0

If (Trading Volume = Low & Return = Down), probability = $3/3$

Gini index = $1 - ((0)^2 + (1)^2) = 0$

Weighted sum of the Gini Indices can be calculated as follows:

Gini Index for Trading Volume = $(7/10)0.49 + (3/10)0 = 0.34$

Past Trend	Open Interest	Trading Volume	Return
Positive	Low	High	Up
Negative	High	Low	Down
Positive	Low	High	Up
Positive	High	High	Up
Negative	Low	High	Down
Positive	Low	Low	Down
Negative	High	High	Down
Negative	Low	High	Down
Positive	Low	Low	Down
Positive	High	High	Up

$P(\text{Trading Volume}=\text{High}): 7/10$

$P(\text{Trading Volume}=\text{Low}): 3/10$

If (Trading Volume = High & Return = Up), probability = $4/7$

If (Trading Volume = High & Return = Down), probability = $3/7$

Gini index = $1 - ((4/7)^2 + (3/7)^2) = 0.49$

If (Trading Volume = Low & Return = Up), probability = 0

If (Trading Volume = Low & Return = Down), probability = $3/3$

Gini index = $1 - ((0)^2 + (1)^2) = 0$

Weighted sum of the Gini Indices can be calculated as follows:

Gini Index for Trading Volume = $(7/10)0.49 + (3/10)0 = 0.34$

Past Trend	Open Interest	Trading Volume	Return
Positive	Low	High	Up
Positive	Low	High	Up
Positive	High	High	Up
Positive	Low	Low	Down
Positive	Low	Low	Down
Positive	High	High	Up

$P(\text{Open Interest}=\text{High}): 2/6$

$P(\text{Open Interest}=\text{Low}): 4/6$

If (Open Interest = High & Return = Up), probability = $2/2$

If (Open Interest = High & Return = Down), probability = 0

Gini index = $1 - (\text{sq}(2/2) + \text{sq}(0)) = 0$

If (Open Interest = Low & Return = Up), probability = $2/4$

If (Open Interest = Low & Return = Down), probability = $2/4$

Gini index = $1 - (\text{sq}(0) + \text{sq}(2/4)) = 0.50$

Weighted sum of the Gini Indices can be calculated as follows:

Gini Index for Open Interest = $(2/6)0 + (4/6)0.50 = 0.33$

Past Trend	Open Interest	Trading Volume	Return
Positive	Low	High	Up
Positive	Low	High	Up
Positive	High	High	Up
Positive	Low	Low	Down
Positive	Low	Low	Down
Positive	High	High	Up

$P(\text{Trading Volume}=\text{High}): 4/6$

$P(\text{Trading Volume}=\text{Low}): 2/6$

If (Trading Volume = High & Return = Up), probability = 4/4

If (Trading Volume = High & Return = Down), probability = 0

Gini index = $1 - (\text{sq}(4/4) + \text{sq}(0)) = 0$

If (Trading Volume = Low & Return = Up), probability = 0

If (Trading Volume = Low & Return = Down), probability = 2/2

Gini index = $1 - (\text{sq}(0) + \text{sq}(2/2)) = 0$

Weighted sum of the Gini Indices can be calculated as follows:

Gini Index for Trading Volume = $(4/6)0 + (2/6)0 = 0$

Gini Index, unlike information gain, isn't computationally intensive as it doesn't involve the logarithm function used to calculate entropy in information gain, which is why Gini Index is preferred over Information gain.

Gain Ratio

$$\textit{SplitInfo}_A(D) = - \sum_{j=1}^v \frac{|D_j|}{|D|} \times \log_2 \left(\frac{|D_j|}{|D|} \right)$$

Regression Trees

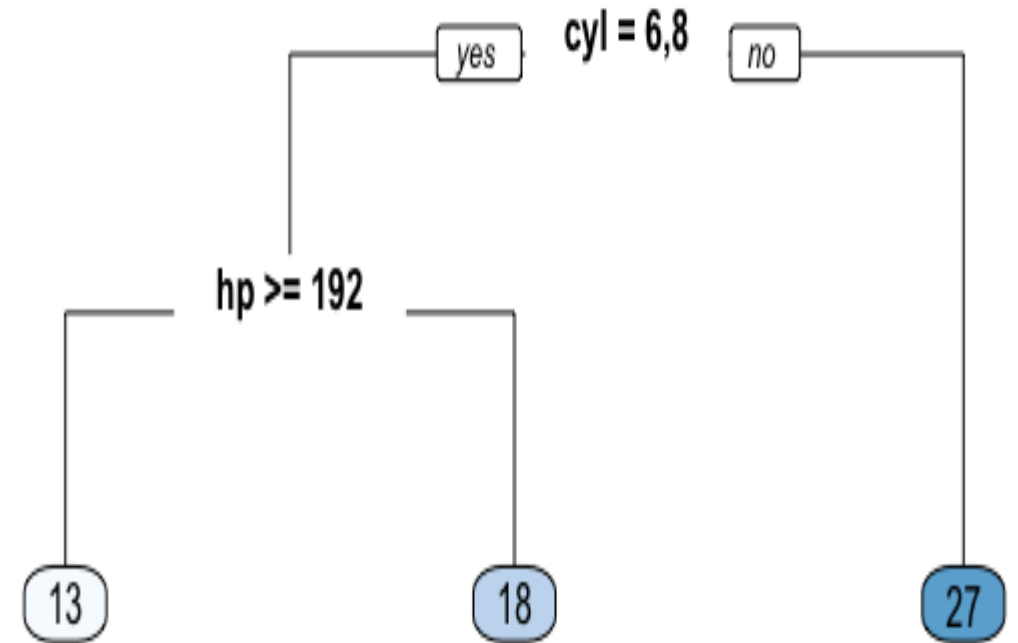
- Basic ***regression trees*** partition a data set into smaller groups and then fit a simple model (constant) for each subgroup.
- Unfortunately, a single tree model tends to be highly unstable and a poor predictor. However, by bootstrap aggregating (***bagging***) regression trees, this technique can become quite powerful and effective.
- This provides the fundamental basis of more complex tree-based models such as *random forests* and *gradient boosting machines*.

Regression Trees

- Classification and regression tree (CART) approach developed by Breiman et al. (1984)
- Basic regression trees partition a data set into smaller subgroups and then fit a simple constant for each observation in the subgroup.
- The partitioning is achieved by successive binary partitions (aka recursive partitioning) based on the different predictors.
- The constant to predict is based on the average response values for all observations that fall in that subgroup.

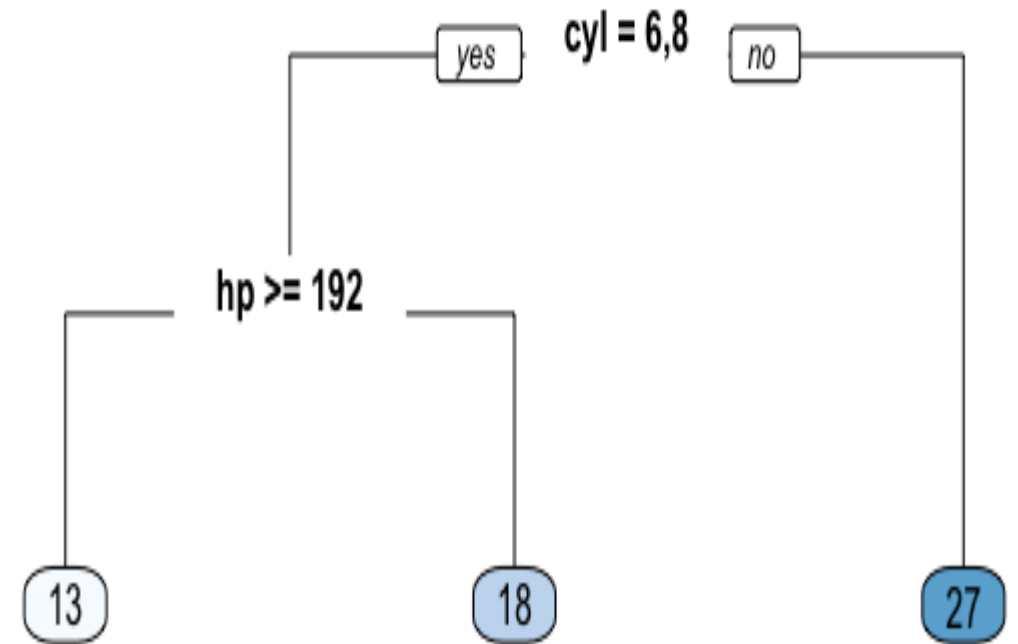
Regression Trees

- we want to predict the miles per gallon a car will average based on cylinders (cyl) and horsepower (hp).
- All cars that do not have 6 or 8 cylinders average 27 mpg.
- All cars that have 6 or 8 cylinders and have more than 192 hp average 13 mpg.



Regression Trees

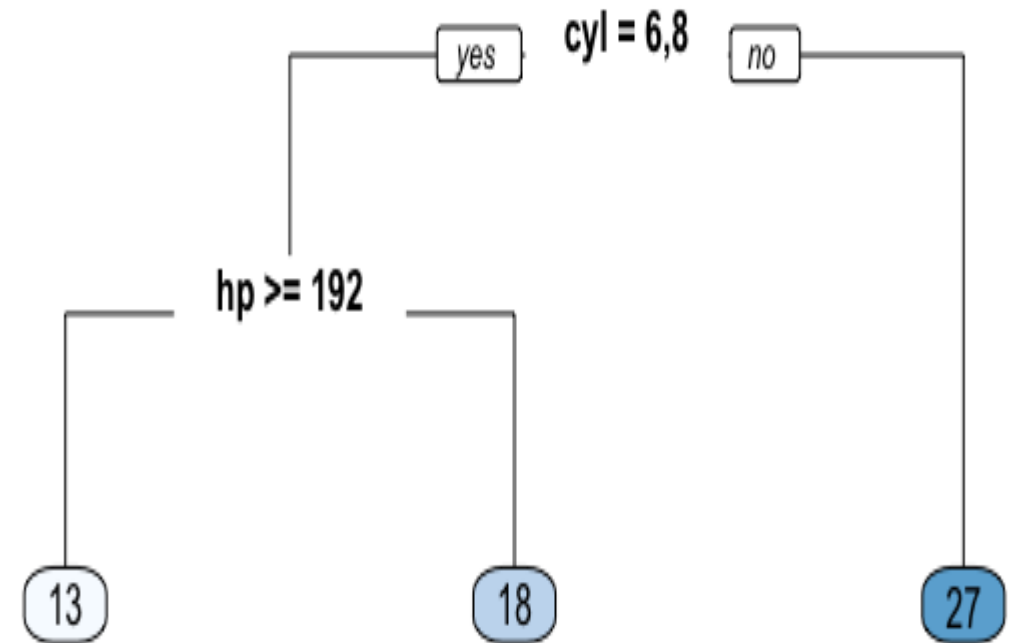
- This simple example can be generalized to state we have a continuous response variable Y and two inputs X_1 and X_2 .
- The recursive partitioning results in three regions (R_1, R_2, R_3) where the model predicts Y with a constant c_m for region R_m :



Regression Trees

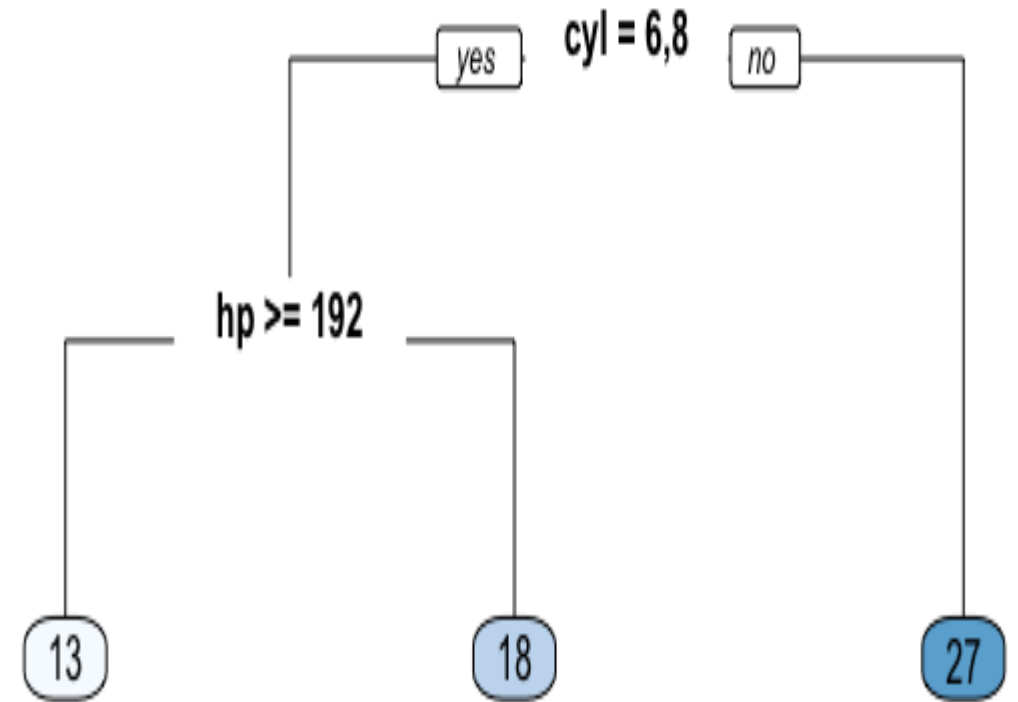
- The model begins with the entire data set, S , and searches every distinct value of every input variable to find the predictor and split value that partitions the data into two regions (R_1 and R_2) such that the overall sums of squares error are minimized:

$$\text{minimize} \left\{ SSE = \sum_{i \in R_1} (y_i - c_1)^2 + \sum_{i \in R_2} (y_i - c_2)^2 \right\}$$



Regression Trees

- Having found the best split, we partition the data into the two resulting regions and repeat the splitting process on each of the two regions. This process is continued until some stopping criterion is reached.
- What results is, typically, a very deep, complex tree that may produce good predictions on the training set, but is likely to overfit the data, leading to poor performance on unseen data.

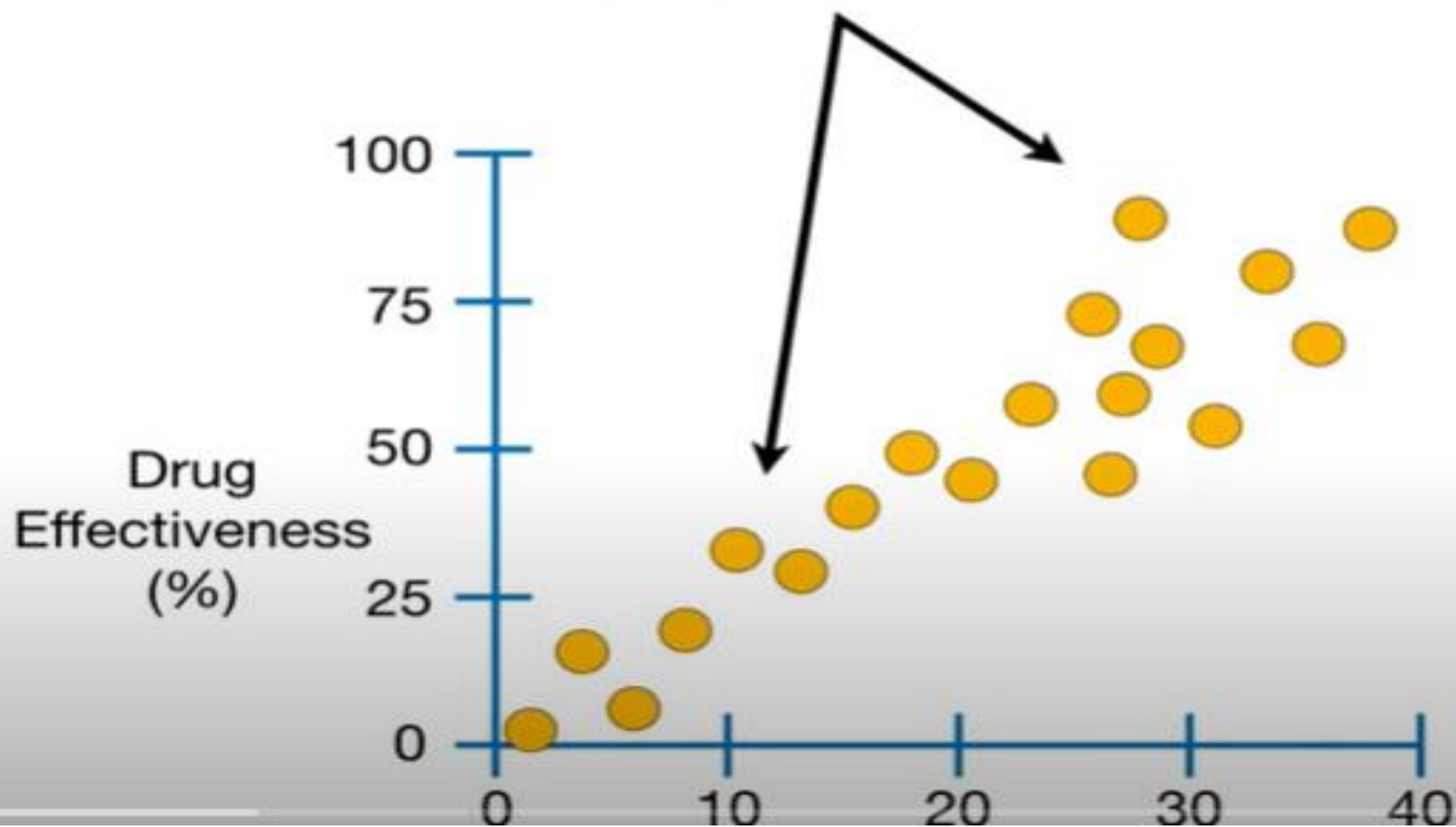


$$\text{minimize} \left\{ SSE = \sum_{i \in R_1} (y_i - c_1)^2 + \sum_{i \in R_2} (y_i - c_2)^2 \right\}$$

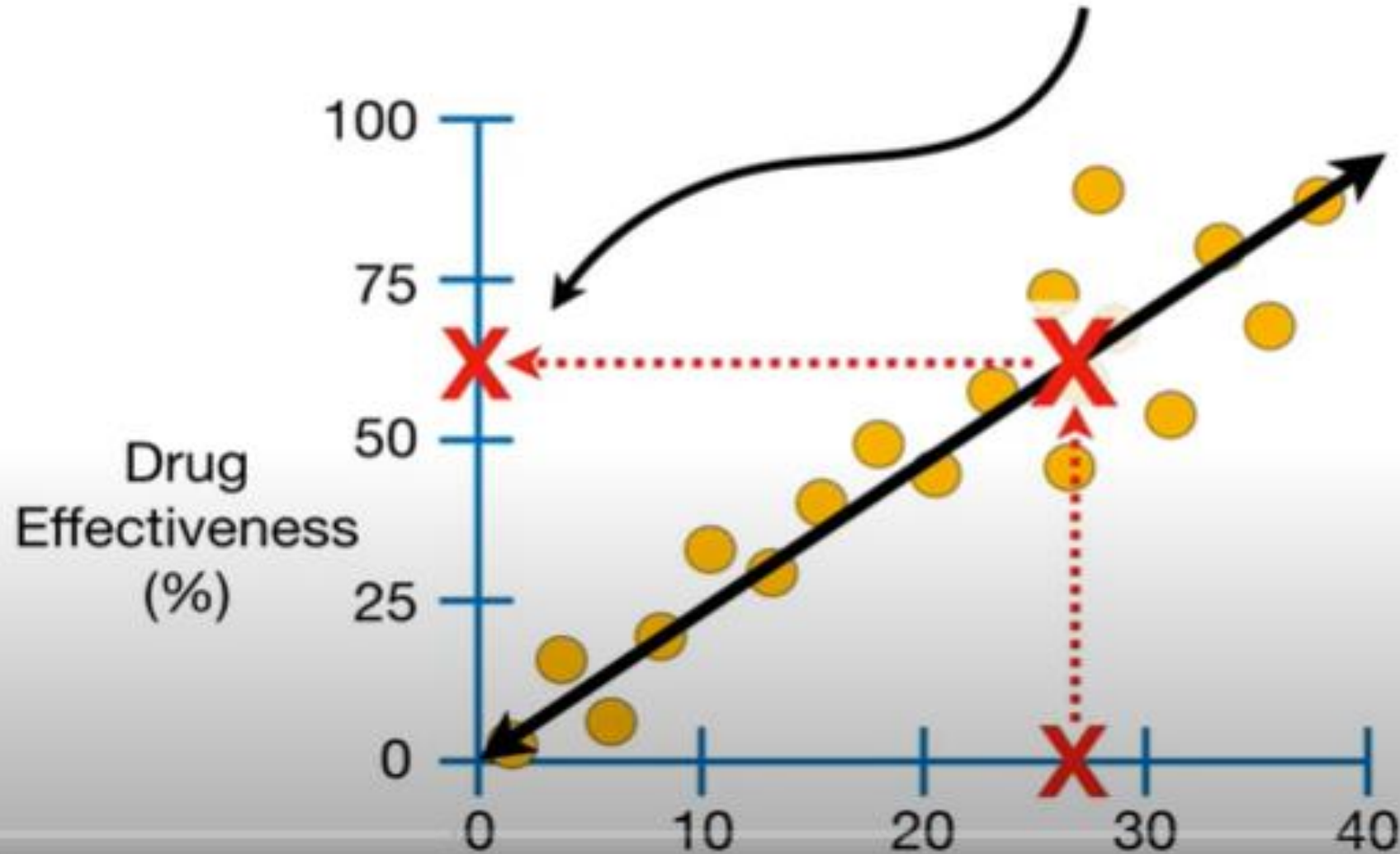
Example

- A new drug is developed but we don't know the optimal dosage to give to the patients
- We do a clinical trial with different dosages
- And measure the drug effectiveness

If the data looked like this...

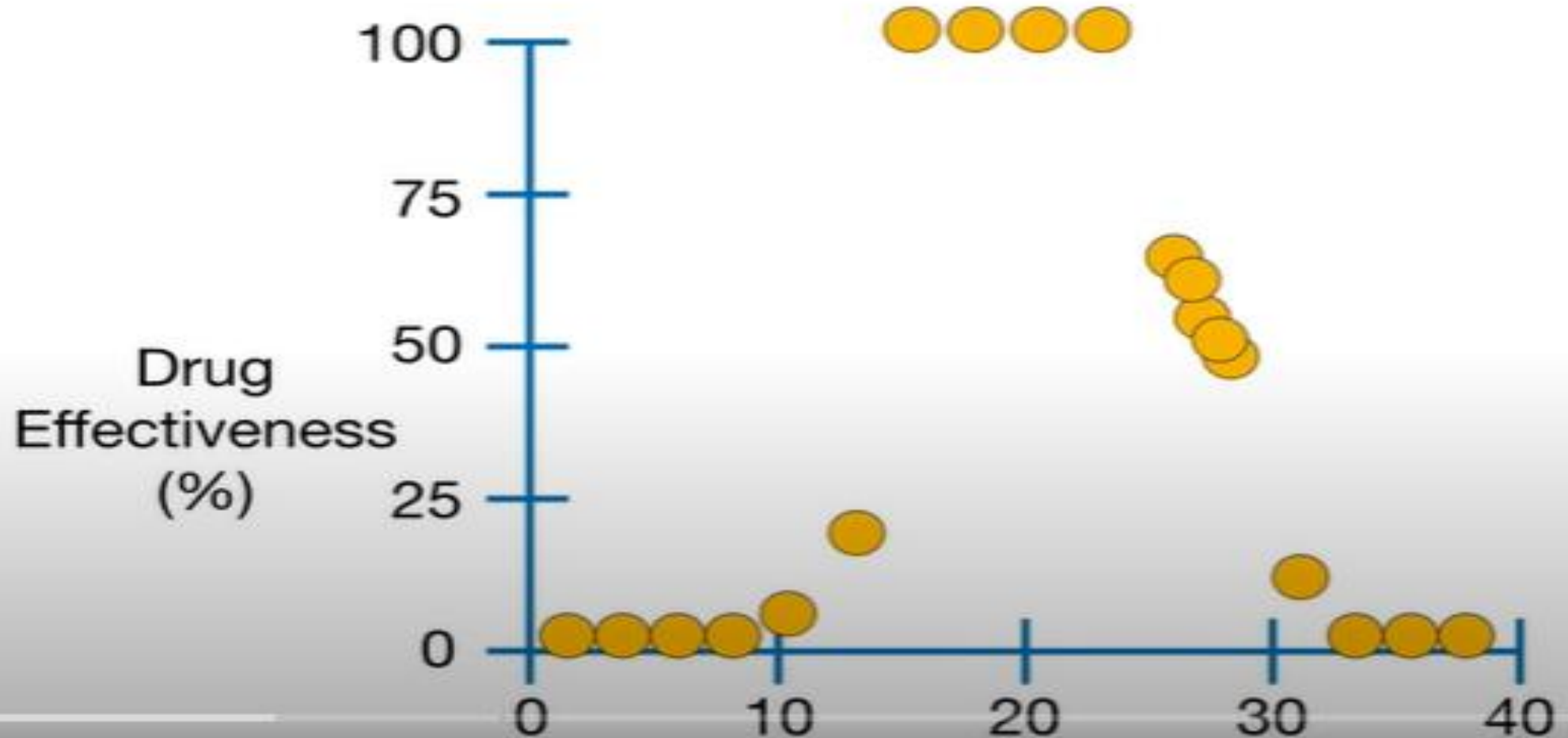


...we could use the line to predict that a **27 mg Dose** should be **62% Effective**.

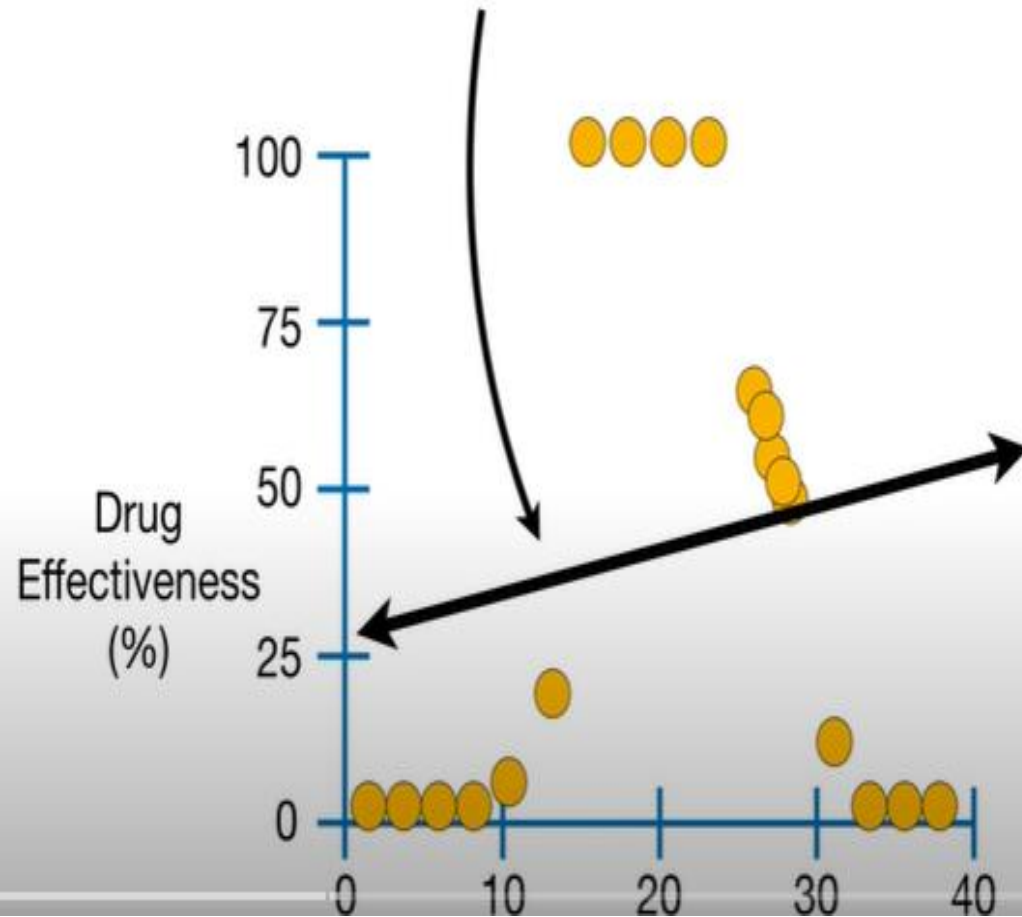


Low Dosages are Not effective
Somewhat higher dosages = 50%
Moderate work really well
High are ineffective

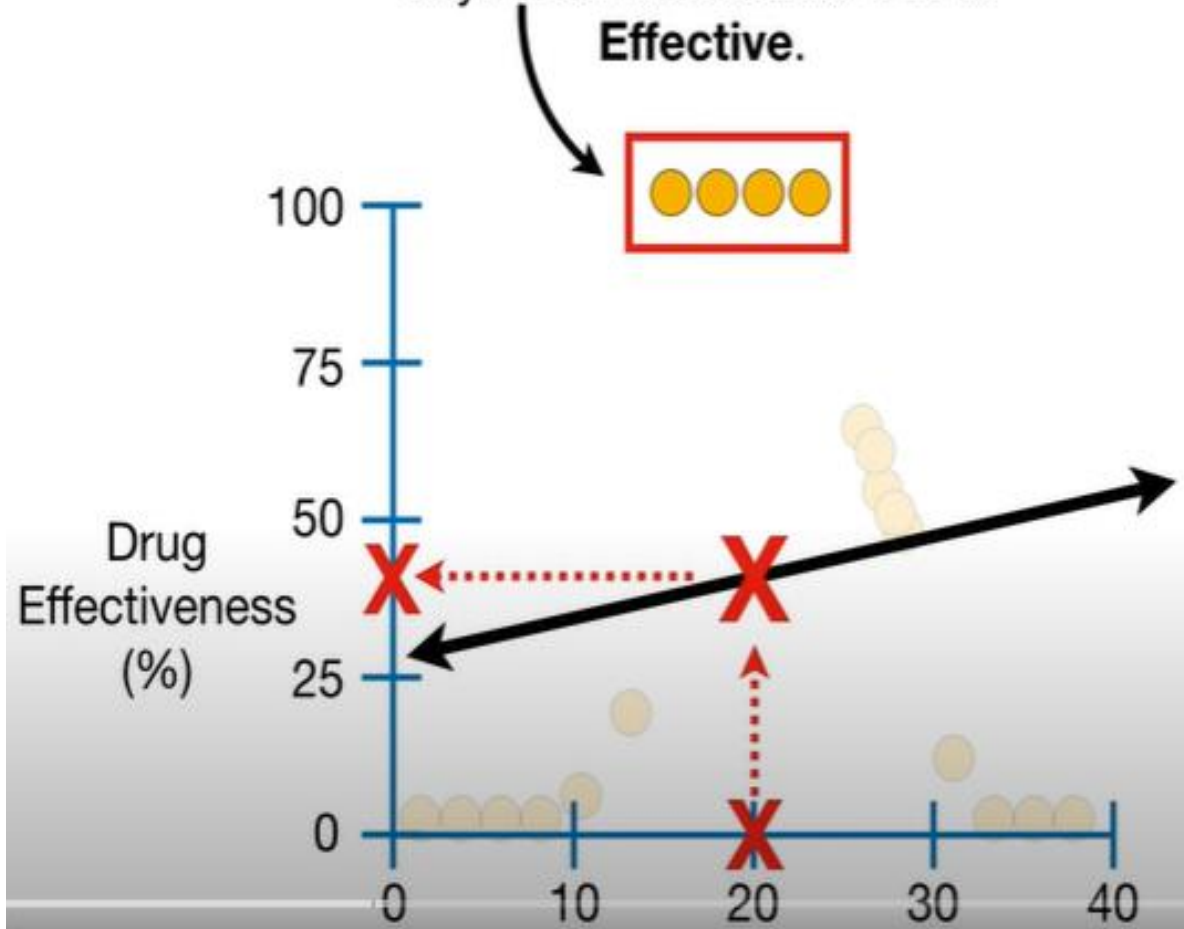
However, what if the data looked like this?



In this case, fitting a straight line to the data will not be very useful.



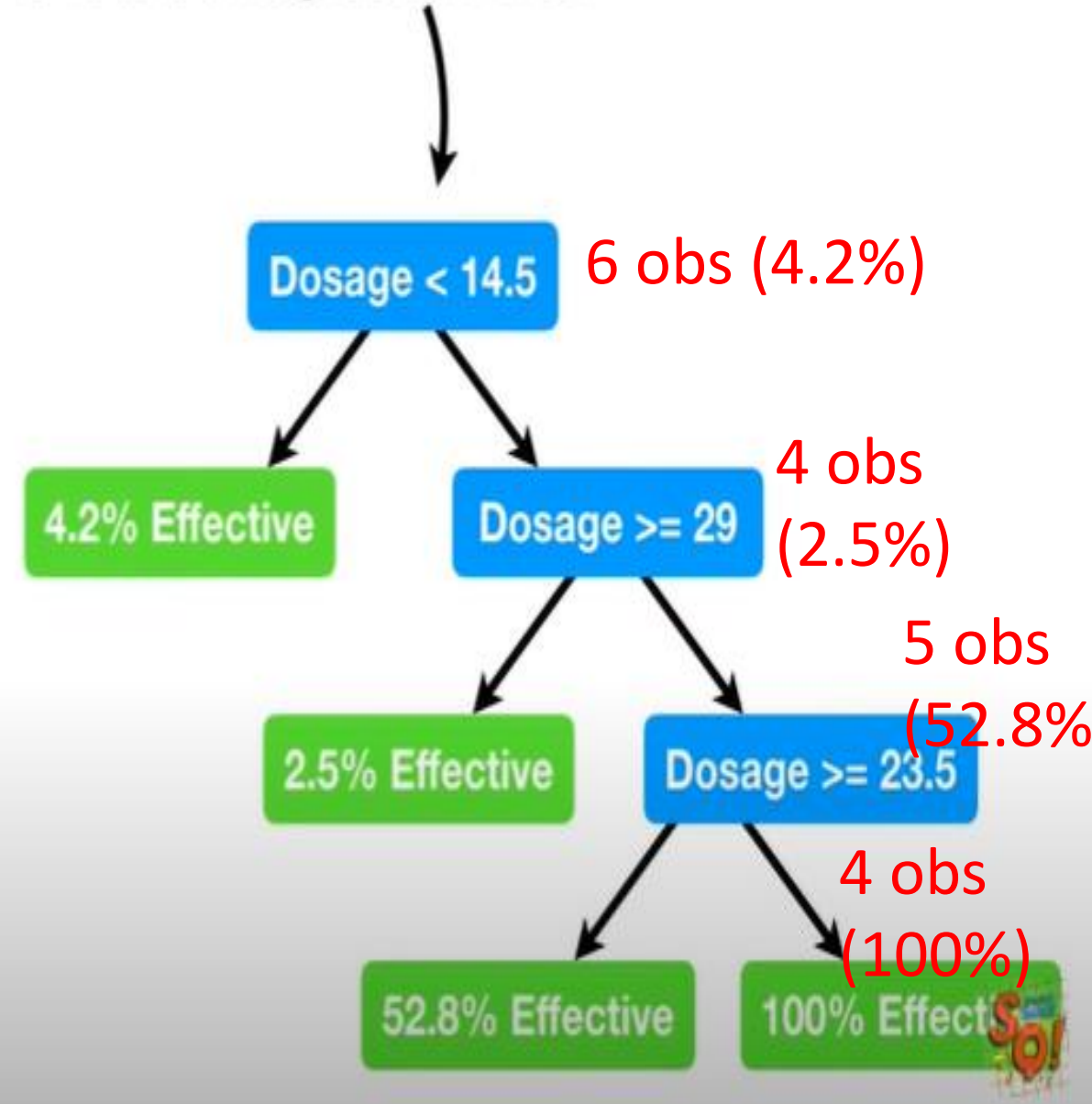
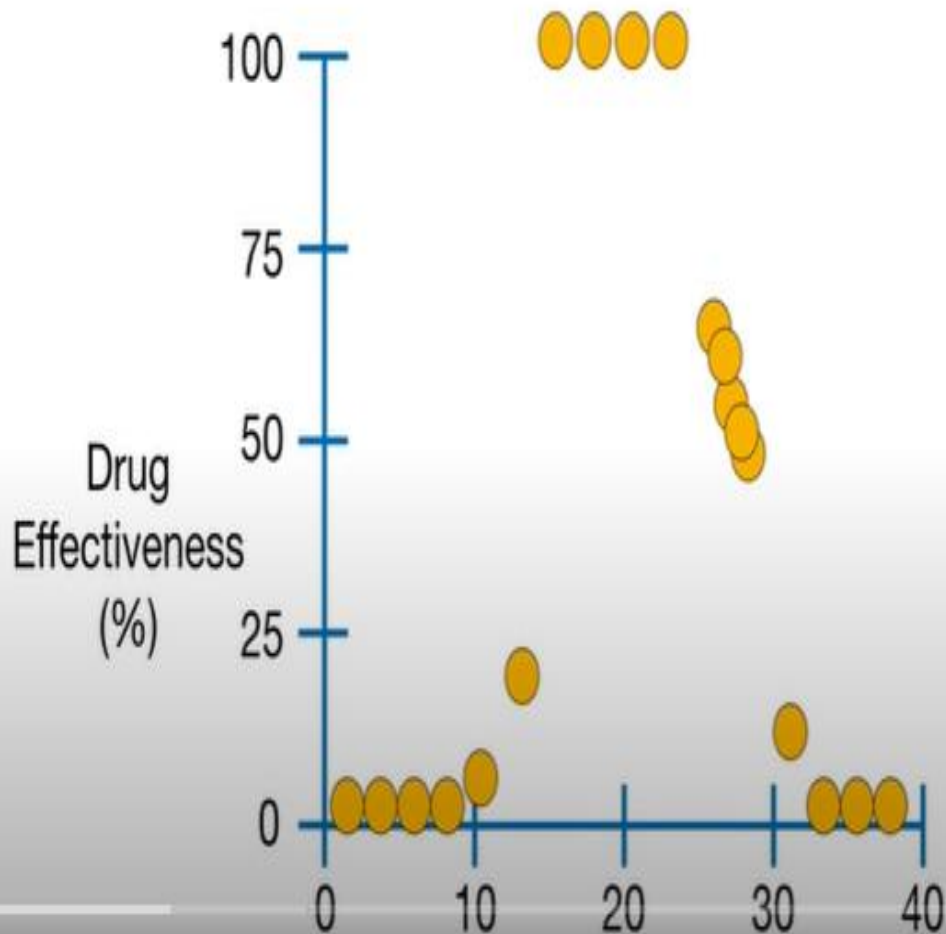
...even though the observed data says that it should be **100% Effective.**



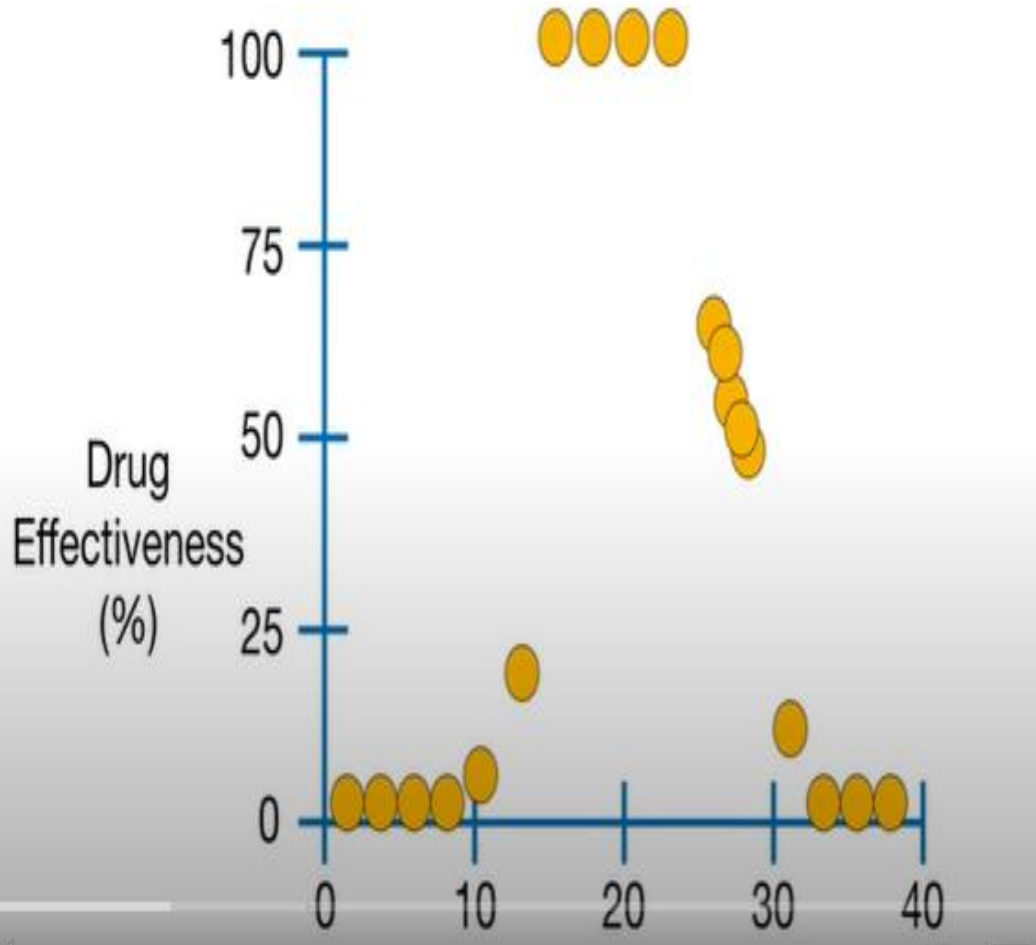
What about effectiveness of 20mg dosage?

One option is to use a **Regression Tree**.

However, what if the data looked like this?



However, what if the data
looked like this?

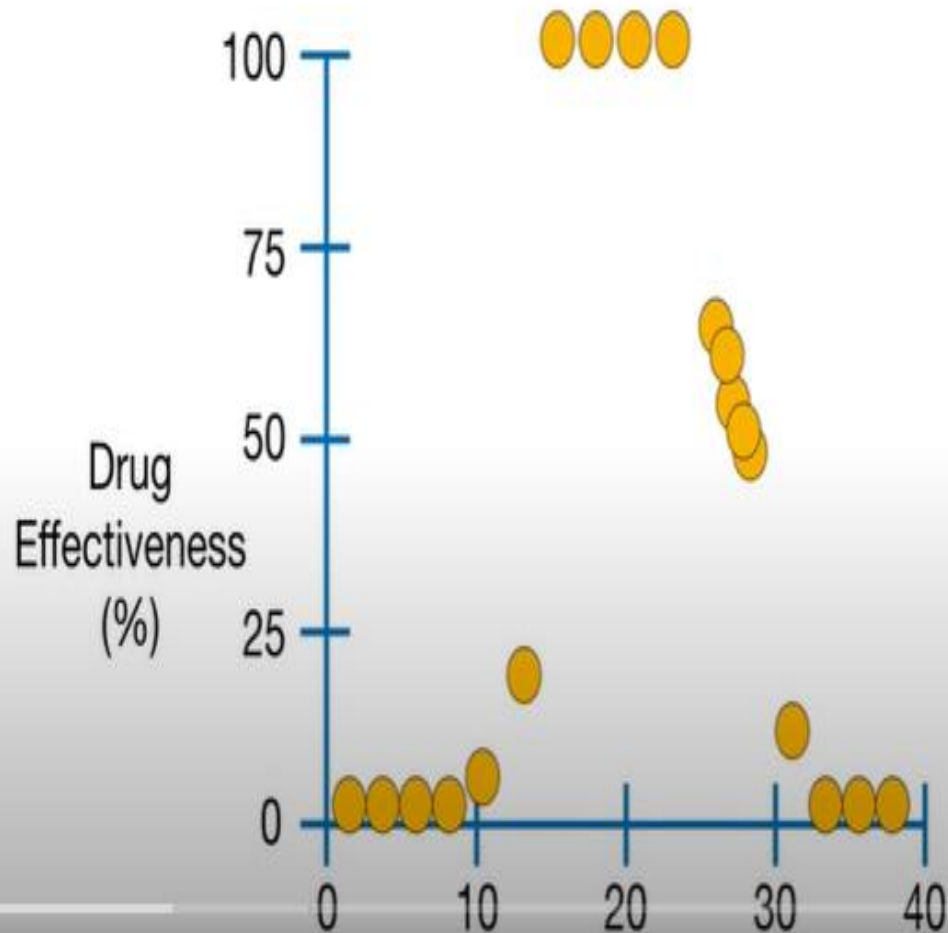


You might think that the
prediction for a 27mg dosage
can be predicted directly from
the shown graph

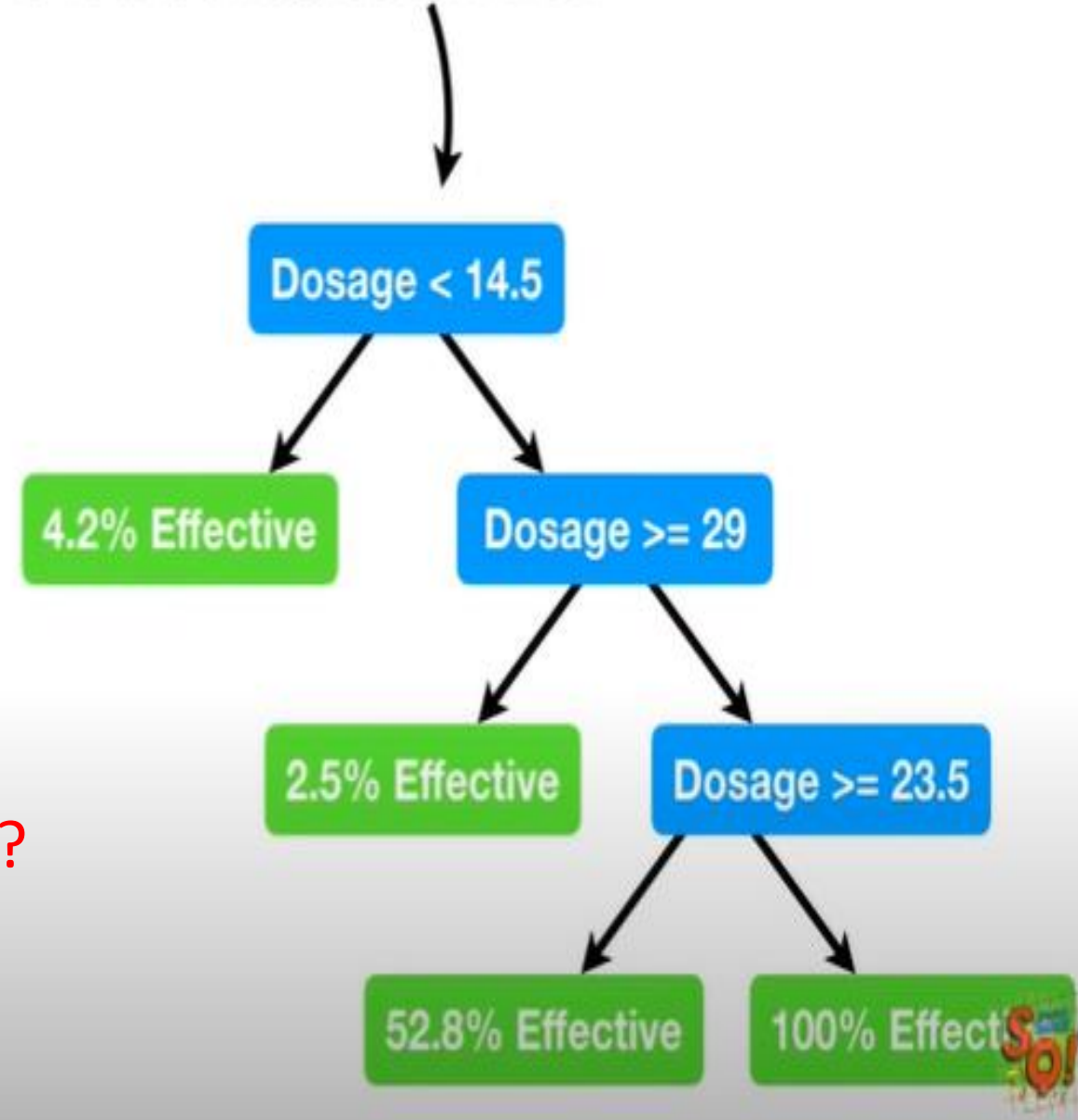
This is true only if the data
is super simple. If there are
many predictors then the regression
tree becomes essential

One option is to use a **Regression Tree**.

However, what if the data looked like this?



Why 14.5?



Avg. = 3

Dosage < 3 {0, 38.8}

Not good prediction (100%)

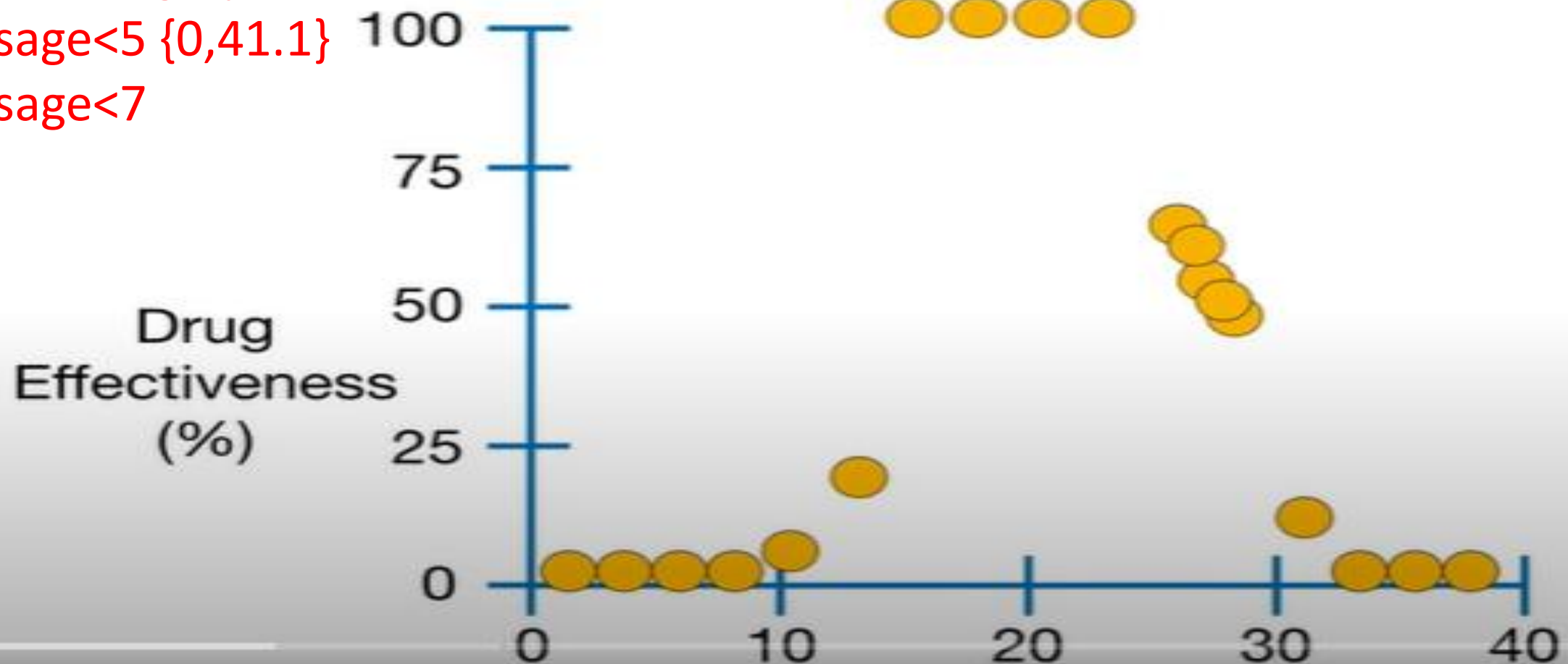
Residuals $(0-0)^2 + (x-38.8)^2$

27,468.5 (graph, 3)

Dosage < 5 {0, 41.1}

Dosage < 7

However, what if the data looked like this?



Avg. = 3

Dosage<3 {0,38.8}

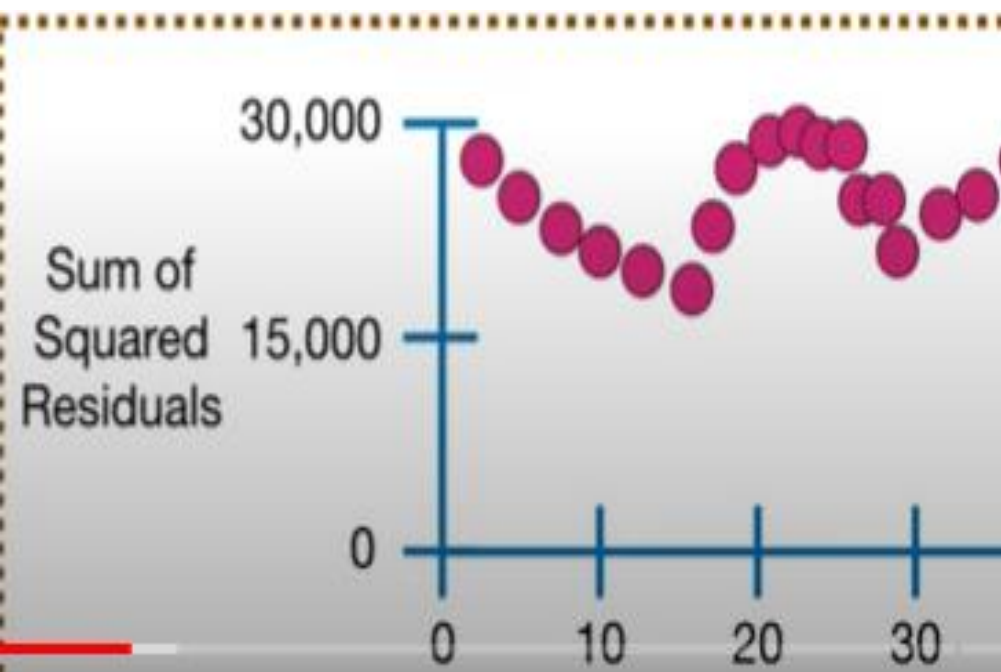
Not good prediction (100%)

Residuals $(0-0)^2 + (x-38.8)^2$

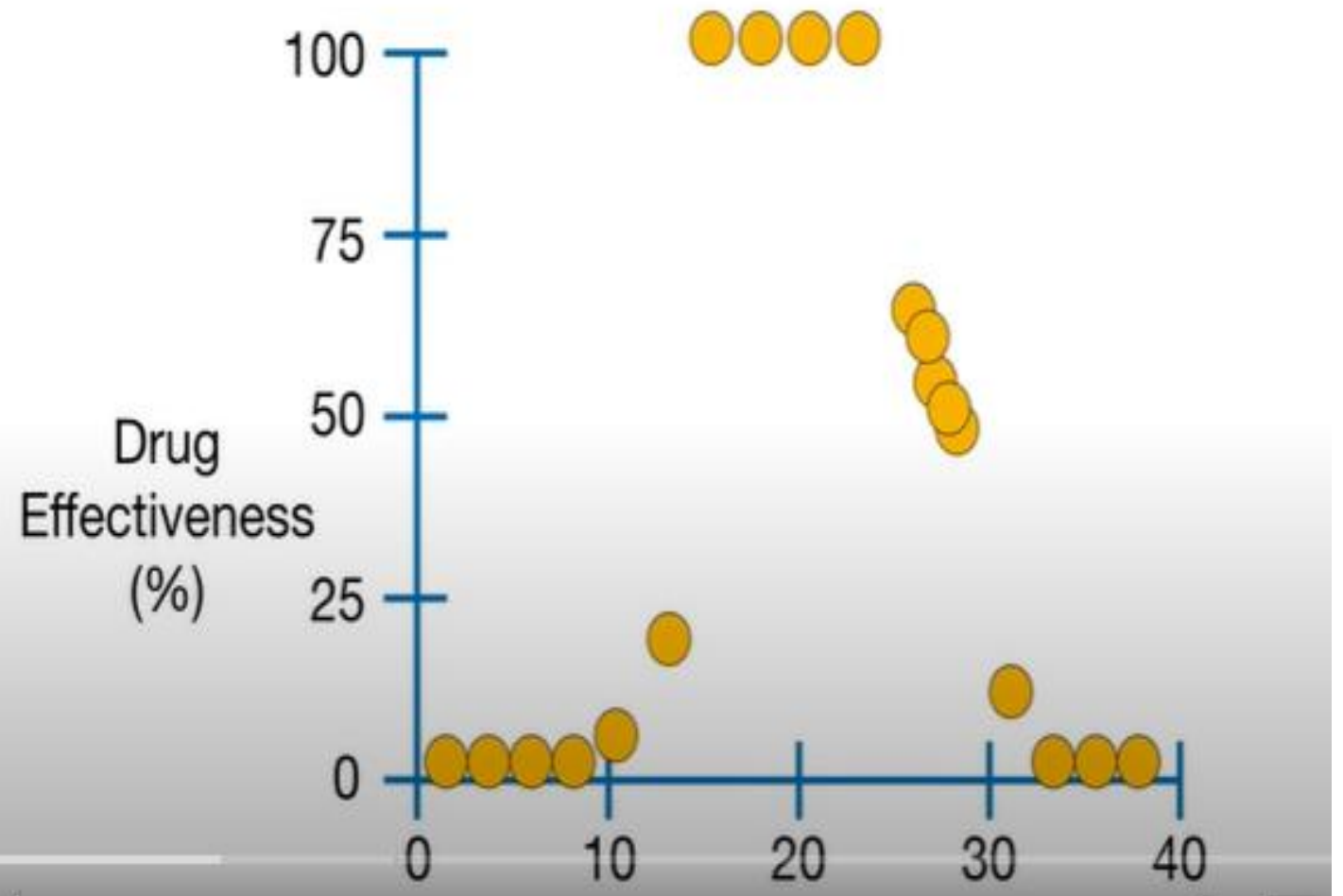
27,468.5 (graph,3)

Dosage<5 {0,41.1}

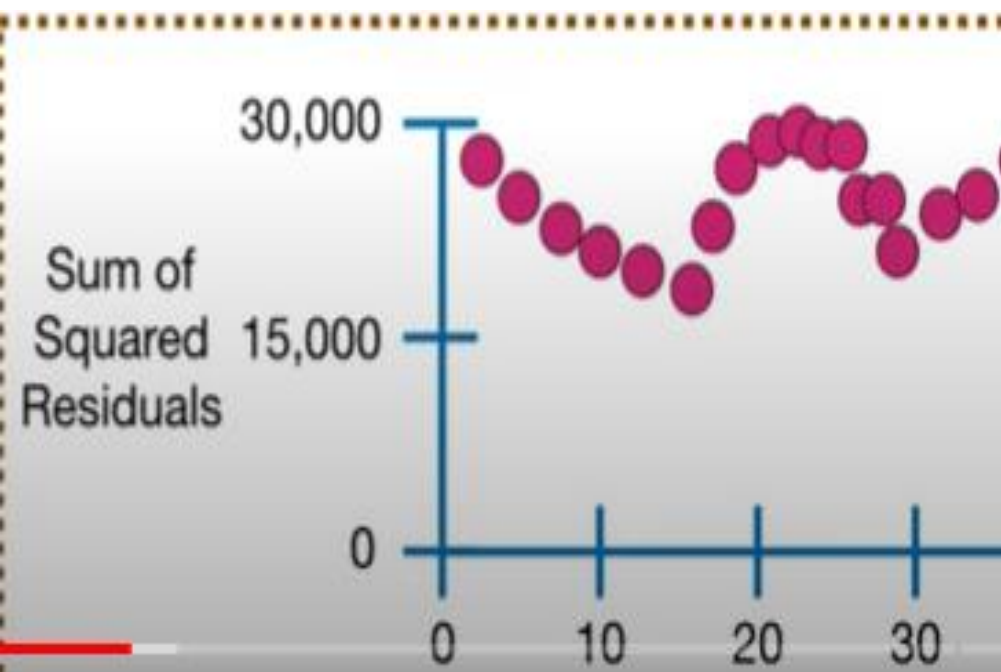
Dosage<7



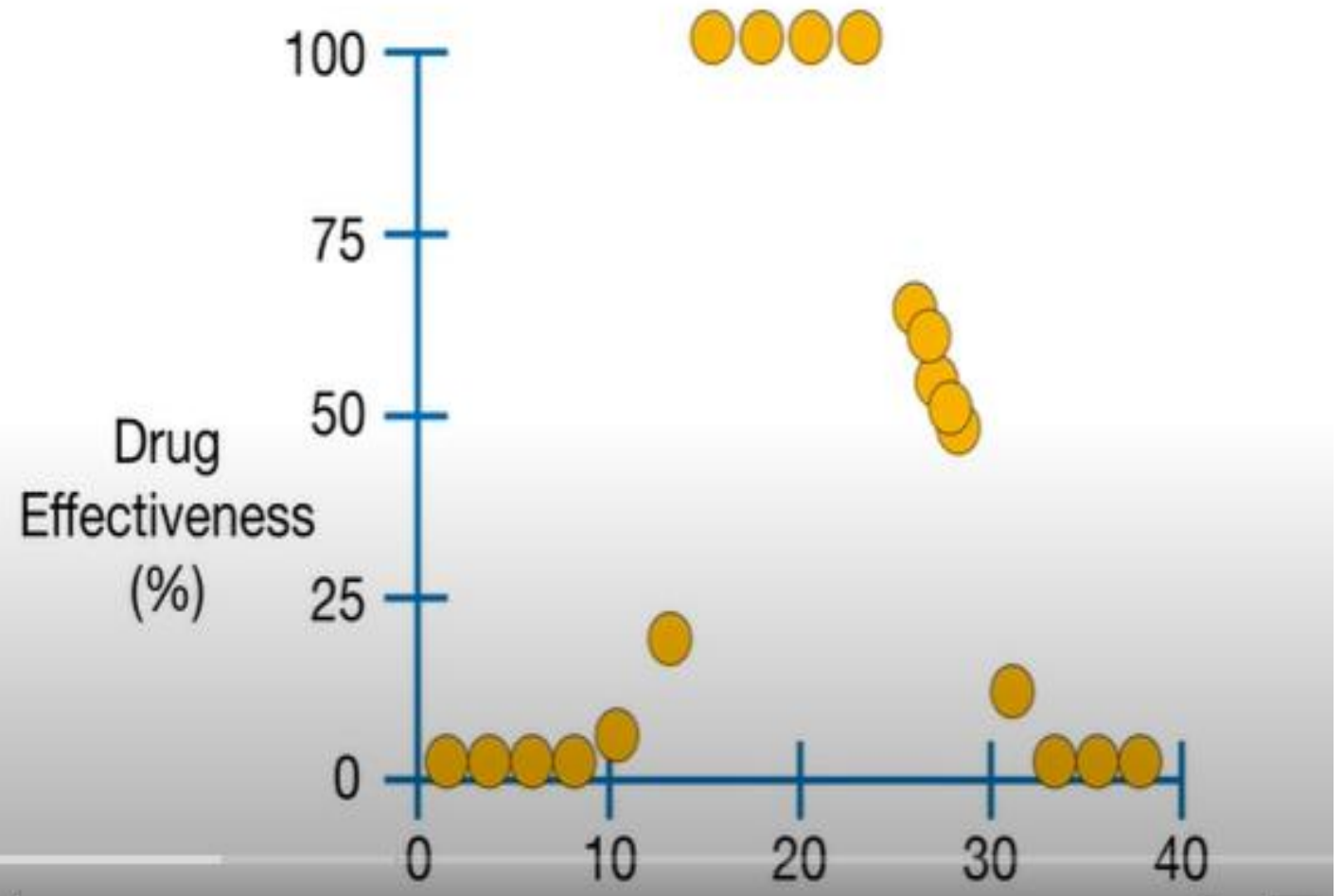
However, what if the data looked like this?



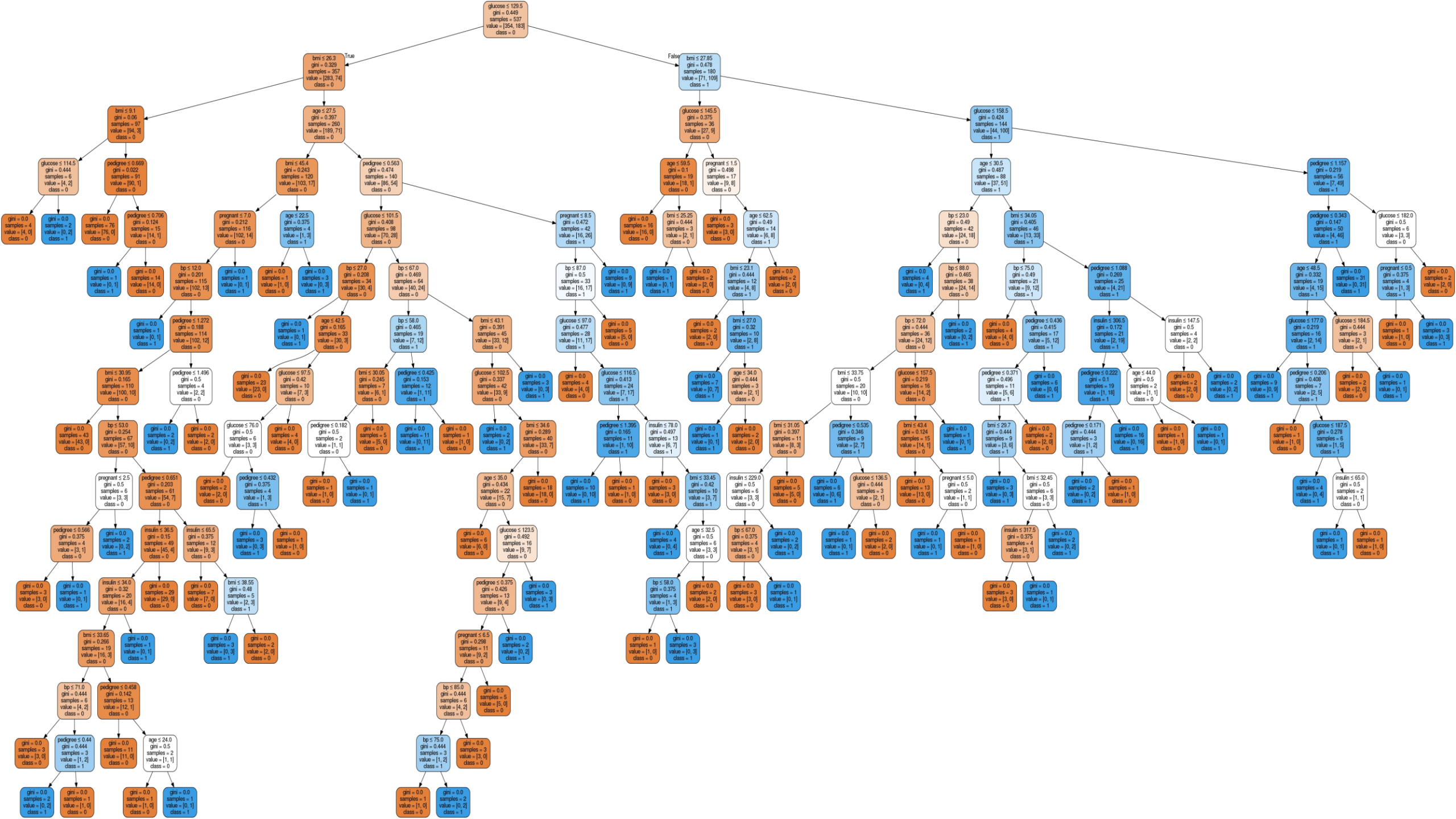
The minimum number of
Observations to allow for a
Split is 20

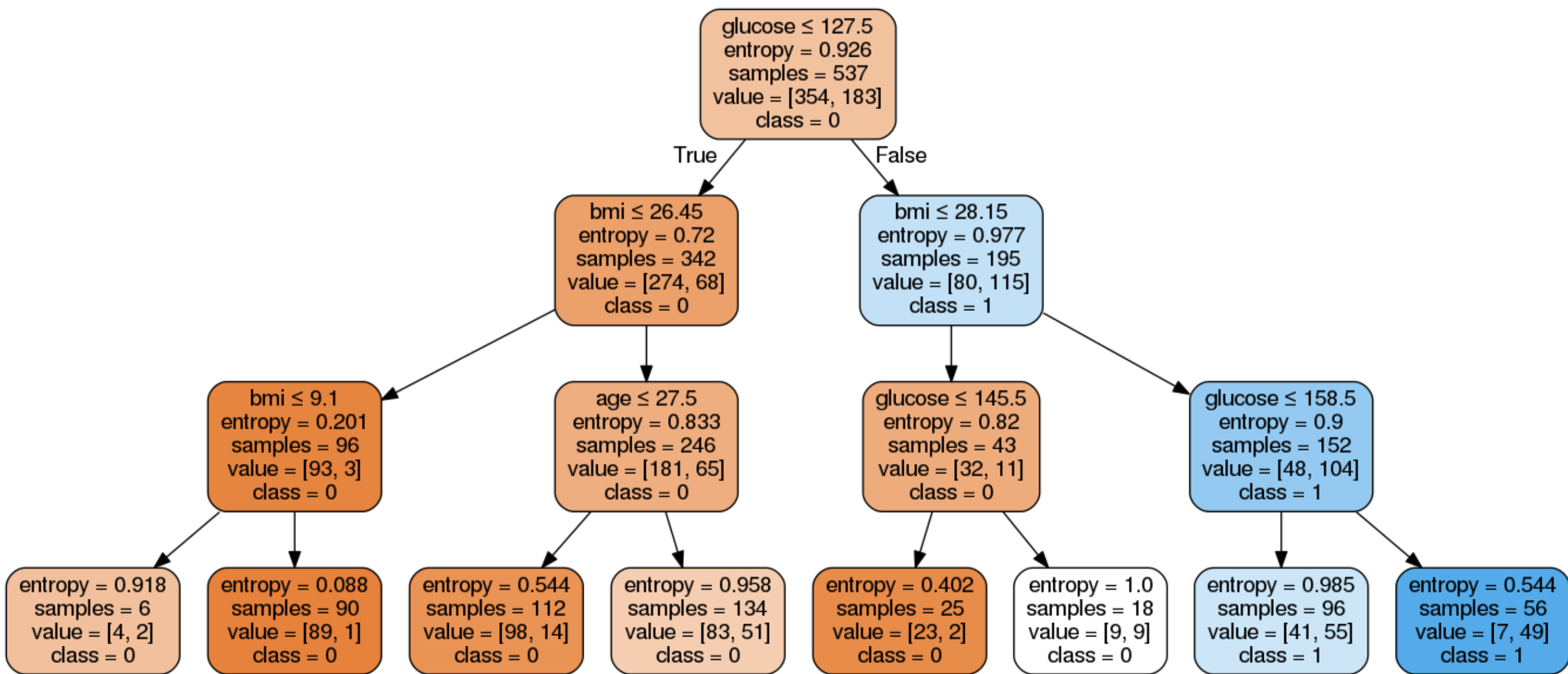


However, what if the data
looked like this?



- They are very interpretable.
 - Making predictions is fast (no complicated calculations, just looking up constants in the tree).
 - It's easy to understand what variables are important in making the prediction. The internal nodes (splits) are those variables that most largely reduced the SSE.
 - If some data is missing, we might not be able to go all the way down the tree to a leaf, but we can still make a prediction by averaging all the leaves in the sub-tree we do reach.
 - The model provides a non-linear “jagged” response, so it can work when the true regression surface is not smooth. If it is smooth, though, the piecewise-constant surface can approximate it arbitrarily closely (with enough leaves).
 - There are fast, reliable algorithms to learn these trees.
-
- Single regression trees have high variance, resulting in unstable predictions (an alternative subsample of training data can significantly change the terminal nodes).
 - Due to the high variance single regression trees have poor predictive accuracy.



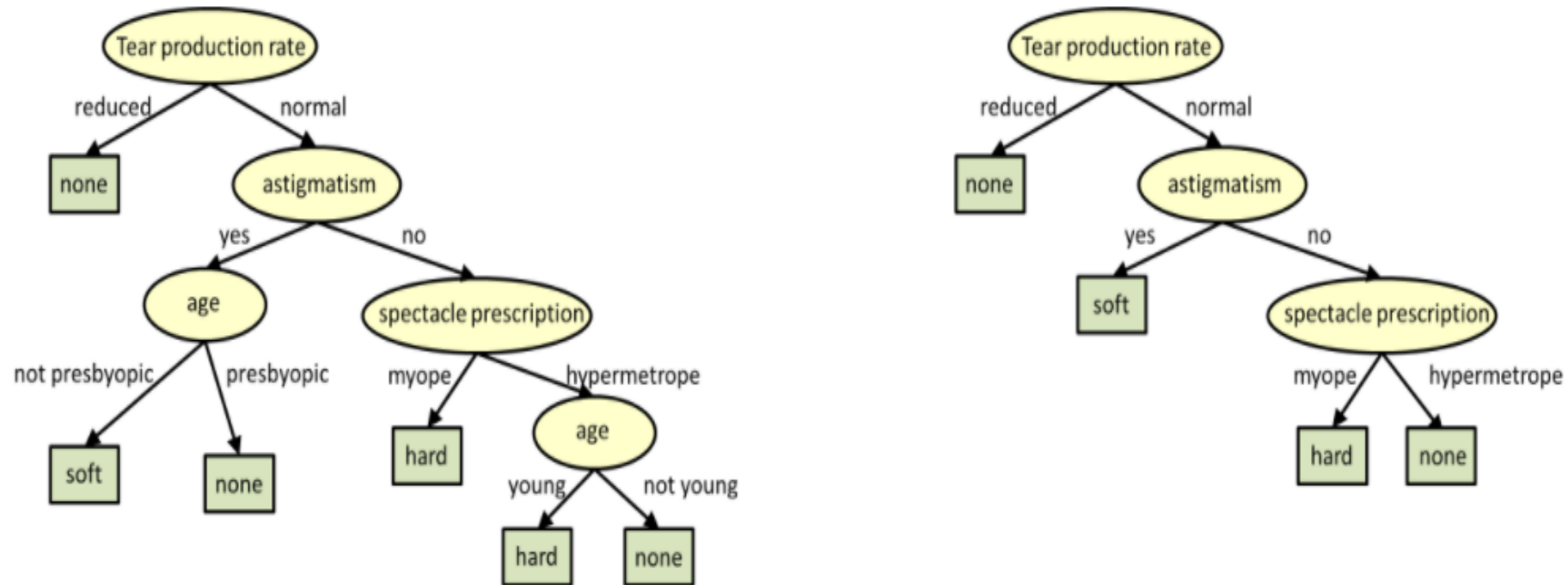


Decision trees that are trained on any training data run the risk of *overfitting* the training data.

What we mean by this is that eventually each leaf will represent a very specific set of attribute combinations that are seen in the training data, and the tree will consequently not be able to classify attribute value combinations that are not seen in the training data.

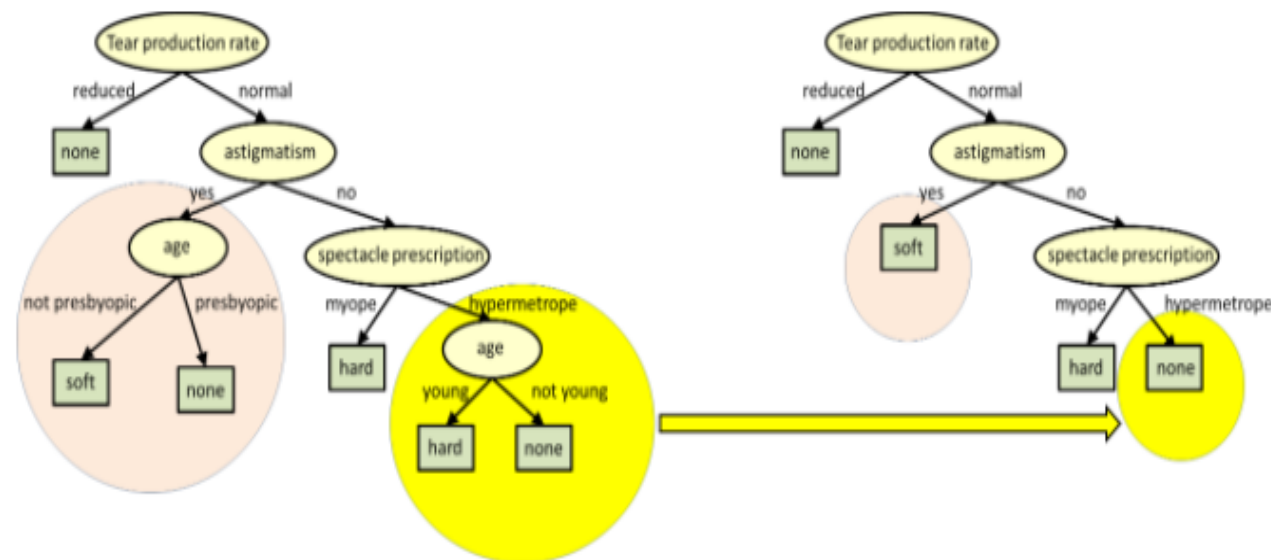
In order to prevent this from happening, we must *prune* the decision tree.

By *pruning* we mean that the lower ends (the leaves) of the tree are “snipped” until the tree is much smaller. The figure below shows an example of a full tree, and the same tree after it has been pruned to have only 4 leaves.



The simplest technique is to prune out portions of the tree that result in the least information gain. This procedure does not require any additional data, and only bases the pruning on the information that is already computed when the tree is being built from training data.

The process of IG-based pruning requires us to identify “twigs”, nodes whose children are all leaves. “Pruning” a twig removes all of the leaves which are the children of the twig, and makes the twig a leaf. The figure below illustrates this.



Caption: Pruning the encircled twig in the left figure results in the tree to the right. The twig now becomes a leaf.

The algorithm for pruning is as follows:

1. Catalog all twigs in the tree
2. Count the total number of leaves in the tree.
3. While the number of leaves in the tree exceeds the desired number:
 - a. Find the twig with the least Information Gain
 - b. Remove all child nodes of the twig.
 - c. Relabel twig as a leaf.
 - d. Update the leaf count.

An alternate approach is to prune the tree to maximize classification performance on a validation set (a data set with known labels, which was not used to train the tree).

We pass the validation data down the tree. At each node, we record the total number of instances and the number of misclassifications, if that node were actually a leaf. We do this at all nodes and leaves.

Subsequently, we prune all twigs where pruning results in the smallest overall increase in classification error.

The overall algorithm for pruning is as follows:

Stage 1:

1. For each instance of validation data:
 - Recursively pass
 - Catalog all twigs in the tree
 - Count the total number of leaves in the tree.
 - While the number of leaves in the tree exceeds the desired number:
 - a. Find the twig with the least Information Gain
 - b. Remove all child nodes of the twig.
 - c. Relabel twig as a leaf.
 - d. Update the leaf count.