Decision Trees and Random Forests

Outline

- Build a decision tree manually
- Look at single and collective impurity measures
- Selecting splitting attributes and test conditions
- Scikit-learn implementation
- Model training and evaluation
- Bias and Variance
- Random forests

Introduction

We'll now start looking into how to build models to predict an outcome variable from labeled data.

Classification problems:

- predict a category
- e.g., spam/not spam, fraud/not fraud, default/not default, malignant/benign, etc.

Regression problems:

- predict a numeric value
- e.g., price of a house, salary of a person, etc.

Loan Default Example

We'll use an example from (Tan et al. 2018).

You are a loan officer at **Terrier Savings and Loan**.

You have a dataset on loans that you have made in the past.

You want to build a model to predict whether a loan will default.

Loans Data Set

ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125000	No
2	No	Married	100000	No
3	No	Single	70000	No
4	Yes	Married	120000	No
5	No	Divorced	95000	Yes
6	No	Married	60000	No
7	Yes	Divorced	220000	No
8	No	Single	85000	Yes
9	No	Married	75000	No
10	No	Single	90000	Yes

Loans Data Set Summary

Here's the summary info of the data set.

▶ Code

```
<class 'pandas.DataFrame'>
RangeIndex: 10 entries, 1 to 10
Data columns (total 4 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Home Owner          10 non-null     str
 1   Marital Status      10 non-null     str
 2   Annual Income       10 non-null     int64
 3   Defaulted Borrower  10 non-null     str
dtypes: int64(1), str(3)
memory usage: 452.0 bytes
```

Convert to Categorical Data Types

Since some of the fields are categorical, let's convert them to categorical data types.

```
1  loans['Home Owner'] = loans['Home Owner'].astype('category')
2  loans['Marital Status'] = loans['Marital Status'].astype('category')
3  loans['Defaulted Borrower'] = loans['Defaulted Borrower'].astype('category')
4  loans.info()
```

```
<class 'pandas.DataFrame'>
RangeIndex: 10 entries, 1 to 10
Data columns (total 4 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Home Owner          10 non-null     category
 1   Marital Status      10 non-null     category
 2   Annual Income       10 non-null     int64
 3   Defaulted Borrower  10 non-null     category
dtypes: category(3), int64(1)
memory usage: 298.0 bytes
```

Simple Model

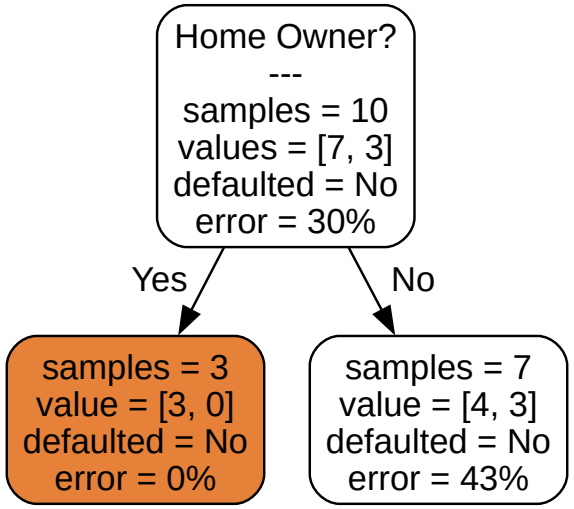
Looking at the table, let's just start with the simplest model possible and just predict that no one will default.

So the output of our model is just to always predict "No".

values = [# No, # Yes] =[7,3]
defaulted = No
error = 30%

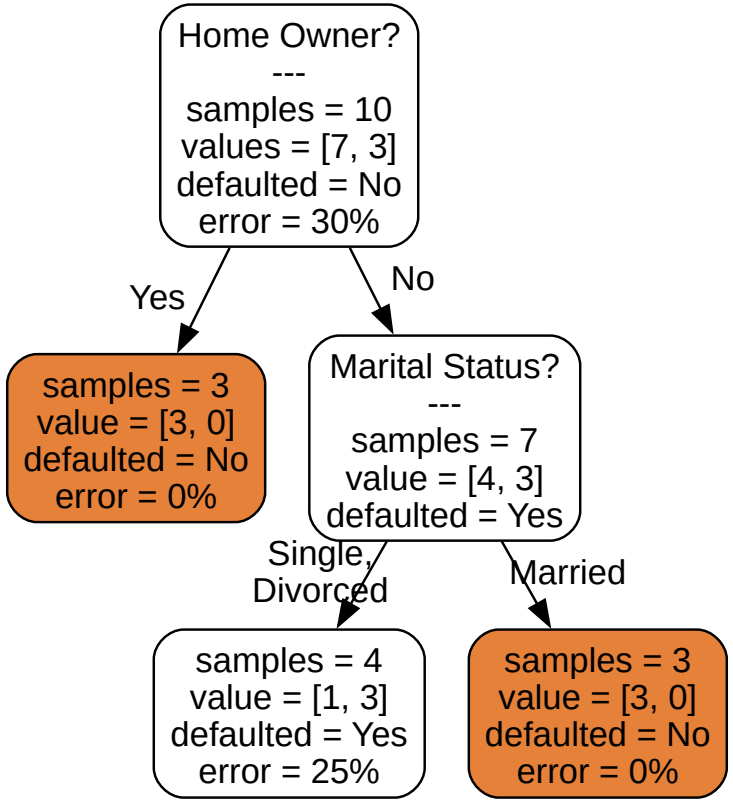
We see a 30% error rate since 3 out of 10 loans defaulted.

Let’s split the data based on the “Home Owner” field. (`values = [# No, # Yes]`).



Let’s split on the “Marital Status” field.

We see that the 3 defaulted loans are all for single or divorced people. Since the node is all one class, we don’t split this node and we call it a **leaf node**.



We can list the subsets for the two criteria to calculate the error rate.

▶ Code

ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
3	No	Single	70000	No
5	No	Divorced	95000	Yes
8	No	Single	85000	Yes
10	No	Single	90000	Yes

Table: Home Owner == No and Marital Status == Single or Divorced –> Defaulted == Yes

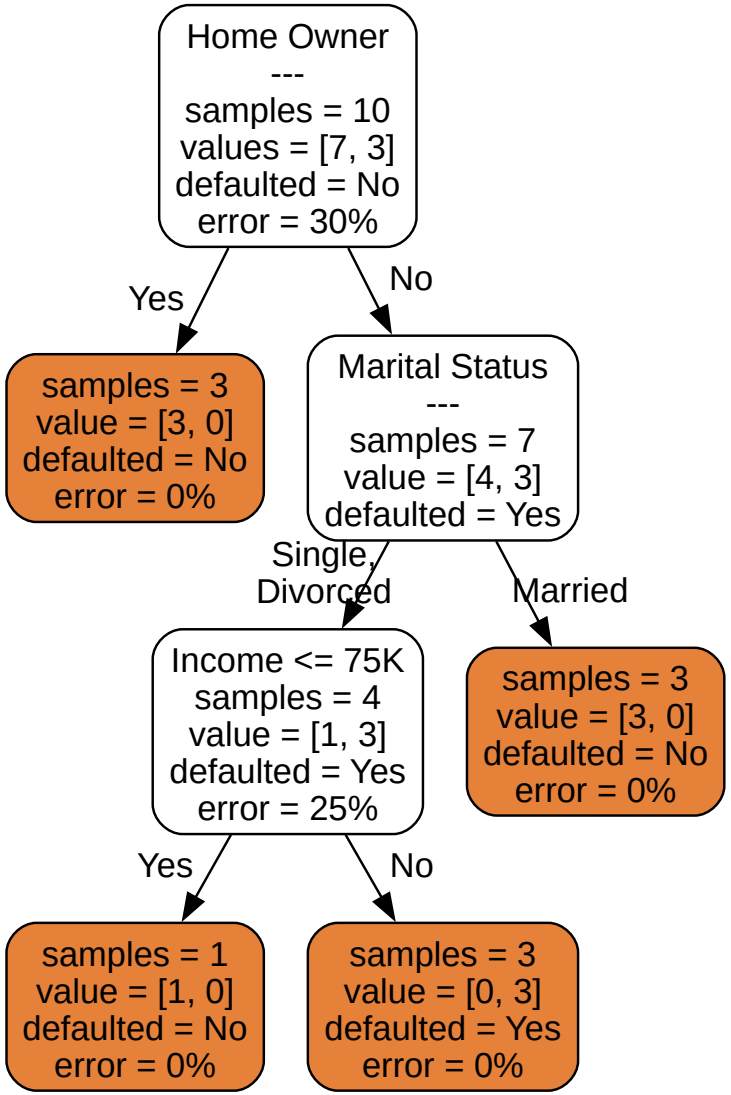
and…

▶ Code

ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
2	No	Married	100000	No
6	No	Married	60000	No
9	No	Married	75000	No

Table: Home Owner == No and Marital Status == Married –> Defaulted == No

- Let’s try to split on the “Annual Income” field.
- We see that the person with income of 70K doesn’t default, so we split the node into two nodes based on the “Income” field.
- We arbitrarily pick a threshold of $75K.



Evaluating the Model

Specifying the Test Condition

Before we continue, we should take a moment to consider how we specify a test condition of a node.

How we specify a test condition depends on the attribute type which can be:

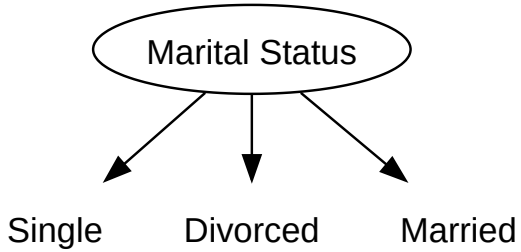
- Binary (Boolean)
- Nominal (Categorical, e.g., cat, dog, bird)
- Ordinal (e.g., Small, Medium, Large)
- Continuous (e.g., 1.5, 2.1, 3.7)

And depends on the number of ways to split:

- **multi-way**
- **binary**

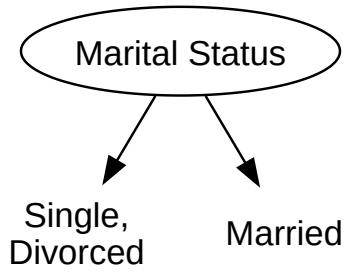
For a **Nominal (Categorical)** attribute:

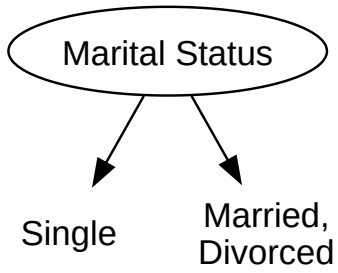
- In a **Multi-way split** we can use as many partitions as there are distinct values of the attribute:

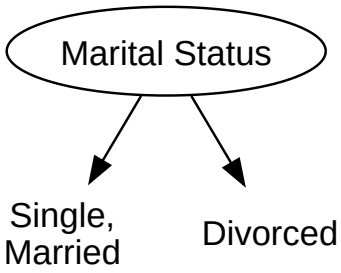


For a **Nominal (Categorical)** attribute:

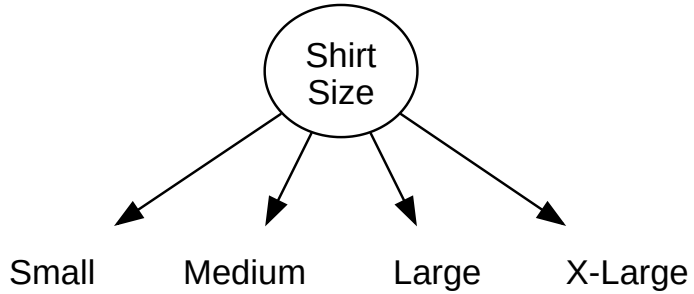
- In a **Binary split** we divide the values into two groups.
- In this case, we need to find an optimal partitioning of values into groups, which we discuss shortly.



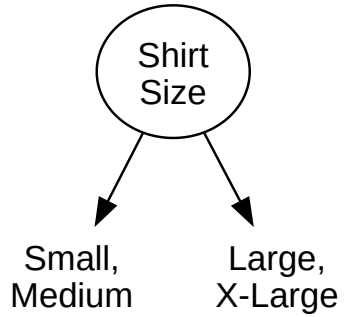


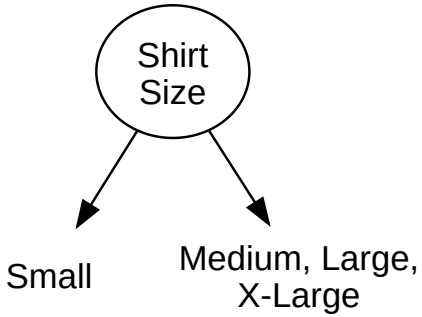


For an **Ordinal** attribute, we can use a multi-way split with as many partitions as there are distinct values.



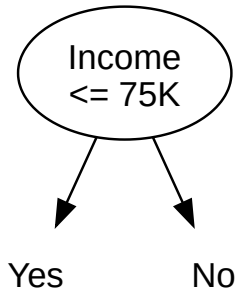
Or we can use a binary split as long we preserve the ordering of the values.



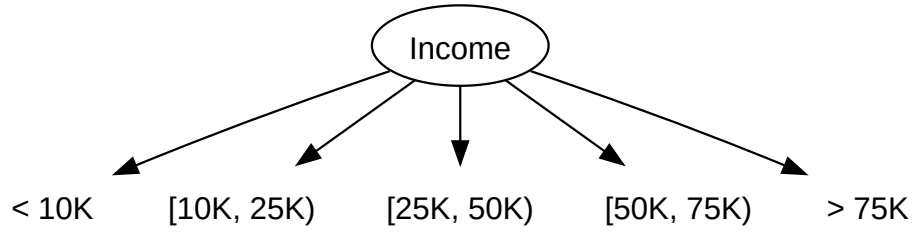


> ⚠ **Warning**
>
> Be careful not to violate the ordering of values such as {Small, Large} and {Medium, X-Large}.

A **Continuous** attribute can be handled two ways:



It can be thresholded to form a binary split.



Or it can be split into contiguous ranges to form an ordinal categorical attribute.

Selecting Attribute and Test Condition

Impurity Measures

The following are three impurity indices:

$$\text{Gini index} = 1 - \sum_{i=0}^{c-1} p_i(t)^2$$

$$\text{Entropy} = -\sum_{i=0}^{c-1} p_i(t) \log_2 p_i(t)$$

$$\text{Classification error} = 1 - \max_i p_i(t)$$

where $p_i(t)$ is the **relative frequency** of training instances of class i at a node t and c is the number of classes.

> **Note**
>
> By convention, we set $0 \log_2 0 = 0$ in entropy calculations.

Impurity Measures

The following are three impurity indices:

$$\text{Gini index} = 1 - \sum_{i=0}^{c-1} p_i(t)^2$$

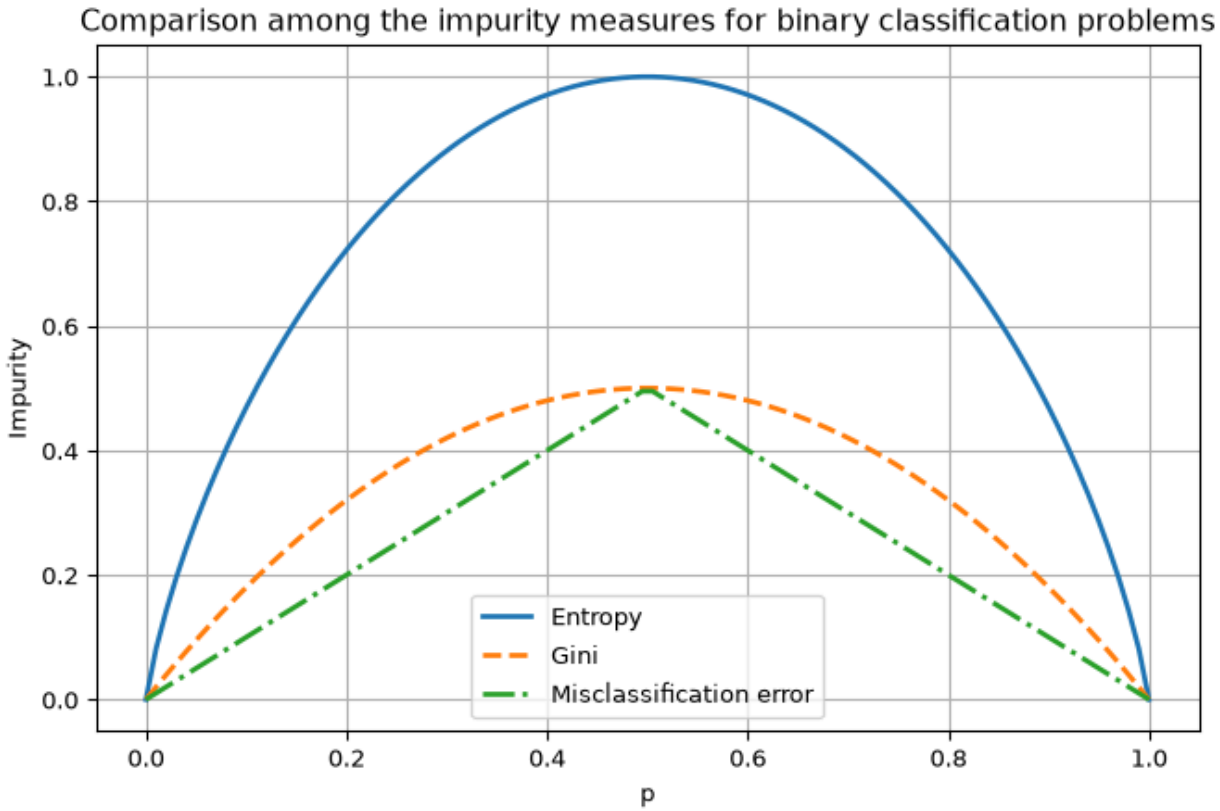
$$\text{Entropy} = -\sum_{i=0}^{c-1} p_i(t) \log_2 p_i(t)$$

$$\text{Classification error} = 1 - \max_i p_i(t)$$

All three impurity indices equal 0 when all the records at a node belong to the same class.

All three impurity indices reach their maximum value when the classes are evenly distributed among the child nodes.

We can plot the three impurity indices to get a sense of how they behave for **binary classification** problems.



They all maintain the same ordering for every relative frequency, i.e., Entropy > Gini > Misclassification error.

Impurity Example 1

Node N_1	Count	p
Class=0	0	$0/6 = 0$
Class=1	6	$6/6 = 1$

$$\text{Gini} = 1 - \left(\frac{0}{6}\right)^2 - \left(\frac{6}{6}\right)^2 = 0$$

$$\text{Entropy} = -\left(\frac{0}{6}\log_2\frac{0}{6} + \frac{6}{6}\log_2\frac{6}{6}\right) = 0$$

$$\text{Error} = 1 - \max\left[\frac{0}{6}, \frac{6}{6}\right] = 0$$

Impurity Example 2

Node N_2	Count	p
Class=0	1	$1/6 = 0.167$
Class=1	5	$5/6 = 0.833$

$$\text{Gini} = 1 - \left(\frac{1}{6}\right)^2 - \left(\frac{5}{6}\right)^2 = 0.278$$

$$\text{Entropy} = -\left(\frac{1}{6}\log_2\frac{1}{6} + \frac{5}{6}\log_2\frac{5}{6}\right) = 0.650$$

$$\text{Error} = 1 - \max\left[\frac{1}{6}, \frac{5}{6}\right] = 0.167$$

Impurity Example 3

Node N_3	Count
Class=0	3
Class=1	3

$$\text{Gini} = 1 - \left(\frac{3}{6}\right)^2 - \left(\frac{3}{6}\right)^2 = 0.5$$

$$\text{Entropy} = -\left(\frac{3}{6}\log_2\frac{3}{6} + \frac{3}{6}\log_2\frac{3}{6}\right) = 1$$

$$\text{Error} = 1 - \max\left[\frac{3}{6}, \frac{3}{6}\right] = 0.5$$

Impurity Class Exercise

Calculate Gini impurity for the following node:

Node N_4	Count	p
Class=0	3	
Class=1	6	
Class=2	1	

$$\text{Gini} =$$

Collective Impurity of Child Nodes

We can compute the collective impurity of child nodes by taking a weighted sum of the impurities of the child nodes.

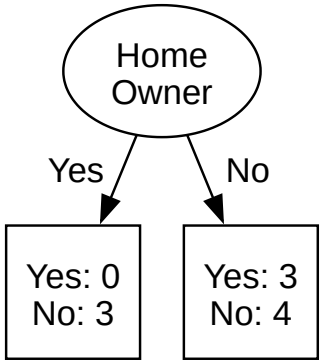
$$I(\text{children}) = \sum_{j=1}^{k} \frac{N(v_j)}{N} \, I(v_j)$$

Here we split N training instances into k child nodes, v_j for $j = 1, \ldots, k$.

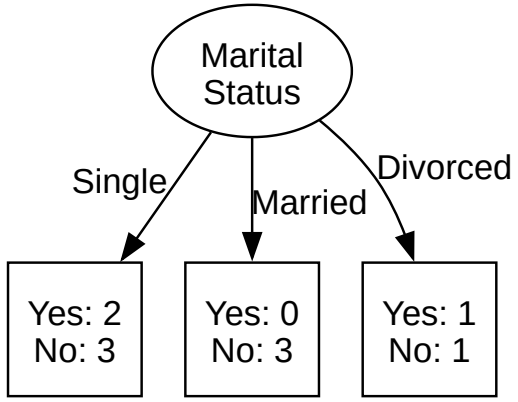
$N(v_j)$ is the number of training instances at child node v_j and $I(v_j)$ is the impurity at child node v_j.

Impurity Example

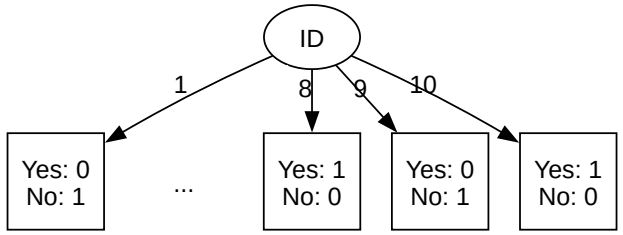
Let's compute collective impurity on our loans dataset to see which feature to split on.



(a) Collective Entropy: 0.690



(b) Collective Entropy: 0.686



(c) Collective Entropy index: 0.00

Tip

Try calculating the collective Entropy for (a) and (b) and see if you get the same values.

Important

The collective entropy for (c) is 0. Why would we not want to use this node?

There are two ways to overcome this problem.

1. One way is to *generate only binary decision trees*, thus avoiding the difficulty of handling attributes with varying number of partitions. This strategy is employed by decision tree classifiers such as **CART**.
2. Another way is to modify the splitting criterion to take into account the number of partitions produced by the attribute. For example, in the **C4.5** decision tree algorithm, a measure known as **gain ratio** is used to compensate for attributes that produce a large number of child nodes.

CART stands for Classification And Regression Tree.

Gain Ratio

See (Tan et al. 2018, chap. 3, p. 127):

- Having a low impurity value alone is insufficient to find a good attribute test condition for a node.
- Having more child nodes can make a decision tree more complex and consequently more susceptible to overfitting.

Hence, the number of children produced by the splitting attribute should also be taken into consideration while deciding the best attribute test condition.

Gain Ratio Formula

$$\text{Gain ratio} = \frac{\Delta_{\text{info}}}{\text{Split Info}} = \frac{\text{Entropy(Parent)} - \sum_{i=1}^{k} \frac{N(v_i)}{N} \text{Entropy}(v_i)}{-\sum_{i=1}^{k} \frac{N(v_i)}{N} \log_2 \frac{N(v_i)}{N}}$$

where $N(v_i)$ is the number of instances assigned to node v_i and k is the total number of splits.

The split information measures the entropy of splitting a node into its child nodes and evaluates if the split results in a larger number of equally-sized child nodes or not.

Gain Ratio Example

Let's compare the Gain Ratio for **Home Owner** and **Marital Status** attributes using the loans dataset.

Recall from the earlier example that the parent node has 3 Yes and 7 No defaulters (10 total instances).

Parent node entropy:

$$\text{Entropy}(\text{Parent}) = -\frac{3}{10}\log_2\frac{3}{10} - \frac{7}{10}\log_2\frac{7}{10} = 0.881$$

Ex: Gain Ratio for Home Owner Attribute

The Home Owner attribute splits the data into 2 child nodes:

- Yes: 0 Yes, 3 No (3 instances)
- No: 3 Yes, 4 No (7 instances)

From the earlier calculation, the Information Gain is:

$$\Delta_{\text{info}} = 0.881 - \left(\frac{3}{10} \times 0 + \frac{7}{10} \times 0.985\right) = 0.192$$

Now we calculate the Split Information:

$$\text{Split Info} = -\frac{3}{10}\log_2\frac{3}{10} - \frac{7}{10}\log_2\frac{7}{10} = 0.881$$

Therefore. the Gain Ratio is:

Ex: Gain Ratio for Marital Status Attribute

The Marital Status attribute splits the data into 3 child nodes:

- Single: 2 Yes, 3 No (5 instances)
- Married: 0 Yes, 3 No (3 instances)
- Divorced: 1 Yes, 1 No (2 instances)

From the earlier calculation, the Information Gain is:

$$\Delta_{\text{info}} = 0.881 - \left(\frac{5}{10} \times 0.971 + \frac{3}{10} \times 0 + \frac{2}{10} \times 1\right) = 0.195$$

Now we calculate the Split Information:

$$\text{Split Info} = -\frac{5}{10}\log_2\frac{5}{10} - \frac{3}{10}\log_2\frac{3}{10} - \frac{2}{10}\log_2\frac{2}{10} = 1.486$$

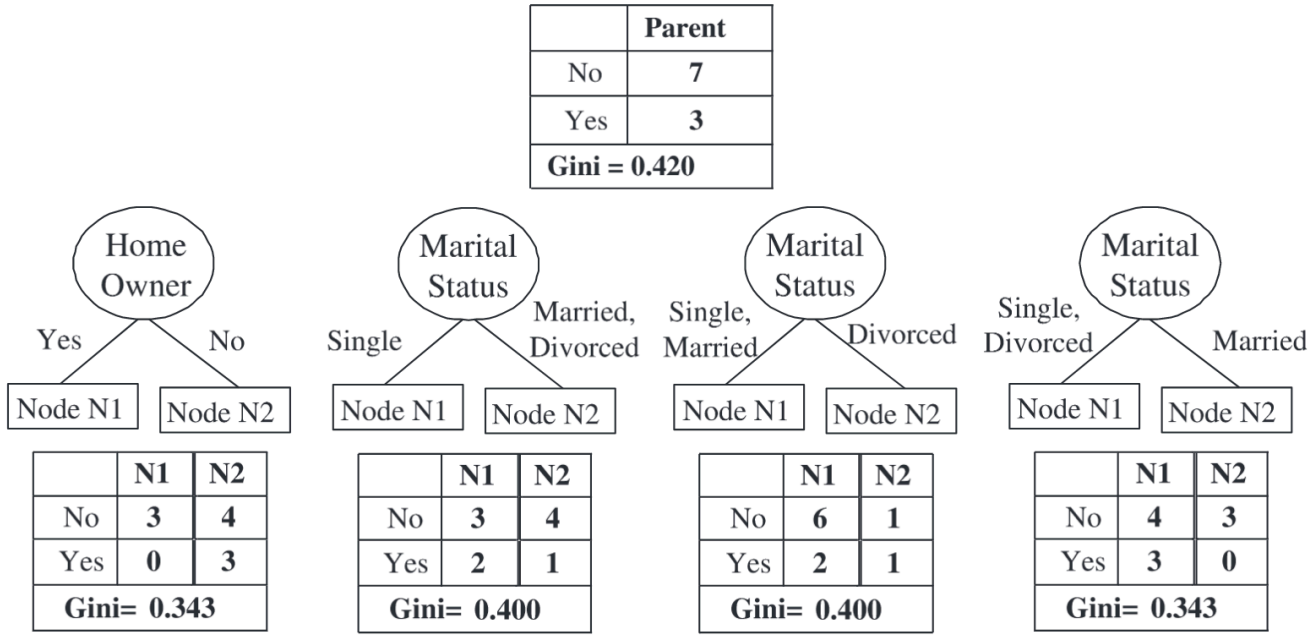
Example: Comparison of Gain Ratios

Attribute	Information Gain	Split Info	Gain Ratio
Home Owner	0.192	0.881	**0.218**
Marital Status	0.195	1.486	0.131

Although Marital Status has a slightly higher Information Gain, it has a **lower Gain Ratio** because it splits the data into 3 child nodes rather than 2.

The Gain Ratio penalizes attributes that create more splits, making **Home Owner** the preferred choice.

Identifying the Best Attribute Test Condition



Here's an example of how to identify the best attribute test condition using the Collective Impurity of the Gini Impurity index.

▶ Code

Splitting Continuous Attributes

For quantitative attributes like *Annual Income*, we need to find some threshold τ that minimizes the impurity index.

The following table illustrates the process.

Class	No	No	No	Yes	Yes	Yes	No	No	No	No
	Annual Income (in '000s)									
Sorted Values →	60	70	75	85	90	95	100	120	125	220
Split Positions →	55	65	72.5	80	87.5	92.5	97.5	110	122.5	172.5	230
	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >
Yes	0 3	0 3	0 3	0 3	1 2	2 1	3 0	3 0	3 0	3 0	3 0
No	0 7	1 6	2 5	3 4	3 4	3 4	3 4	4 3	5 2	6 1	7 0
Gini	0.420	0.400	0.375	0.343	0.417	0.400	***0.300***	0.343	0.375	0.400	0.420

Procedure:

1. Sort all the training instances by *Annual Income* in increasing order.

Run Decision Tree on Loans Data Set

Let's run the Scikit-learn Decision Tree, `sklearn.tree`, on the loans data set.

`sklearn.tree` requires all fields to be numeric.

So first we have to convert the categorical fields to category index numeric fields.

```
loans['Defaulted Borrower'] = loans['Defaulted Borrower'].cat.codes
loans['Home Owner'] = loans['Home Owner'].cat.codes
loans['Marital Status'] = loans['Marital Status'].cat.codes
loans.head()
```

	Home Owner	Marital Status	Annual Income	Defaulted Borrower
ID				
1	1	2	125000	0
2	0	1	100000	0
3	0	2	70000	0
4	1	1	120000	0
5	0	0	95000	1

Then the independent variables are all the fields except the “Defaulted Borrower” field, which we’ll assign to X.

The dependent variable is the “Defaulted Borrower” field, which we’ll assign to y.

```
1 from sklearn import tree
2
3 X = loans.drop('Defaulted Borrower', axis=1)
4 y = loans['Defaulted Borrower']
```

X:

```
<class 'pandas.DataFrame'>
RangeIndex: 10 entries, 1 to 10
Data columns (total 3 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Home Owner      10 non-null     int8
 1   Marital Status  10 non-null     int8
 2   Annual Income   10 non-null     int64
dtypes: int64(1), int8(2)
memory usage: 232.0 bytes
```

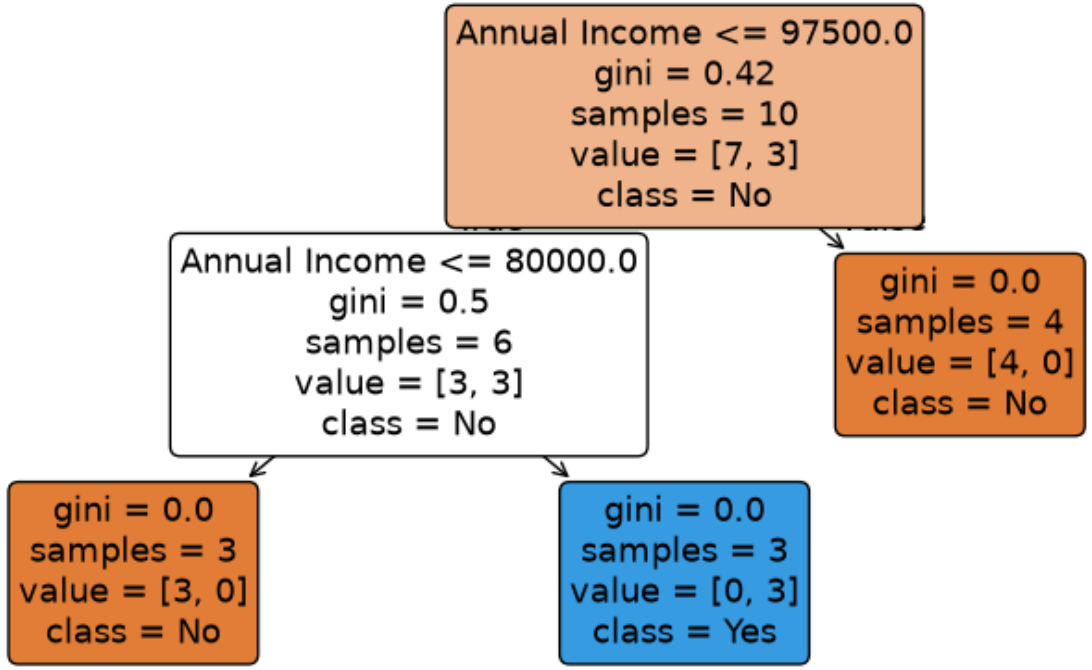
y:

```
<class 'pandas.Series'>
RangeIndex: 10 entries, 1 to 10
Series name: Defaulted Borrower
Non-Null Count  Dtype
--------------  -----
10 non-null     int8
dtypes: int8(1)
memory usage: 142.0 bytes
```

Let’s fit a decision tree to the data.

```
clf = tree.DecisionTreeClassifier(criterion='gini', random_state=42)
clf = clf.fit(X, y)
```

Let’s plot the tree.



Interestingly, the tree was built using only the Income field.

That’s arguably an advantage of Decision Trees: they automatically perform feature

Ensemble Methods

(See Tan et al. (2018), Chapter 4)

Motivated around the idea that combining several noisy classifiers can result in a better prediction under certain conditions.

- The base classifiers are independent
- The base classifiers are noisy (high variance)
- The base classifiers are low (ideally zero) bias

Bias and Variance

Recall bias and variance from Generalization: **bias** is error from a model too simple to capture the pattern (a shallow tree underfits); **variance** is error from a model so flexible that it changes with every training sample (a deep tree overfits).

A fully grown tree is low bias, high variance. Averaging many such trees keeps the low bias and cancels out the variance – **ensembles reduce variance.**

Random Forests

Random forests are an ensemble of decision trees that:

- Construct a set of base classifiers from random sub-samples of the training data.
- Train each base classifier to completion.
- Take a majority vote of the base classifiers to form the final prediction.

Titanic Example

We'll use the Titanic data set and excerpts of this Kaggle tutorial to illustrate the concepts of overfitting and random forests.

▶ Code

Let’s look at the training data.

▶ Code

```
<class 'pandas.DataFrame'>
RangeIndex: 891 entries, 1 to 891
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Name      891 non-null    str
 3   Sex       891 non-null    str
 4   Age       714 non-null    float64
 5   SibSp     891 non-null    int64
 6   Parch     891 non-null    int64
 7   Ticket    891 non-null    str
 8   Fare      891 non-null    float64
 9   Cabin     204 non-null    str
 10  Embarked  889 non-null    str
dtypes: float64(2), int64(4), str(5)
memory usage: 76.7 KB
```

We can see that there are 891 entries with 11 fields. ‘Age’, ‘Cabin’, and ‘Embarked’ have missing values.

▶ Code

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tick
PassengerId								
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 211
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 1759
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O 3101282
	1	1	Futrelle, Mrs	female	35.0	1	0	113803

Let’s look at the test data.

▶ Code

```
<class 'pandas.DataFrame'>
RangeIndex: 418 entries, 892 to 1309
Data columns (total 10 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Pclass    418 non-null    int64
 1   Name      418 non-null    str
 2   Sex       418 non-null    str
 3   Age       332 non-null    float64
 4   SibSp     418 non-null    int64
 5   Parch     418 non-null    int64
 6   Ticket    418 non-null    str
 7   Fare      417 non-null    float64
 8   Cabin     91 non-null     str
 9   Embarked  418 non-null    str
dtypes: float64(2), int64(3), str(5)
memory usage: 32.8 KB
```

	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
PassengerId								
892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292
	3	Wilkes,	female	47.0	1	0	363272	7.0000

We'll do some data cleaning and preparation.

▶ Code

Look at the dataframes again.

▶ Code

```
<class 'pandas.DataFrame'>
RangeIndex: 891 entries, 1 to 891
Data columns (total 12 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Name      891 non-null    str
 3   Sex       891 non-null    category
 4   Age       891 non-null    float64
 5   SibSp     891 non-null    int64
 6   Parch     891 non-null    int64
 7   Ticket    891 non-null    str
 8   Fare      891 non-null    float64
 9   Cabin     891 non-null    str
 10  Embarked  891 non-null    category
 11  LogFare   891 non-null    float64
dtypes: category(2), float64(3), int64(4), str(3)
memory usage: 71.5 KB
```

▶ Code

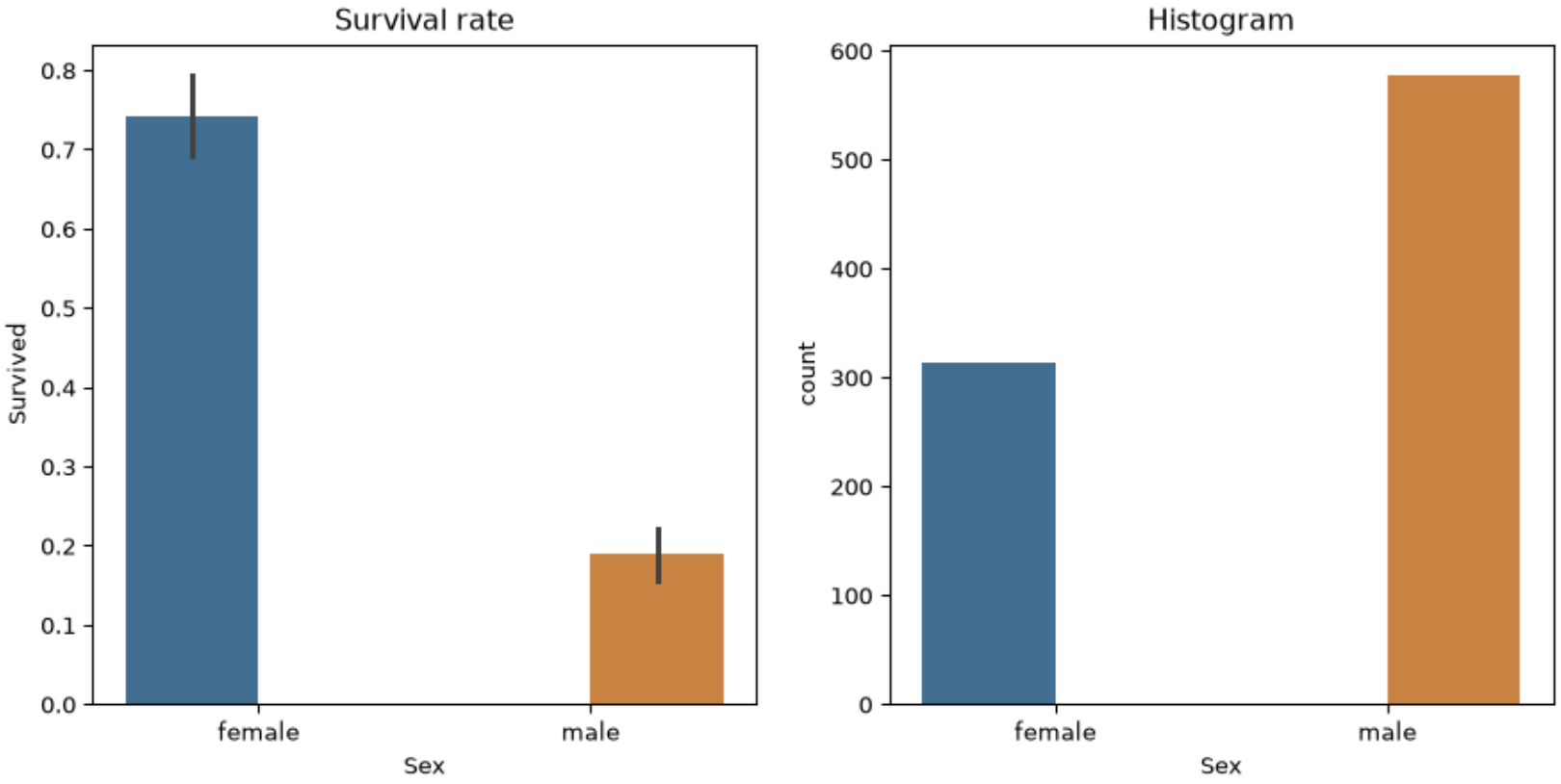
```
<class 'pandas.DataFrame'>
RangeIndex: 418 entries, 892 to 1309
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Pclass    418 non-null    int64
 1   Name      418 non-null    str
 2   Sex       418 non-null    category
 3   Age       418 non-null    float64
 4   SibSp     418 non-null    int64
 5   Parch     418 non-null    int64
 6   Ticket    418 non-null    str
 7   Fare      418 non-null    float64
 8   Cabin     418 non-null    str
 9   Embarked  418 non-null    category
 10  LogFare   418 non-null    float64
dtypes: category(2), float64(3), int64(3), str(3)
memory usage: 30.4 KB
```

We'll create lists of features by type.

```
cats=["Sex","Embarked"]  # Categorical
conts=['Age', 'SibSp', 'Parch', 'LogFare',"Pclass"]  # Continuous
dep="Survived"  # Dependent variable
```

Let’s explore some fields starting with survival rate by gender.

▶ Code



Indeed, “women and children first” was enforced on the Titanic.

Since we don’t have labels for the test data, we’ll split the training data into training and validation.

```
from numpy import random
from sklearn.model_selection import train_test_split

random.seed(42)
trn_df,val_df = train_test_split(df_train, test_size=0.25)

# Replace categorical fields with numeric codes
trn_df[cats] = trn_df[cats].apply(lambda x: x.cat.codes)
val_df[cats] = val_df[cats].apply(lambda x: x.cat.codes)
```

Let’s split the independent (input) variables from the dependent (output) variable.

```
def xs_y(df):
    xs = df[cats+conts].copy()
    return xs,df[dep] if dep in df else None

trn_xs,trn_y = xs_y(trn_df)
val_xs,val_y = xs_y(val_df)
```

Here's the predictions for our extremely simple model, where `female` is coded as `0`:

```
1 preds = val_xs.Sex==0
```

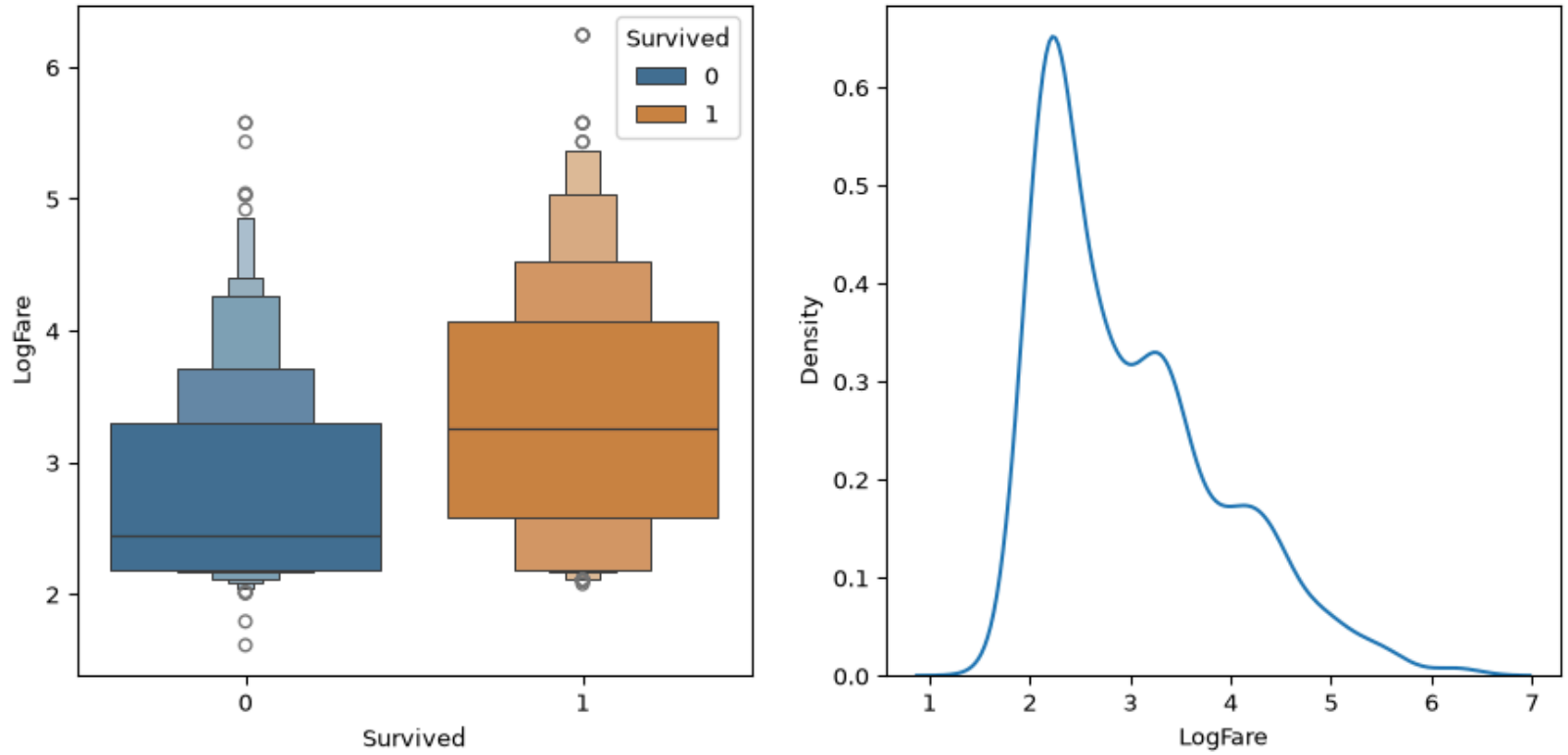
We'll use mean absolute error to measure how good this model is:

```
1 from sklearn.metrics import mean_absolute_error
2 mean_absolute_error(val_y, preds)
```

```
0.21524663677130046
```

Alternatively, we could try splitting on a continuous column. We have to use a somewhat different chart to see how this might work – here's an example of how we could look at `LogFare`:

▶ Code



The boxenplot above shows quantiles of `LogFare` for each group of `Survived==0` and

Let’s create a simple model based on this observation:

```
preds = val_xs.LogFare>2.7
```

…and test it out:

```
mean_absolute_error(val_y, preds)
```

```
0.336322869955157
```

This is quite a bit less accurate than our model that used `Sex` as the single binary split.

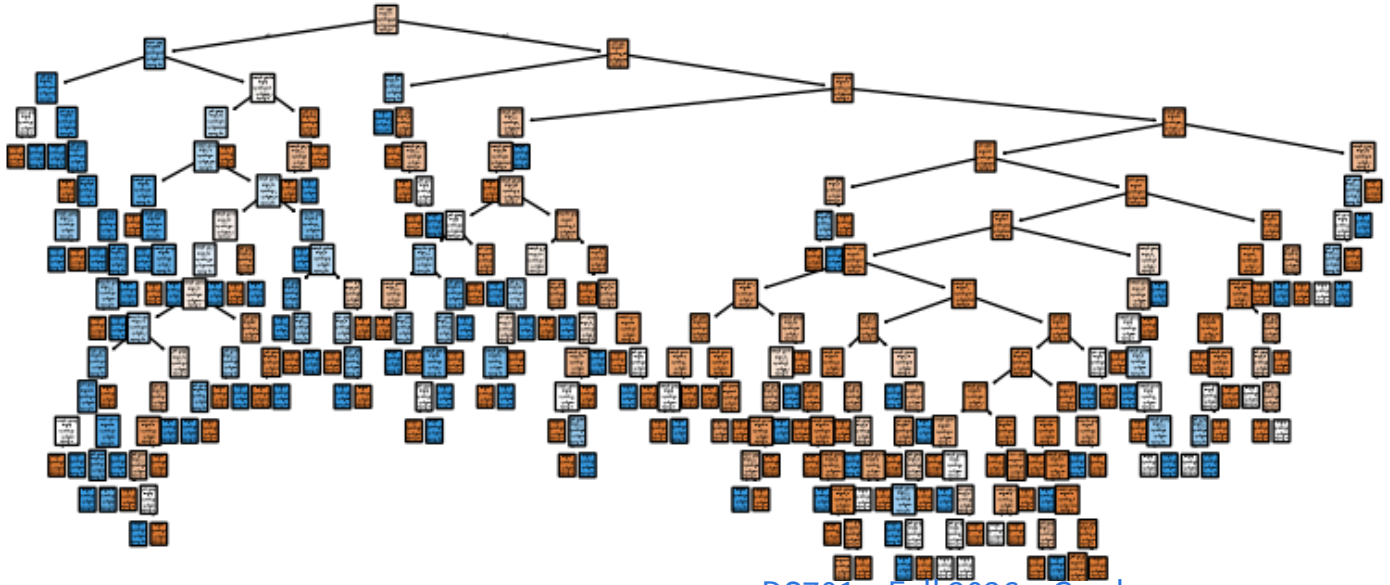
Full Decision Tree

Ok. Let's build a decision tree model using all the features.

```
clf = tree.DecisionTreeClassifier(criterion='gini', random_state=42)
clf = clf.fit(trn_xs, trn_y)
```

Let's draw the tree.

```
annotations = tree.plot_tree(clf,
                filled=True,
                rounded=True,
                feature_names=trn_xs.columns,
                class_names=['No', 'Yes'])
```



Full Tree – Evaluation Error

Let's see how it does on the validation set.

```
preds = clf.predict(val_xs)
mean_absolute_error(val_y, preds)
```

```
0.2645739910313901
```

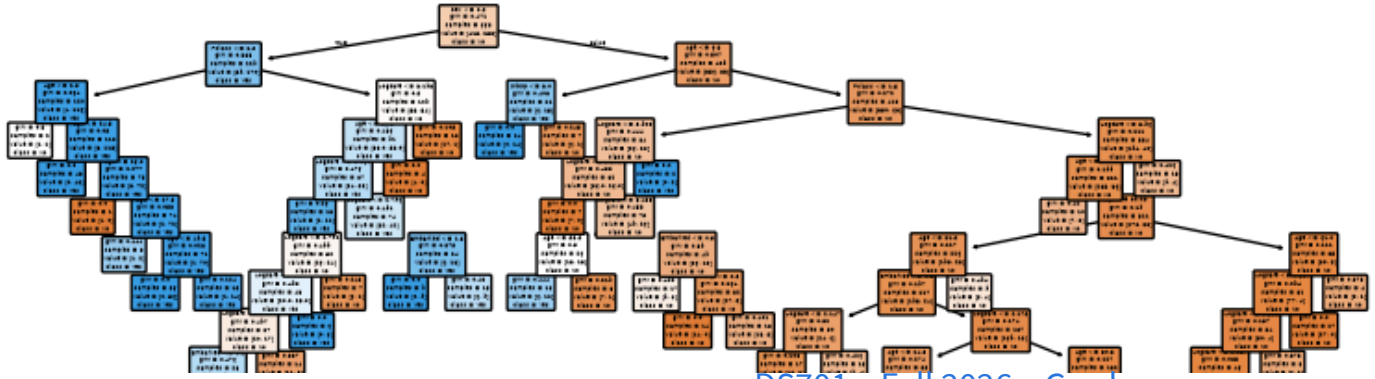
Stopping Criteria – Minimum Samples Split

Let's train the decision tree again but with stopping criteria based on the number of samples in a node.

```
clf = tree.DecisionTreeClassifier(criterion='gini', random_state=42, min_samples_split=20)
clf = clf.fit(trn_xs, trn_y)
```

Let's draw the tree.

```
annotations = tree.plot_tree(clf,
                filled=True,
                rounded=True,
                feature_names=trn_xs.columns,
                class_names=['No', 'Yes'])
```



Min Samples Split – Evaluation Error

Let’s see how it does on the validation set.

```
1 preds = clf.predict(val_xs)
2 mean_absolute_error(val_y, preds)
```

```
0.18834080717488788
```

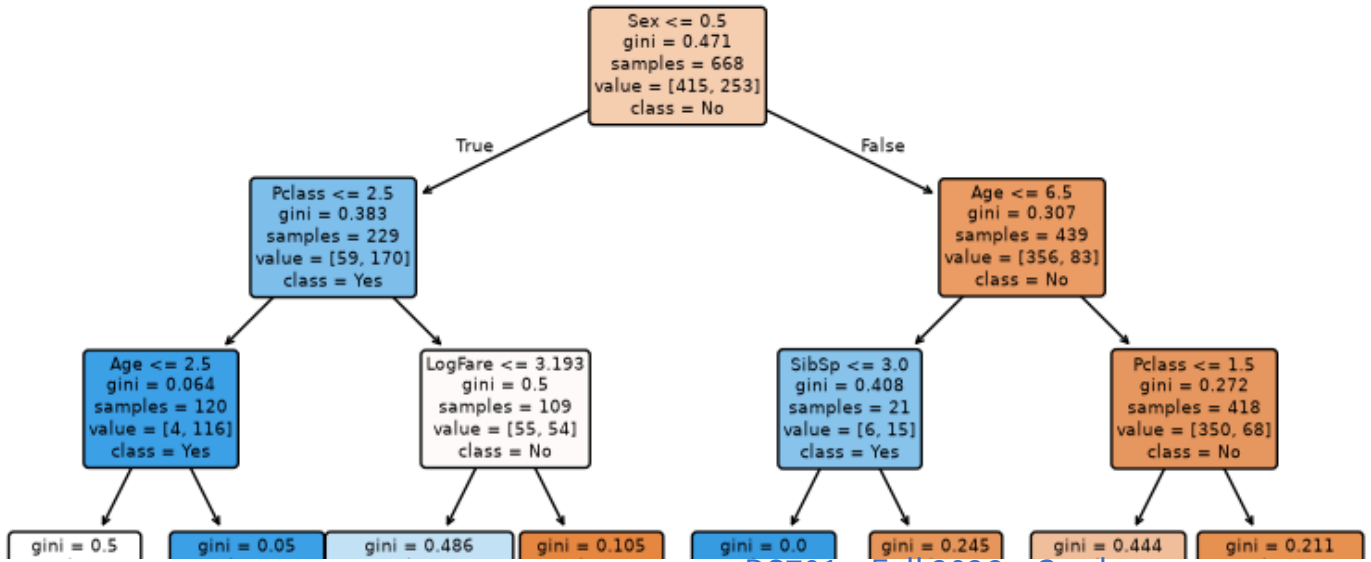
Decision Tree – Maximum Depth

Let's train the decision tree again but with a maximum depth of 3.

```
clf = tree.DecisionTreeClassifier(criterion='gini', random_state=42, max_depth=3)
clf = clf.fit(trn_xs, trn_y)
```

Let's draw the tree.

```
annotations = tree.plot_tree(clf,
                             filled=True,
                             rounded=True,
                             feature_names=trn_xs.columns,
                             class_names=['No', 'Yes'])
```



Maximum Depth – Evaluation Error

Let’s see how it does on the validation set.

```
1 preds = clf.predict(val_xs)
2 mean_absolute_error(val_y, preds)
```

```
0.19730941704035873
```

Random Forest

Let's try a random forest.

```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf = clf.fit(trn_xs, trn_y)
```

Let's see how it does on the validation set.

```
preds = clf.predict(val_xs)
mean_absolute_error(val_y, preds)
```

```
0.21076233183856502
```

Recap

- Decision Trees
- Impurity Measures
- Avoiding Overfitting
- Random Forests

References

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning*. 2nd ed. Springer. https://hastie.su.domains/ElemStatLearn/.

Tan, Pang-Ning, Michael Steinbach, and Vipin Kumar. 2018. *Introduction to Data Mining*. 2nd ed. Pearson. https://www-users.cse.umn.edu/~kumar001/dmbook/ch3_classification.pdf.