DECISION TREES WITH PYTHON

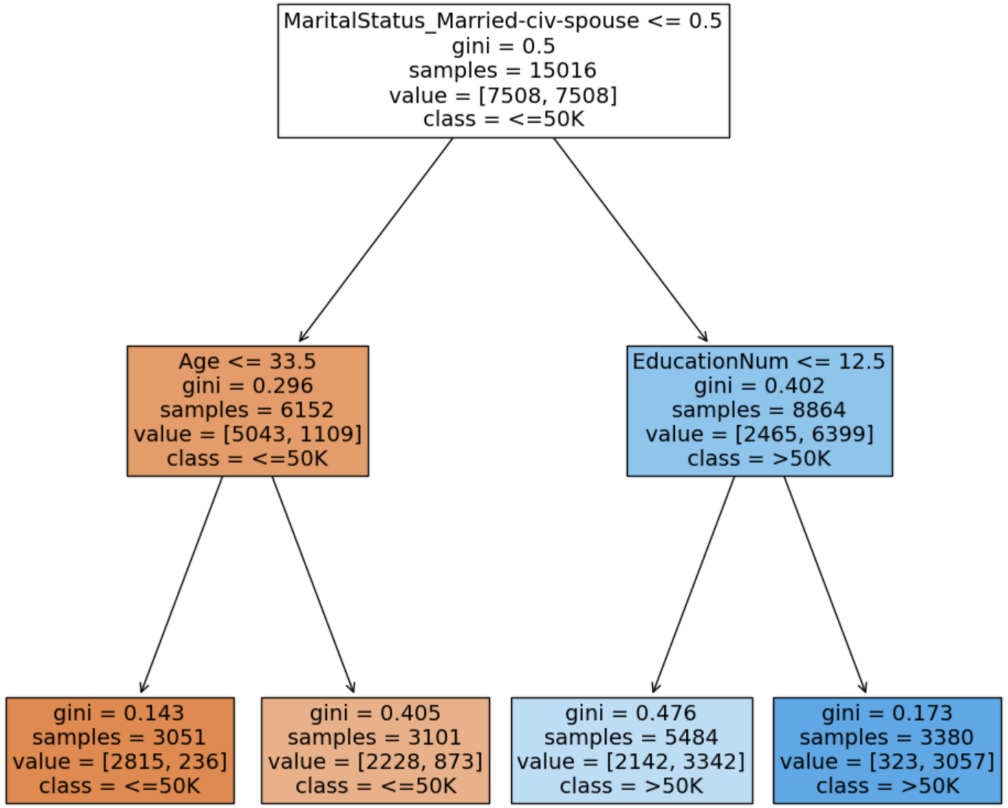
A CRASH COURSE

```python
from sklearn.tree import DecisionTreeClassifier

# Train a CART-like classification tree
tree_1 = DecisionTreeClassifier(min_samples_leaf = 3000,
                                random_state = 12345)

tree_1.fit(adult_X, adult_y)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(min_samples_leaf=3000, random_state=12345)
```



The Code Is the Easy Part!

```python
from sklearn.tree import DecisionTreeClassifier

# Train a CART-like classification tree
tree_1 = DecisionTreeClassifier(min_samples_leaf = 3000,
                                random_state = 12345)

tree_1.fit(adult_X, adult_y)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(min_samples_leaf=3000, random_state=12345)
```

Coding up ML models is ridiculously easy.

Crafting useful ML models is another story.

```
PY
from sklearn.tree import DecisionTreeClassifier

# Train a CART-like classification tree
tree_1 = DecisionTreeClassifier(min_samples_leaf = 3000, random_state = 12345)
tree_1.fit(adult_X, adult_y)
```

ML Fundamentals

What Is Machine Learning?

Dave ON DATA

"The field of study which gives computers the capability to learn without being explicitly programmed."

- Arthur Samuel, pioneer in computer gaming and AI who coined the term "machine learning" in 1959

Also known as **predictive analytics**, machine learning uses historical data to "learn" patterns and offer probabilistic predictions on never-before-seen data.

What Is an Algorithm?

An **algorithm** is a well-defined procedure or formula that takes input and produces output. *It's a detailed "recipe" computers follow in order to perform a task.*

Machine learning uses historical data and algorithms to make predictions. There are many, many algorithms, each with its own recipe for solving the optimization problem at hand.

No Free Lunch Theorem:
No single algorithm is going to offer the most optimal outcome for *every* given data set.

LinkedIn uses an algorithm to select posts to show you.

Amazon uses an algorithm to suggest relevant items while you shop.

Algorithms help you answer questions…

Can we use census data to predict whether or not someone earns >50k?

Types of Data

- Table / Dataset / DataFrame / Matrix
- Rows / Examples / Observations / Samples
- Columns / Character traits / Attributes / Features
- Label / Prediction / Output

Dave ON DATA

Age	Education	Marital Status	Race	Sex	Hours Per Week	Label
39	Bachelors	Never-married	White	Male	40	<=50K
50	Bachelors	Married-civ-spouse	White	Male	13	<=50K
38	HS-grad	Divorced	White	Male	40	<=50K
53	11th	Married-civ-spouse	Black	Male	40	<=50K
28	Bachelors	Married-civ-spouse	Black	Female	40	<=50K
37	Masters	Married-civ-spouse	White	Female	40	<=50K
49	9th	Married-spouse-absent	Black	Female	16	<=50K
52	HS-grad	Married-civ-spouse	White	Male	45	>50K
31	Masters	Never-married	White	Female	50	>50K

Numeric: Data that can be measured (e.g., age, height, weight, price)

Categorical: Data that can be divided into groups/classes (e.g., race, gender, spam)

DATA TRUMPS ALGORITHM

Types of Machine Learning

1. **Supervised Learning:**
   Your training data set is a collection of *labeled* examples
2. **Unsupervised Learning:**
   Your training data set is a collection of *unlabeled* examples
3. **Semi-supervised Learning:**
   Your training data set is a collection of both *labeled and unlabeled* examples
4. **Reinforcement Learning:**
   You have no initial training data set, rather the machine is rewarded for finding the most optimal *path* to a desired outcome

The majority of practical machine learning uses supervised learning.

Supervised Learning can be broken out into two types:

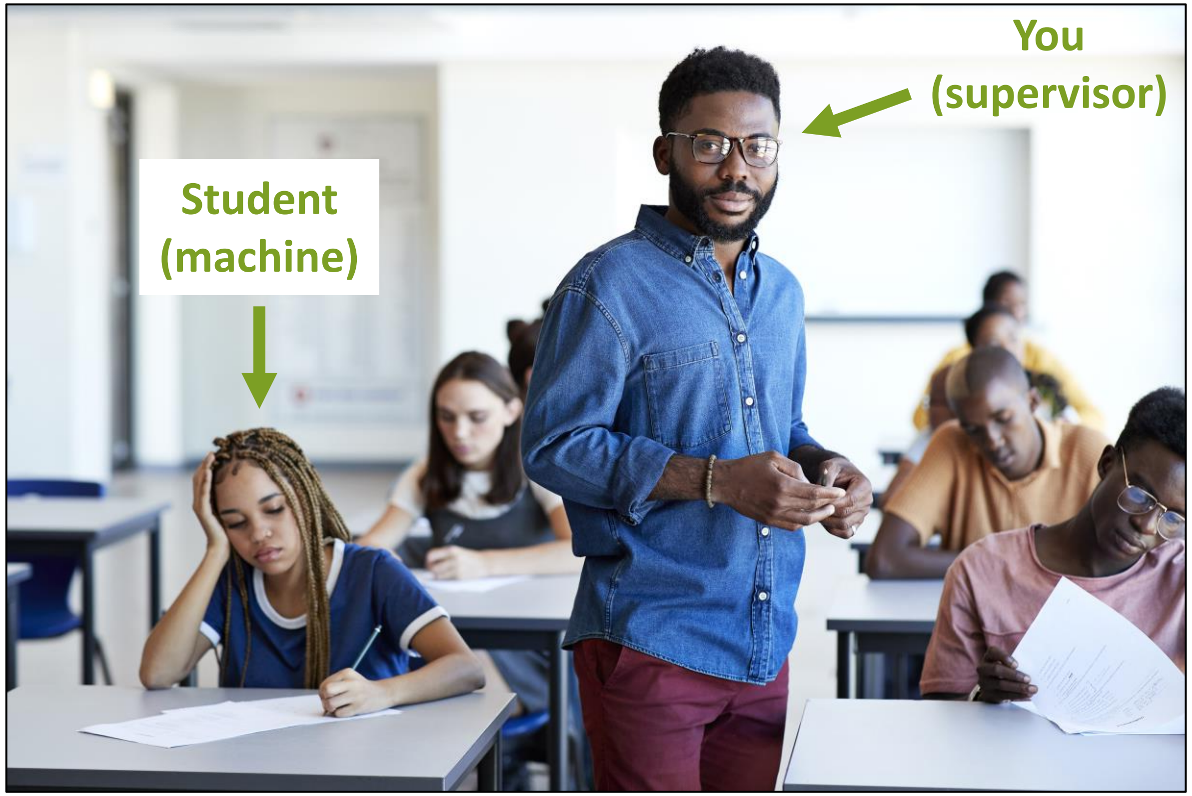
1. *Classification*: The thing we're trying to predict is categorical and we want to assign an accurate class label. *Spam or not spam?*

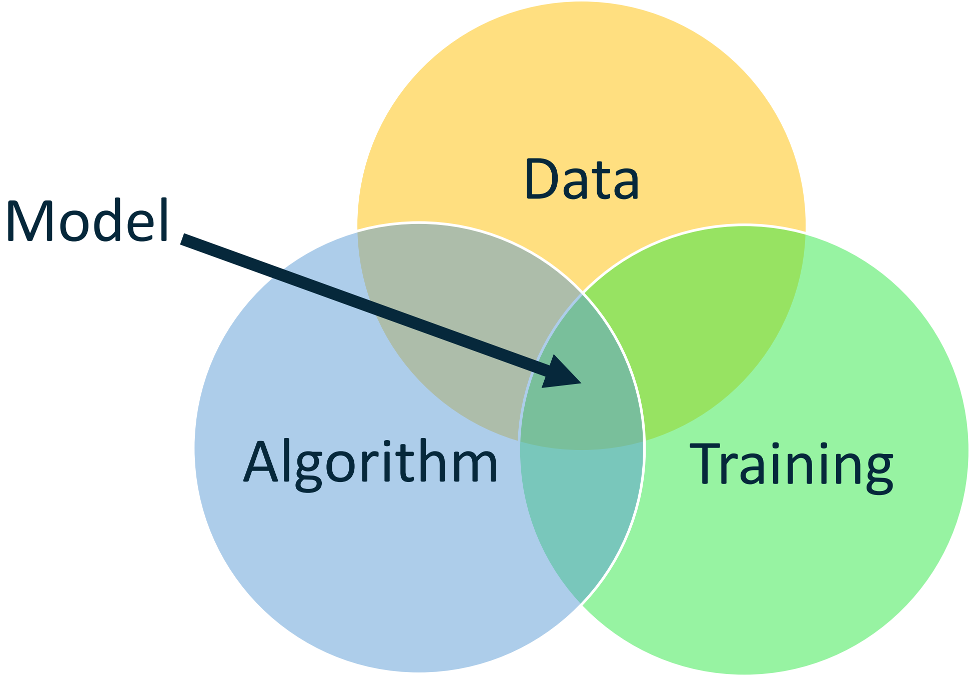
2. *Regression*: The thing we're trying to predict is numeric and we want to assign an accurate number as our target. *How much will this house cost given the square footage, number of bathrooms, etc.?*

Supervised Learning

Machine learning encompasses many areas of study.

The focus of this crash course will be supervised learning…

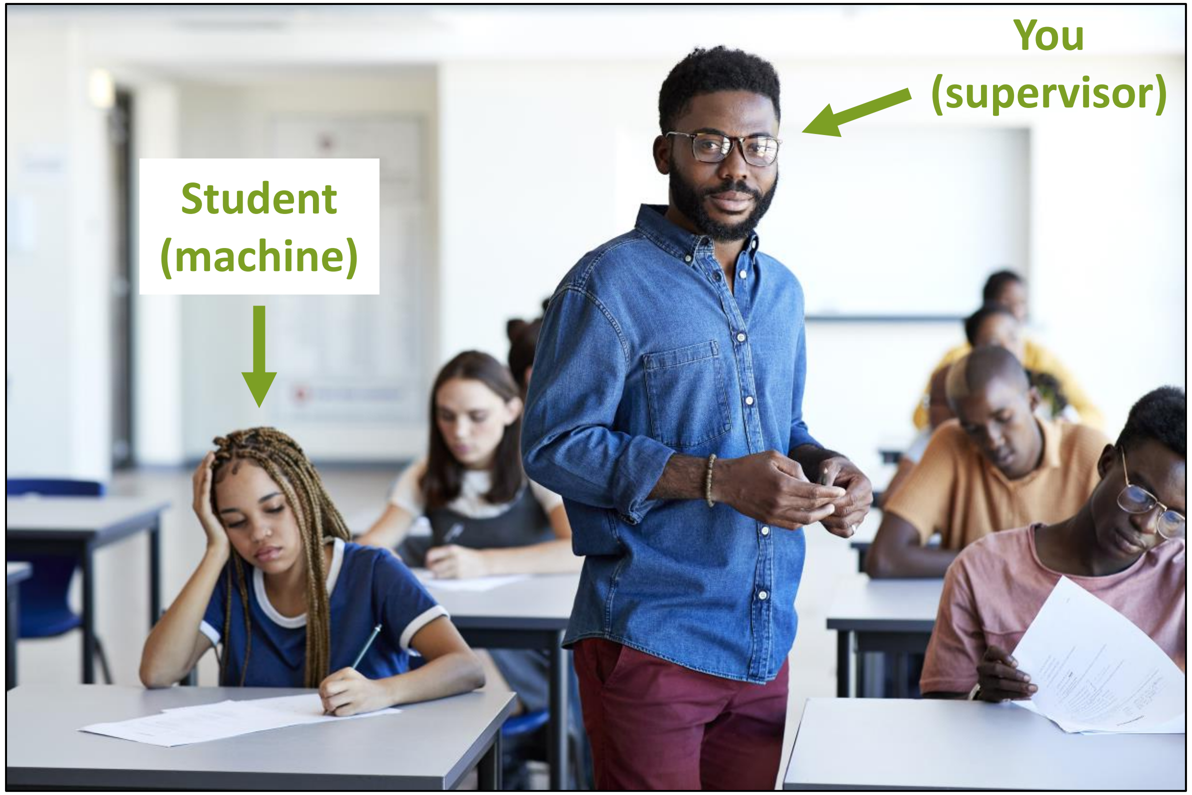


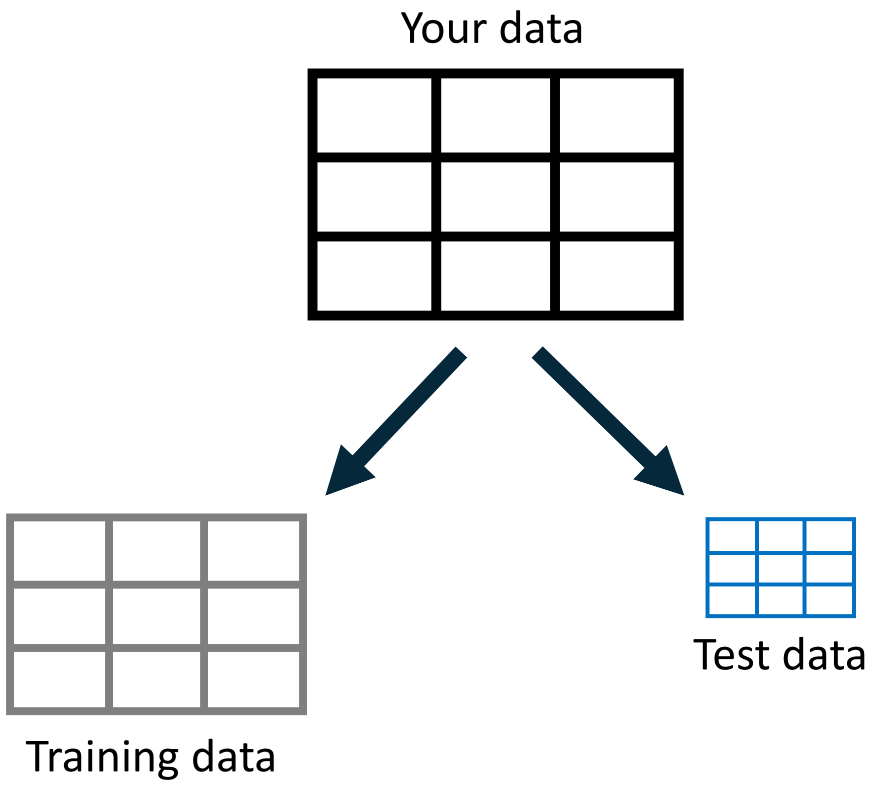


Did the Machine Learn?

As the “teacher” supervising the student’s learning, you want to evaluate how much the machine has learned.

Just as with humans, this involves **testing**.





Decision Trees

Decision Trees

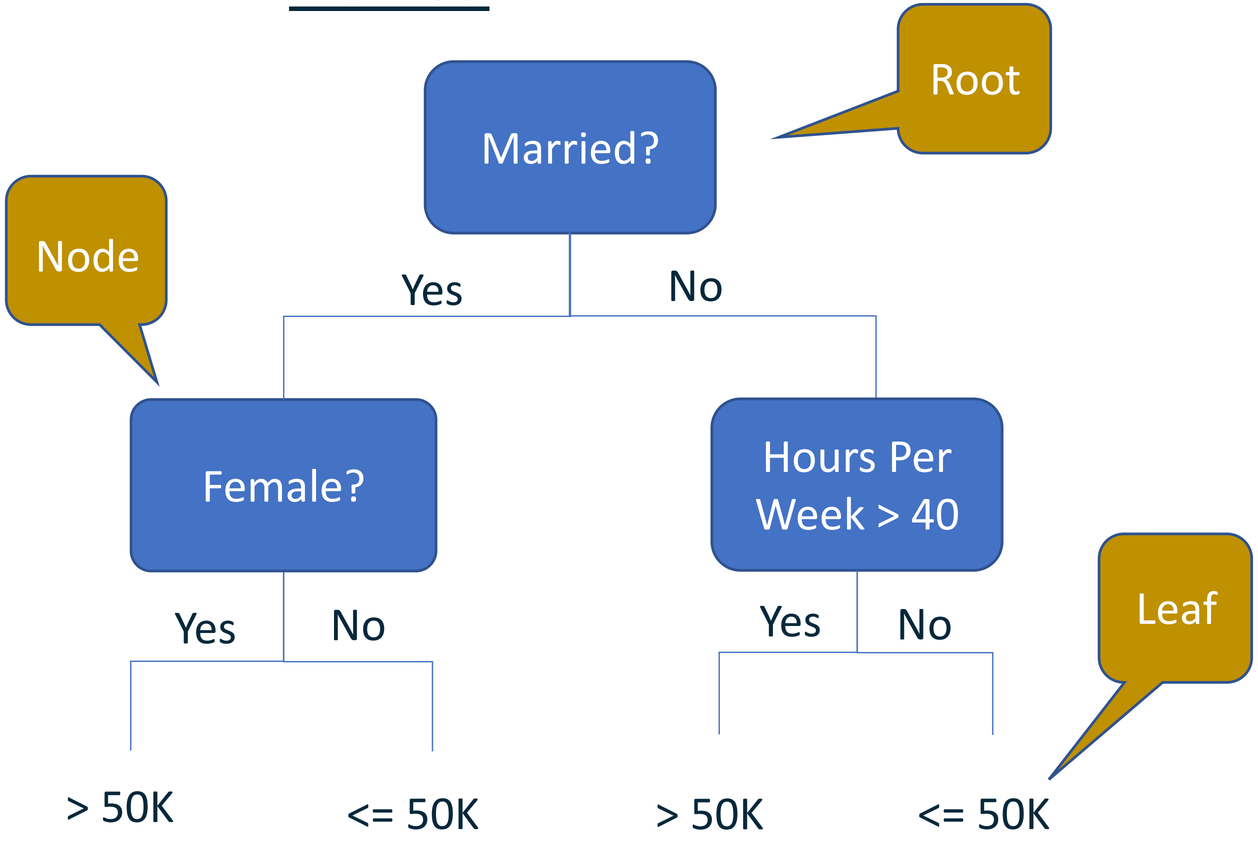
A fundamental supervised learning algorithm

Super intuitive and easy to understand

Recursively splits data into the largest, **purest** groups (all examples have the same label)

- A node is 100% ***pure*** when all of its data falls into a single class (i.e., share the same label)
- A node is 100% ***impure*** when its data is split 50/50 between classes

Trained Decision Tree



How will the above tree label this new row?

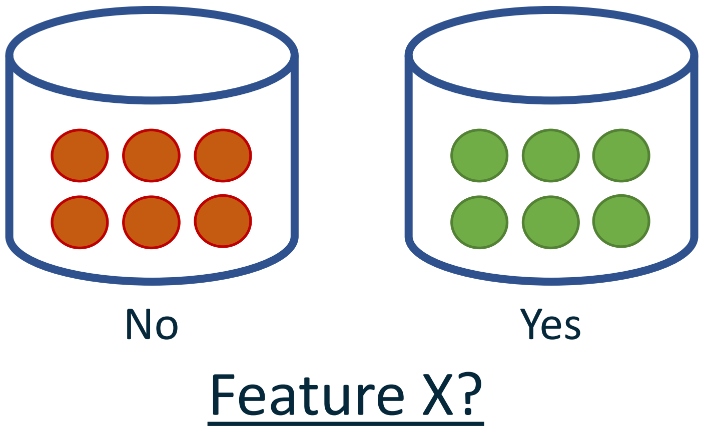
> 50K

Age	Education	Marital Status	Race	Sex	Hours Per Week	Label
49	9th	Married-spouse-absent	White	Female	16	?

Decision Trees

Age	Education	Marital Status	Race	Sex	Hours Per Week	Label
53	Masters	Married-civ-spouse	White	Male	40	<=50K
49	HS-grad	Married-spouse-absent	White	Female	16	<=50K
52	HS-grad	Married-civ-spouse	Black	Male	45	>50K
31	Masters	Never-married	Black	Female	50	>50K

Which of these features… … best splits the labels into the biggest, purest buckets?

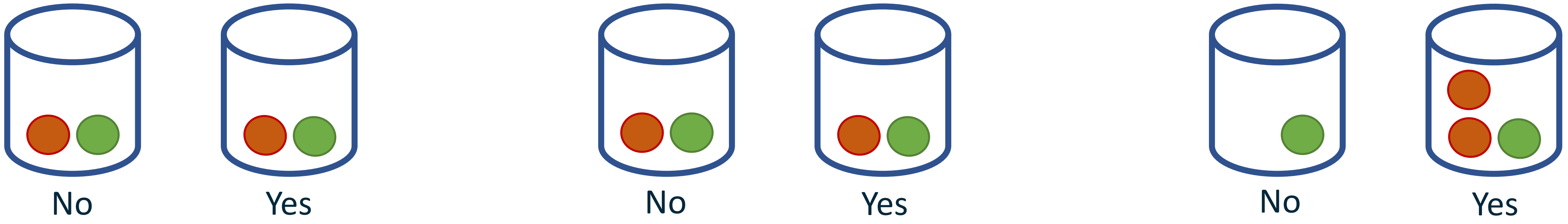


Splitting Labels

Find the feature that best separates the <=50K earners from the >50K earners, moving left to right.

Age	Education	Marital Status	Race	Sex	Hours Per Week	Label
53	Masters	Married-civ-spouse	White	Male	40	<=50K
49	HS-grad	Married-spouse-absent	White	Female	16	<=50K
52	HS-grad	Married-civ-spouse	Black	Male	45	>50K
31	Masters	Never-married	Black	Female	50	>50K

- Age creates a 50/50 split. We are completely uncertain of its effect on salary.
- Education is also 50/50. This feature won't help us make predictions.
- Marital Status doesn't offer a clean split either. Let's inspect the rest of our features…



Age >= 50?

Education = Masters?

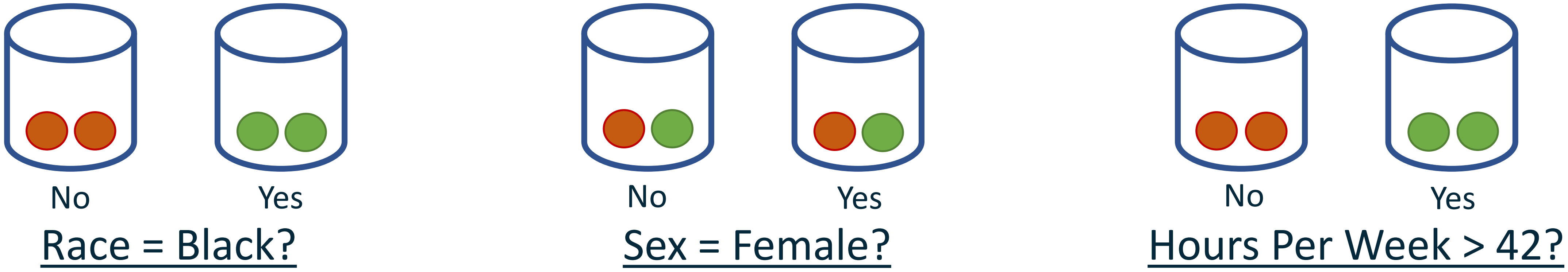
Marital Status = Married?

Splitting Labels

Trees are greedy! They use the *first* optimal feature they find.
Given our training data, **RACE IS USED AT THE ROOT NODE.**

Age	Education	Marital Status	Race	Sex	Hours Per Week	Label
53	Masters	Married-civ-spouse	White	Male	40	<=50K
49	HS-grad	Married-spouse-absent	White	Female	16	<=50K
52	HS-grad	Married-civ-spouse	Black	Male	45	>50K
31	Masters	Never-married	Black	Female	50	>50K

- Race offers a perfect split. Nice! We are completely certain of its effect on salary prediction.
- Sex is 50/50. We are completely uncertain of its effect on prediction.
- Wait, Hours Per Week > 42 yields a perfect split, too, *so which feature do we split on?*



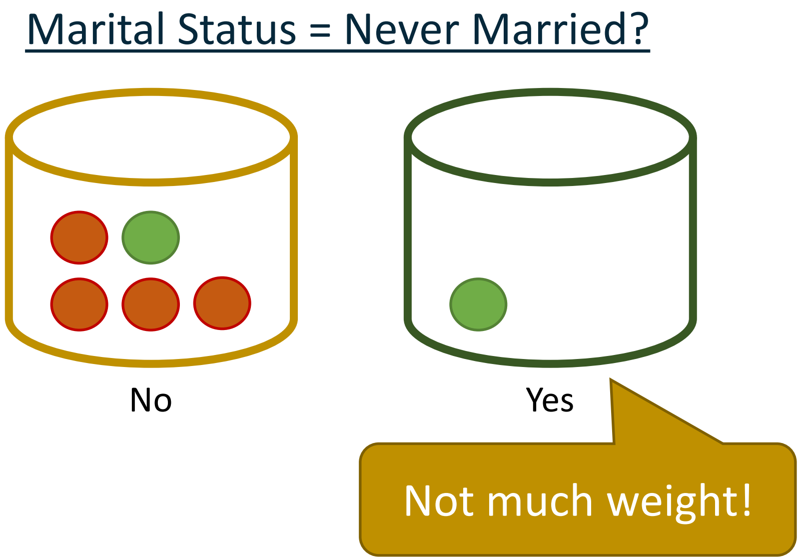
Another Example

Age	Education	Marital Status	Race	Sex	Hours Per Week	Label
53	Masters	Married-civ-spouse	White	Male	40	<=50K
49	HS-grad	Married-spouse-absent	White	Female	16	<=50K
52	HS-grad	Married-civ-spouse	Black	Male	45	>50K
31	Masters	Never-married	Black	Female	50	>50K
28	HS-grad	Married-civ-spouse	Black	Female	40	<=50K
39	Masters	Divorced	White	Male	45	<=50K

Let’s make things more interesting… More data and more details.

Gini Impurity

Age	Education	Marital Status	Race	Sex	Hours Per Week	Label
53	Masters	Married-civ-spouse	White	Male	40	<=50K
49	HS-grad	Married-spouse-absent	White	Female	16	<=50K
52	HS-grad	Married-civ-spouse	Black	Male	45	>50K
31	Masters	Never-married	Black	Female	50	>50K
28	HS-grad	Married-civ-spouse	Black	Female	40	<=50K
39	Masters	Divorced	White	Male	45	<=50K



Gini Impurity: The probability that we mislabel a data point. *Whoopsie!*

Before choosing a feature to split on, the tree runs through every feature (left to right) and calculates the gini for each split. *The tree will ultimately build the decision node using the feature that offers the lowest gini.*

Gini considers both the **purity** of the leaves (% of training observations with the same label) and the **weight** of the leaves (# of training observations dropped into each leaf) following a split.

Binning

Dave ON DATA

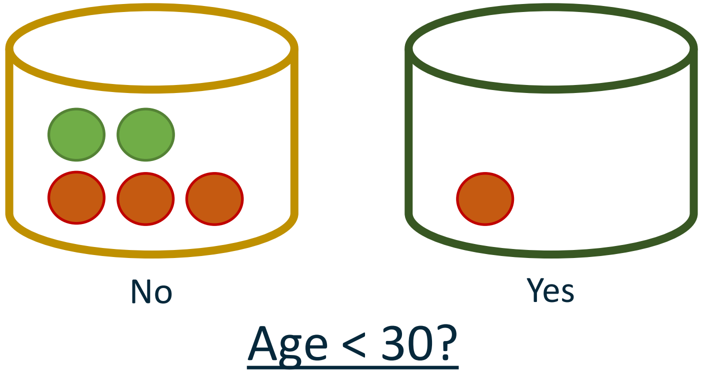
To calculate the gini offered by a continuous feature such as Age, the tree must first use a process called **Binning** to convert the numeric feature into multiple classes (e.g., Age < 30?).

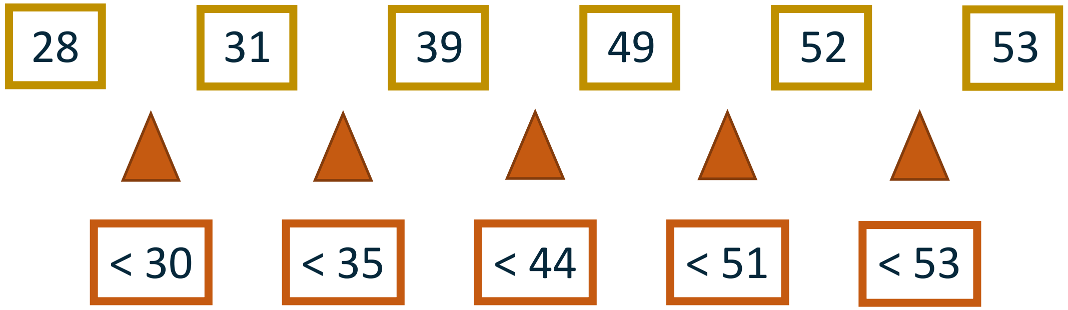
Age	Education	Marital Status	Race	Sex	Hours Per Week	Label
53	Masters	Married-civ-spouse	White	Male	40	<=50K
49	HS-grad	Married-spouse-absent	White	Female	16	<=50K
52	HS-grad	Married-civ-spouse	Black	Male	45	>50K
31	Masters	Never-married	Black	Female	50	>50K
28	HS-grad	Married-civ-spouse	Black	Female	40	<=50K
39	Masters	Divorced	White	Male	45	<=50K

Where is the bin threshold?

Split points – the midpoint between adjacent values (e.g., 30 is equidistant to 28 and 31).

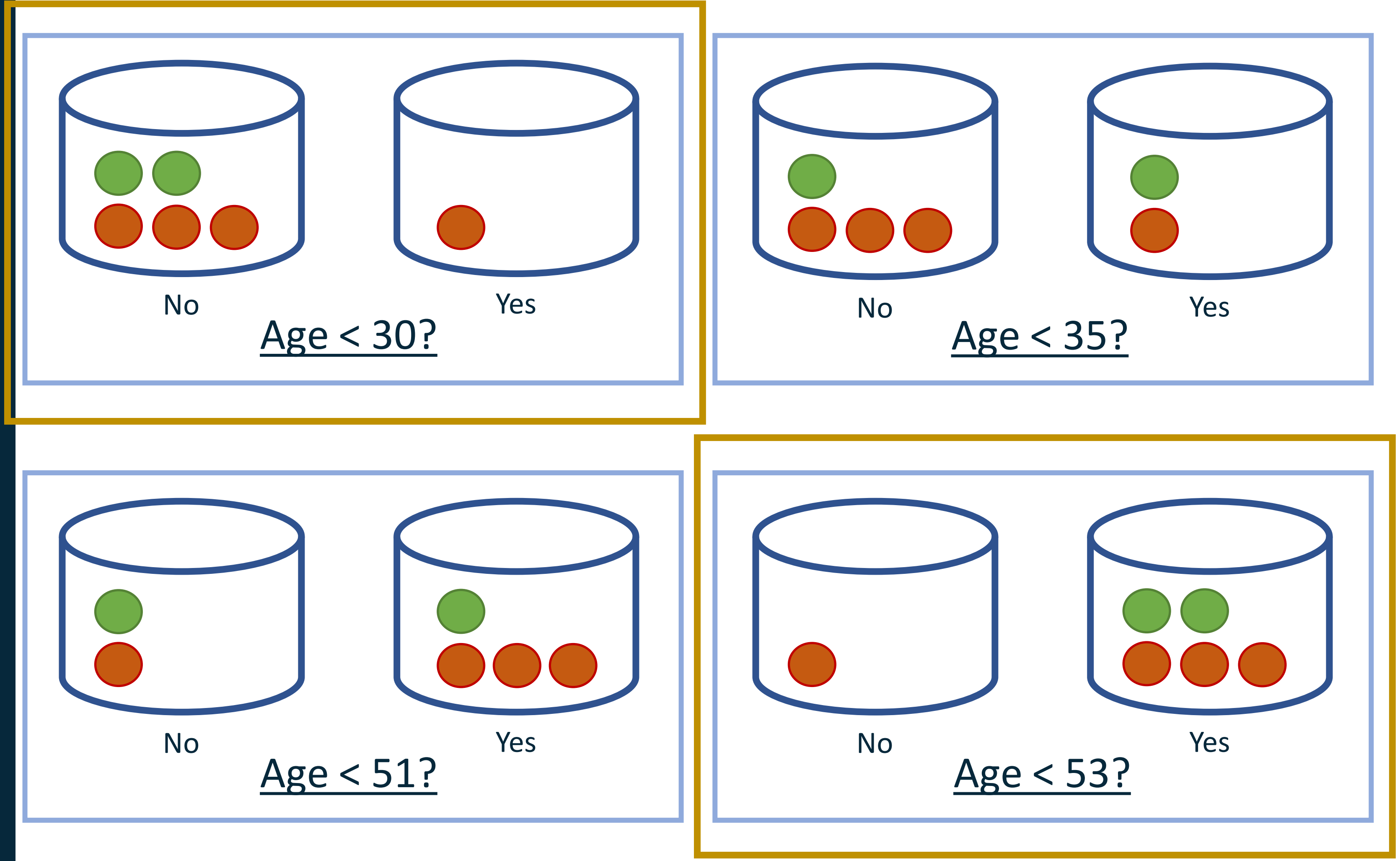
The tree calculates the gini associated with every split point and selects the value which gives the lowest gini.





Binning

Although you *can* manually create bins, decision trees are clever enough to handle the grunt work for you! They create optimal bins at *every* node in the tree.



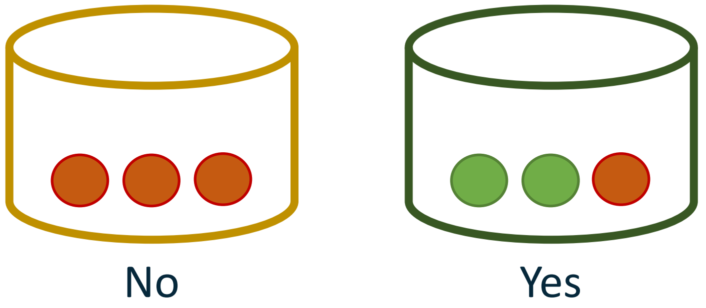
This gini is stored and compared against the gini indexes offered by all other features!

Age < 30 was the first optimal gini found for the Age feature.

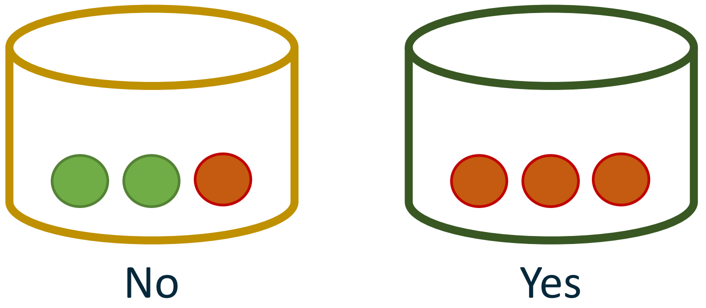
If this proves the lowest gini, the root node will be 'Age < 30?'

Moving On…

Age	Education	Marital Status	Race	Sex	Hours Per Week	Label
53	Masters	Married-civ-spouse	White	Male	40	<=50K
49	HS-grad	Married-spouse-absent	White	Female	16	<=50K
52	HS-grad	Married-civ-spouse	Black	Male	45	>50K
31	Masters	Never-married	Black	Female	50	>50K
28	HS-grad	Married-civ-spouse	Black	Female	40	<=50K
39	Masters	Divorced	White	Male	45	<=50K



Race = Black?



Hours Per Week < 43?

The tree calculates the gini offered by the remaining features, moving left to right.

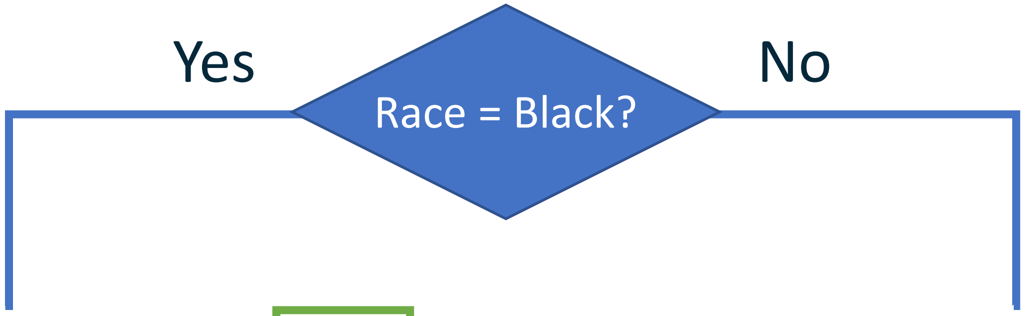
Splitting on Race doesn't offer a perfectly pure split, but when considering purity AND weight, it offers a low gini.

Hours Per Week offers the same low gini! Remember that trees are a GREEDY algorithm.

After comparing the gini indexes across all features, Race offered the first, lowest gini.

RACE WINS THE SPLIT!

The Root Node



52	HS-grad	Married-civ-spouse	Black	Male	45	>50K
31	Masters	Never-married	Black	Female	50	>50K
28	HS-grad	Married-civ-spouse	Black	Female	40	<=50K

Here we have multiple labels, so we'll continue to split

53	Masters	Married-civ-spouse	White	Male	40	<=50K
49	HS-grad	Married-spouse-absent	White	Female	16	<=50K
39	Masters	Divorced	White	Male	45	<=50K

Yay, we have a pure split! No need to continue splitting, so this becomes a leaf with a prediction label <=50K

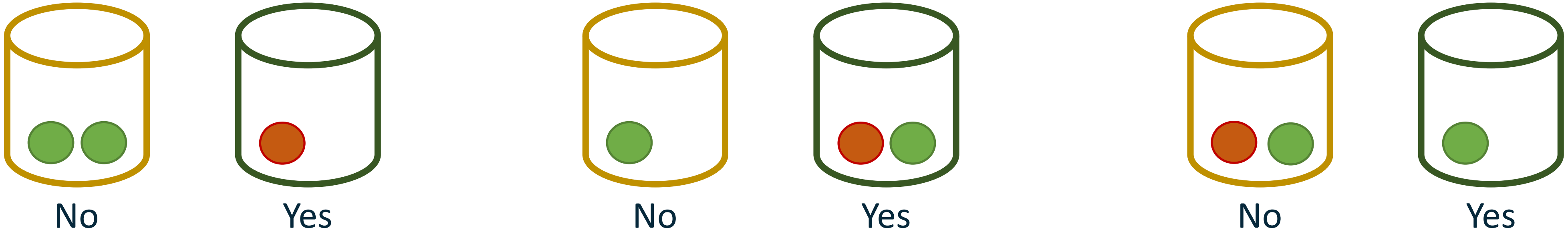
The Second Split

Age	Education	Marital Status	Race	Sex	Hours Per Week	Label
52	HS-grad	Married-civ-spouse		Male	45	>50K
31	Masters	Never-married		Female	50	>50K
28	HS-grad	Married-civ-spouse		Female	40	<=50K

Now the tree only considers those rows which flowed into the left side…

… and ignores the Race feature because it's been tapped.

Let's take it from the top!



Age < 30?

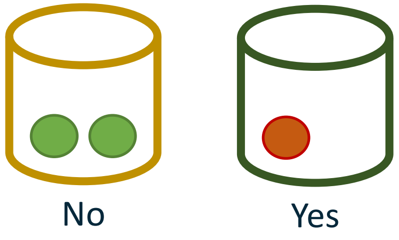
Education = HS-grad?

Marital Status = Never Married?

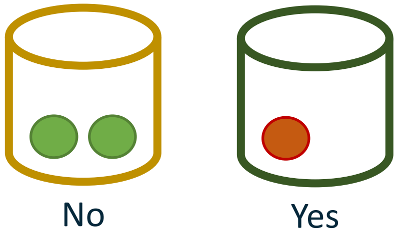
The Second Split

Age	Education	Marital Status	Race	Sex	Hours Per Week	Label
52	HS-grad	Married-civ-spouse		Male	45	>50K
31	Masters	Never-married		Female	50	>50K
28	HS-grad	Married-civ-spouse		Female	40	<=50K

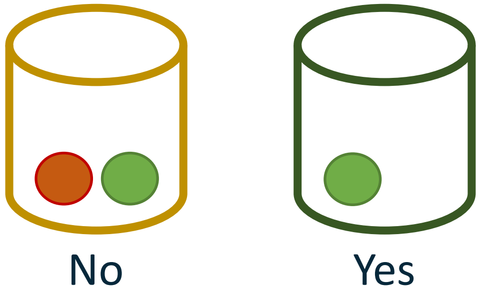
Age and Hours Per Week offer equally low gini.



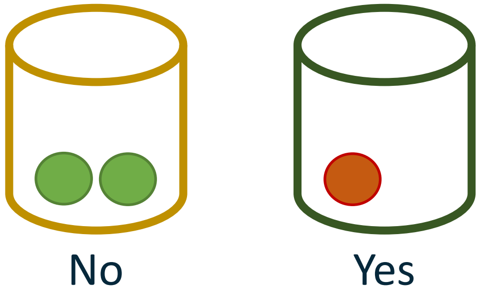
Age < 30?



Hours Per Week < 43?



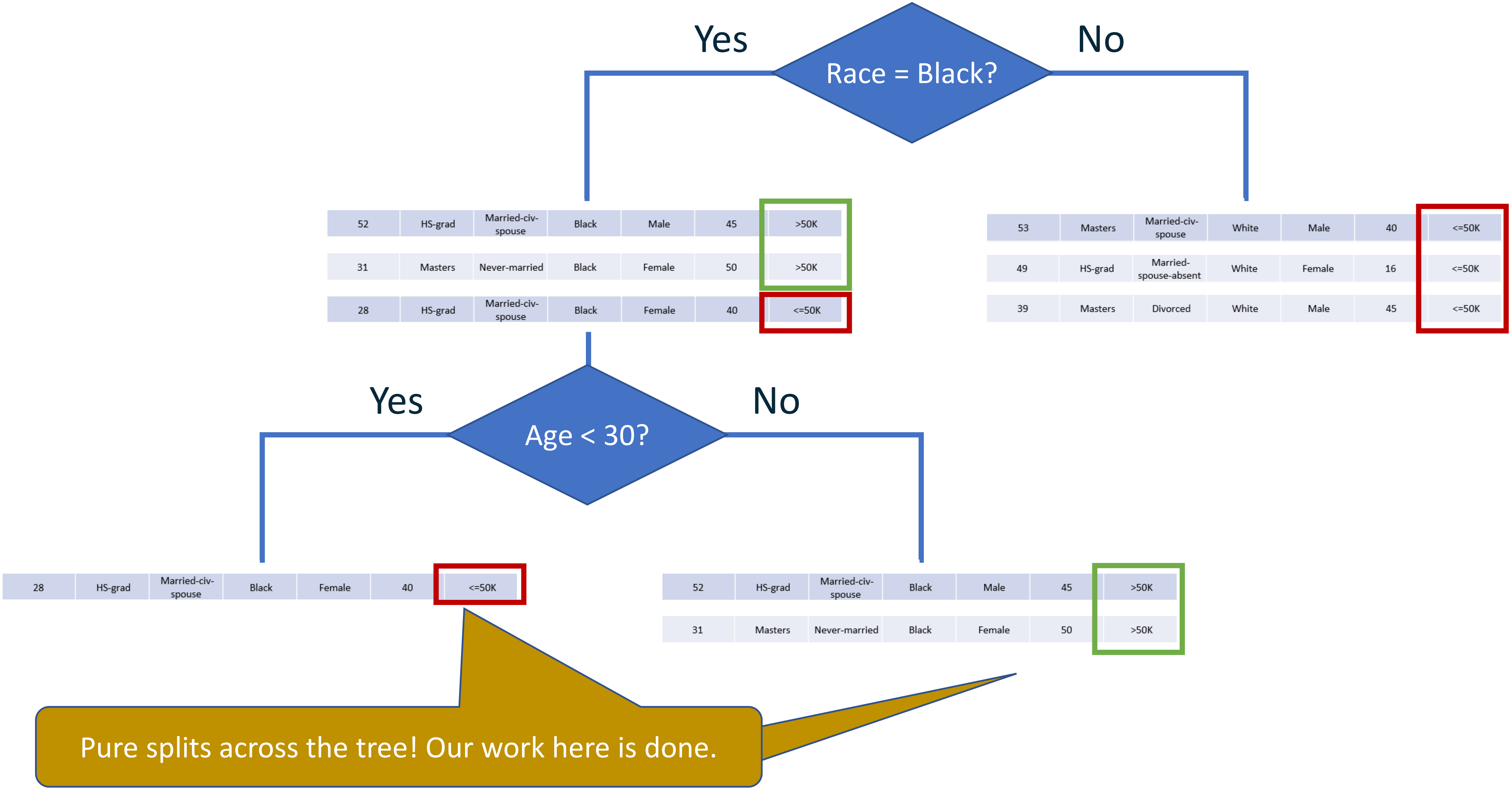
Sex = Female?



Hours Per Week < 43?

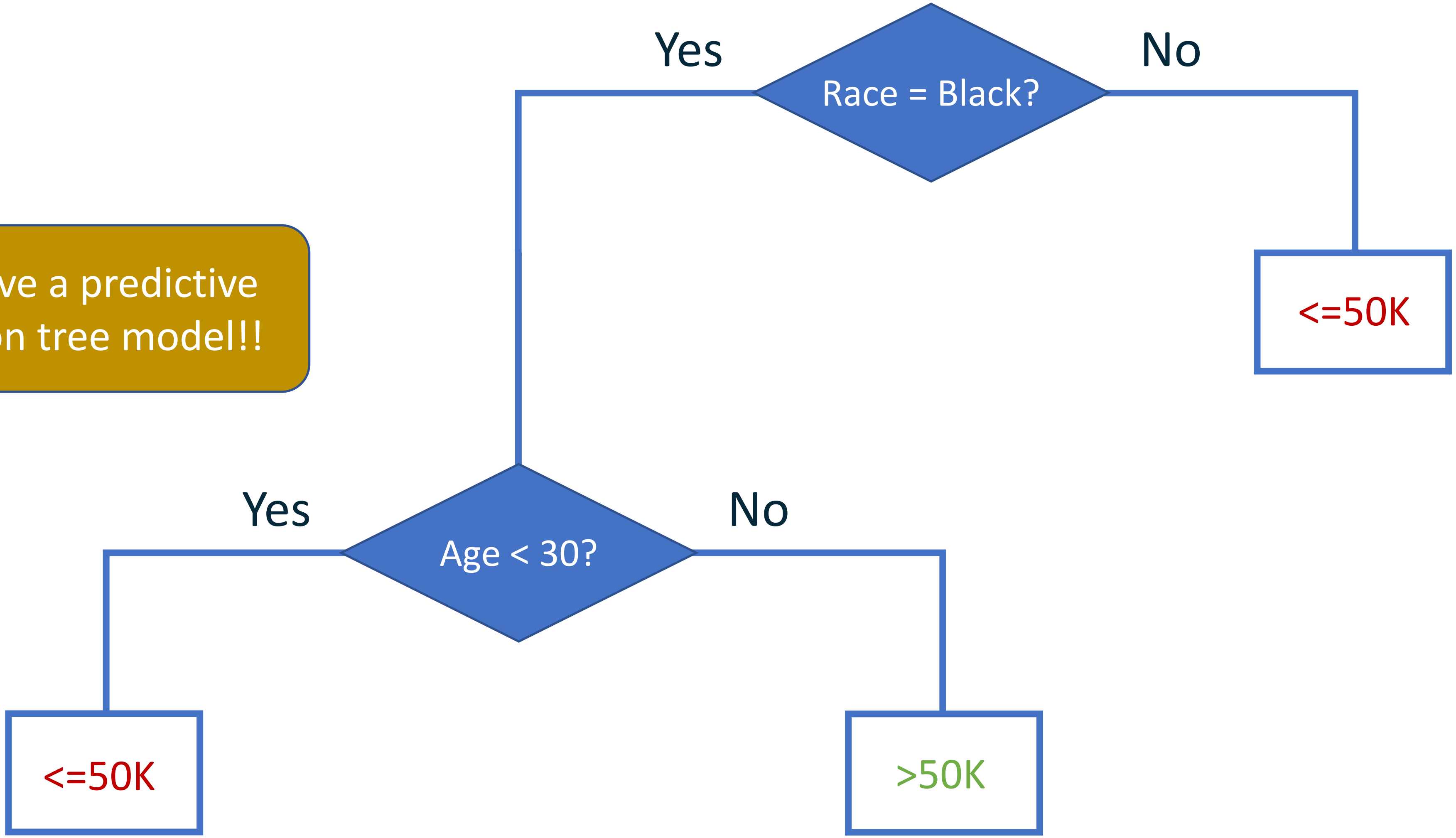
The tree *GREEDILY* opts to split on Age at the second node because it was the first optimal feature found.

The Second Split



The Decision Tree

We have a predictive decision tree model!!



Age	Education	Marital Status	Race	Sex	Hours Per Week	Label
49	Bachelors	Married-spouse-absent	Black	Female	35	?

>50K

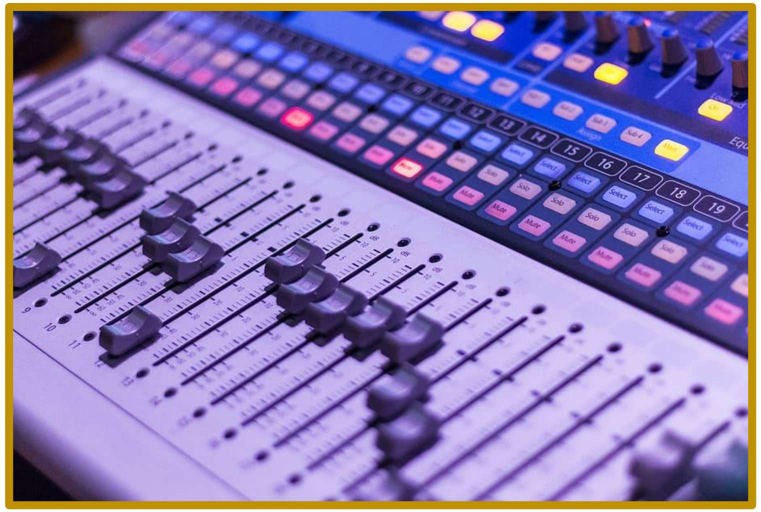
Stopping Conditions

When will they stop splitting?

- When the node is 100% pure
- When the remaining data has identical features but different class labels
- Based on *hyperparameters* – knobs and dials at your disposal

28	HS-grad	Married-civ-spouse	Black	Female	40	<=50K
28	HS-grad	Married-civ-spouse	Black	Female	40	>50K

😬 🤷‍♀️



SOME DECISION TREE HYPERPARAMETERS:

You can set thresholds for such things as:

- The min number of observations required to perform a split
- The min number of observations that fall into a leaf
- The min impurity decrease required to perform a split
- The max depth of the tree

The conditions controls if the tree continues splitting (i.e., growing).

Decision Trees in Python

Decision Trees in Python

We've been studying the Classification and Regression Tree (CART) algorithm so far.

The scikit-learn library offers the *DecisionTreeClassifier* class that is based on CART.

However, there are some differences between the *DecisionTreeClassifier* class and CART:

- The *DecisionTreeClassifier* only works with numeric data. Categorical features must be transformed (i.e., **encoded**) to a numeric form.

- In the case of a tie between features (i.e., they offer the same purity for a split), the *DecisionTreeClassifier* chooses between the tied features at **random**.

The easiest way to transform categorical features is using **one-hot encoding**.

One-Hot Encoding

Dave ON DATA

Consider the *Race* feature of the *Adult Census* dataset:

```
# Get the counts of categorical levels
adult_train['Race'].value_counts(dropna = False)
```

```
White                 13126
Black                  1219
Asian-Pac-Islander      462
Amer-Indian-Eskimo      116
Other                    93
Name: Race, dtype: int64
```

Include any missing data.

The Race categorical levels can be transformed (i.e., **encoded**) into a collection of exclusive binary indicators:

A feature for each categorical level

White	Black	Asian-Pac-Islander	Amer-Indian-Eskimo	Other
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

Race = Black

Race = White

One-Hot Encoding with Python

The *get_dummies()* function from the *pandas* library is the easiest way to one-hot encode categorical data:

```
# Use the get_dummies() function to one-hot encode
race_encoding = pd.get_dummies(adult_train['Race'])
race_encoding.head()
```

Level values become feature names

	Amer-Indian-Eskimo	Asian-Pac-Islander	Black	Other	White
0	0	0	0	0	1
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	0	1
4	0	0	0	0	1

One-Hot Encoding with Python

Dave ON DATA

```
# Use the get_dummies() function to one-hot encode with a prefix
race_encoding = pd.get_dummies(adult_train['Race'], prefix = 'Race')
race_encoding.head()
```

Prefix

	Race_Amer-Indian-Eskimo	Race_Asian-Pac-Islander	Race_Black	Race_Other	Race_White
0	0	0	0	0	1
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	0	1
4	0	0	0	0	1

NOTE – The default separator is an underscore, but you can specify the separator you want

One-Hot Encoding with Python

```
# Create a ist of feature to one-hot encode
cat_features = ['Race', 'Sex']

# One-hot encode multiple features
encoded_train = pd.get_dummies(adult_train, prefix = cat_features, columns = cat_features)
encoded_train.head()
```

Original features removed and one-hot encodings added to the end

NativeCountry	Label	Race_Amer-Indian-Eskimo	Race_Asian-Pac-Islander	Race_Black	Race_Other	Race_White	Sex_Female	Sex_Male
United-States	<=50K	0	0	0	0	1	1	0
United-States	<=50K	0	0	0	0	1	0	1
Mexico	<=50K	0	0	0	1	0	0	1
United-States	<=50K	0	0	0	0	1	1	0
United-States	<=50K	0	0	0	0	1	1	0

Preparing the Features

Dave ON DATA

The first step in performing machine learning with any technology is preparing the data.

When using *scikit-learn*, the convention is to create a *DataFrame* of the predictive features:

```
# Prepare data for machine learning
all_features = ['Age', 'EducationNum', 'MaritalStatus', 'HoursPerWeek']

# Select the above features and one-hot encode MaritalStatus
adult_X = pd.get_dummies(adult_train[all_features], prefix = 'MaritalStatus', columns = ['MaritalStatus'])
adult_X.head()
```

Age	EducationNum	HoursPerWeek	MaritalStatus_Divorced	MaritalStatus_Married-AF-spouse	MaritalStatus_Married-civ-spouse
20	12	40	0	0	0
22	10	40	0	0	0
63	1	30	0	0	1
32	13	42	1	0	0
36	10	40	0	0	0

Preparing the Labels

The labels of the *Adult Census* dataset are categorical string data.

When using *scikit-learn*, string labels need to be encoded using the *LabelEncoder* class:

```
from sklearn.preprocessing import LabelEncoder

# Encode labels
label_encoder = LabelEncoder()
adult_y = label_encoder.fit_transform(adult_train['Label'])

print(label_encoder.classes_)
print(adult_y)
```

```
['<=50K' '>50K']
[0 0 0 ... 1 1 1]
```

Training a Model

Dave ON DATA

Be default, the *DecisionTreeClassifier* class allows for huge decision trees to be built.

One of the easiest ways to control the size of the tree is to set a value for *min_samples_leaf*.

To ensure reproducibility, you can also set the *random_state* value.

```
from sklearn.tree import DecisionTreeClassifier

# Train a CART-like classification tree
tree_1 = DecisionTreeClassifier(min_samples_leaf = 3000, random_state = 12345)

tree_1.fit(adult_X, adult_y)
```

Leaves must have at least 3000 observations

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(min_samples_leaf=3000, random_state=12345)
```

Visualizing the Model

Dave ON DATA

The *plot_tree()* function from scikit-learn can be used to visualize a *DecisionTreeClassifier:*

NOTE – Large tree do not visualize well!

```
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# Set the size of the tree visual to be 12 by 12 inches
plt.figure(figsize=(12,12))

# Create a visual representions of the tree
plot_1 = plot_tree(tree_1, feature_names = list(adult_X.columns), fontsize = 14,
                   class_names = list(label_encoder.classes_), filled = True)
```

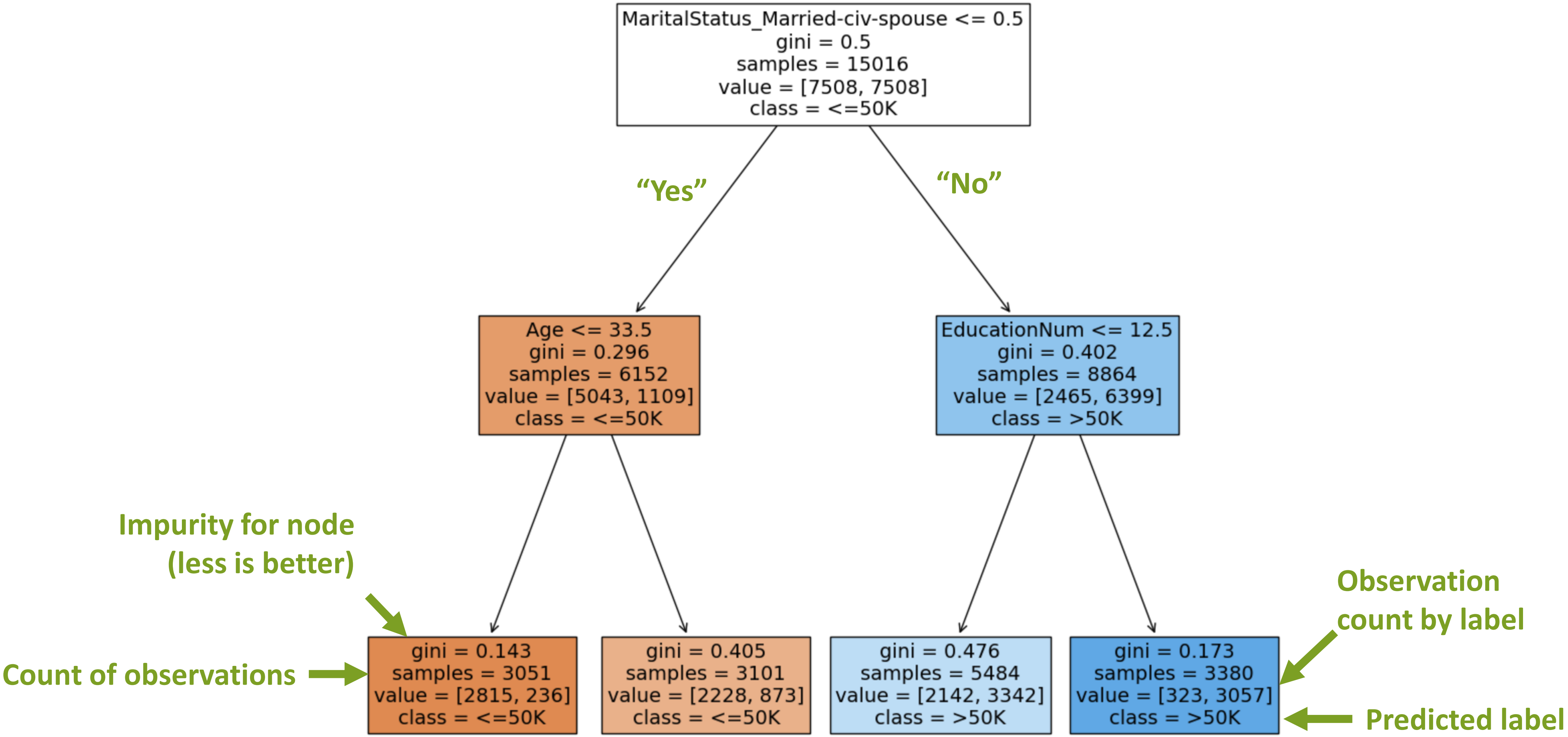
Use feature names in visual

The model

Use original label names in visual

Color code the visual

Visualizing the Model



Did the Model Learn?

Imagine a course at university where the final exam counts for 100% of the grading.

In machine learning, the test dataset is this kind of final exam (i.e., you only get **<u>one</u>** try)!

```
# Use the same training features and one-hot encode
adult_test_X = pd.get_dummies(adult_test[all_features],
                              prefix = 'MaritalStatus',
                              columns = ['MaritalStatus'])
adult_test_X.head()
```

Age	EducationNum	HoursPerWeek	MaritalStatus_Divorced	MaritalStatus_Married-AF-spouse	MaritalStatus_Married-civ-spouse
32	13	40	0	0	0
34	10	32	0	0	1
20	10	40	0	0	0
36	4	35	0	0	1
73	9	35	1	0	0

Making Predictions

```
# Encode the labels of the test dataset
adult_test_y = label_encoder.transform(adult_test['Label'])

# Make predictions on the test dataset
test_preds = tree_1.predict(adult_test_X)

print(test_preds)
```

```
[0 1 0 ... 1 1 0]
```

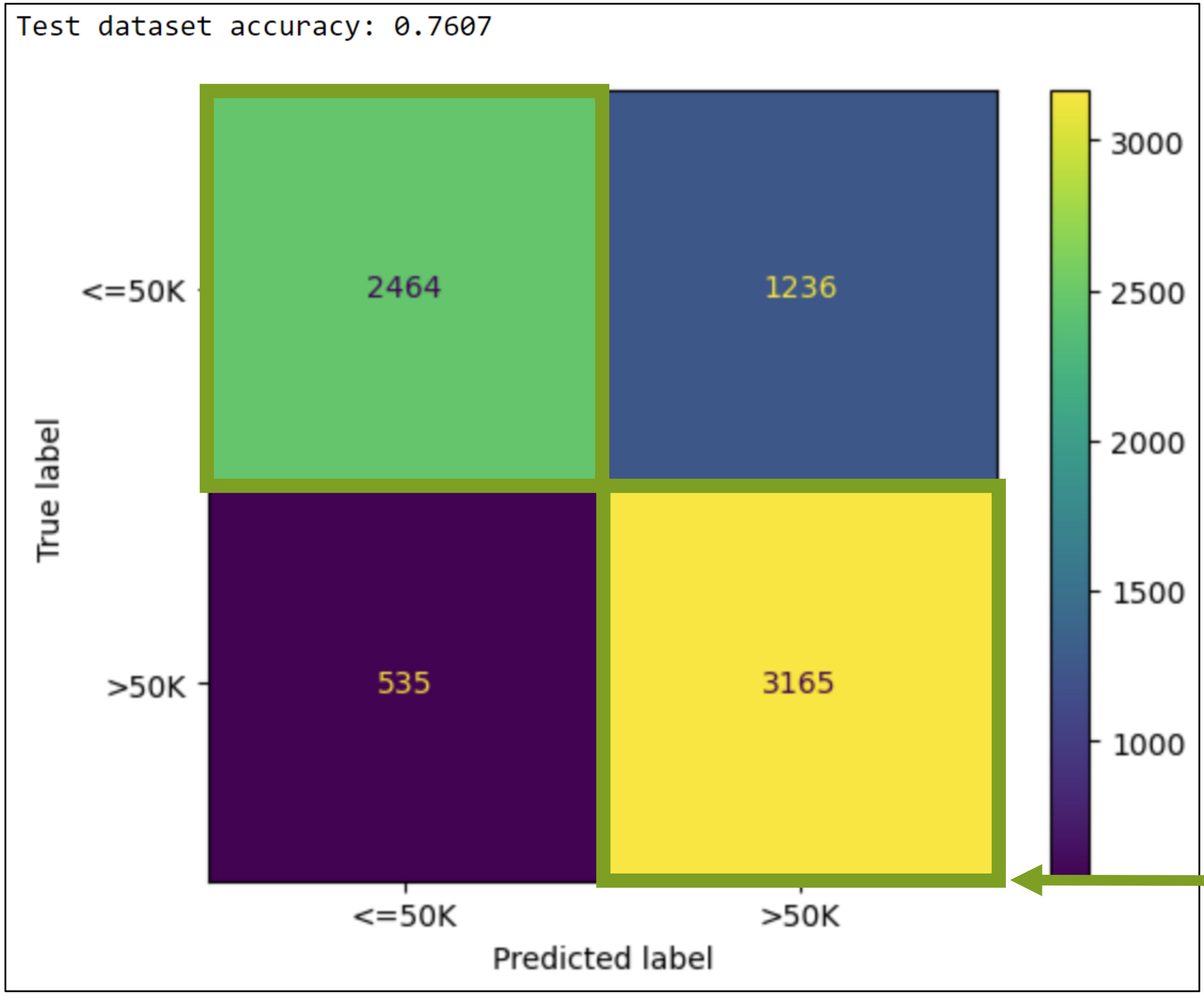
How Well Did the Model Learn?

```python
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# What is the overall accuracy on the test dataset?
print(f'Test dataset accuracy: {tree_1.score(adult_test_X, adult_test_y):.4f}')

# Display a confusion matrix
cm = confusion_matrix(adult_test_y, test_preds)
cmd = ConfusionMatrixDisplay(cm, display_labels = label_encoder.classes_)
cmd.plot();
```

How Well Did the Model Learn?



Correct predictions