**Standardizing**

**Standardizing** is completed by taking each value of your column, subtracting the mean of the column, and then dividing by the standard deviation of the column. In Python, let's say you have a column in df called height. You could create a standardized height as:

df["height\_standard"] = (df["height"] - df["height"].mean()) / df["height"].std()

This will create a new "standardized" column where each value is a comparison to the mean of the column, and a new, standardized value can be interpreted as the number of standard deviations the original height was from the mean. This type of feature scaling is by far the most common of all techniques (for the reasons discussed here, but also likely because of precedent).

**Normalizing**

A second type of feature scaling that is very popular is known as **normalizing**. With normalizing, data are scaled between 0 and 1. Using the same example as above, we could perform normalizing in Python in the following way:

df["height\_normal"] = (df["height"] - df["height"].min()) / \

(df["height"].max() - df['height'].min())

Chart, scatter chart

Description automatically generated

# Hyperparameters for Decision Trees

In order to create decision trees that will generalize to new problems well, we can tune a number of different aspects about the trees. We call the different aspects of a decision tree "hyperparameters". These are some of the most important hyperparameters used in decision trees:

### Maximum Depth

The maximum depth of a decision tree is simply the largest possible length between the root to a leaf. A tree of maximum length k*k* can have at most 2^k2*k* leaves.

Chart

Description automatically generated

Maximum depth of a decision tree

### Minimum number of samples to split

A node must have at least min\_samples\_split samples in order to be large enough to split. If a node has fewer samples than min\_samples\_split samples, it will not be split, and the splitting process stops.

A picture containing text, athletic game

Description automatically generated

Minimum number of samples to split

However, min\_samples\_split doesn't control the minimum size of leaves. As you can see in the example on the right, above, the parent node had 20 samples, greater than min\_samples\_split = 11, so the node was split. But when the node was split, a child node was created with that had 5 samples, less than min\_samples\_split = 11.

### Minimum number of samples per leaf

When splitting a node, one could run into the problem of having 99 samples in one of them, and 1 on the other. This will not take us too far in our process, and would be a waste of resources and time. If we want to avoid this, we can set a minimum for the number of samples we allow on each leaf.

A picture containing athletic game

Description automatically generated

Minimum number of samples per leaf

This number can be specified as an integer or as a float. If it's an integer, it's the minimum number of samples allowed in a leaf. If it's a float, it's the minimum percentage of samples allowed in a leaf. For example, 0.1, or 10%, implies that a particular split will not be allowed if one of the leaves that results contains less than 10% of the samples in the dataset.

If a threshold on a feature results in a leaf that has fewer samples than min\_samples\_leaf, the algorithm will not allow that split, but it may perform a split on the same feature at a different threshold, that does satisfy min\_samples\_leaf.