Decision Trees and Random Forest

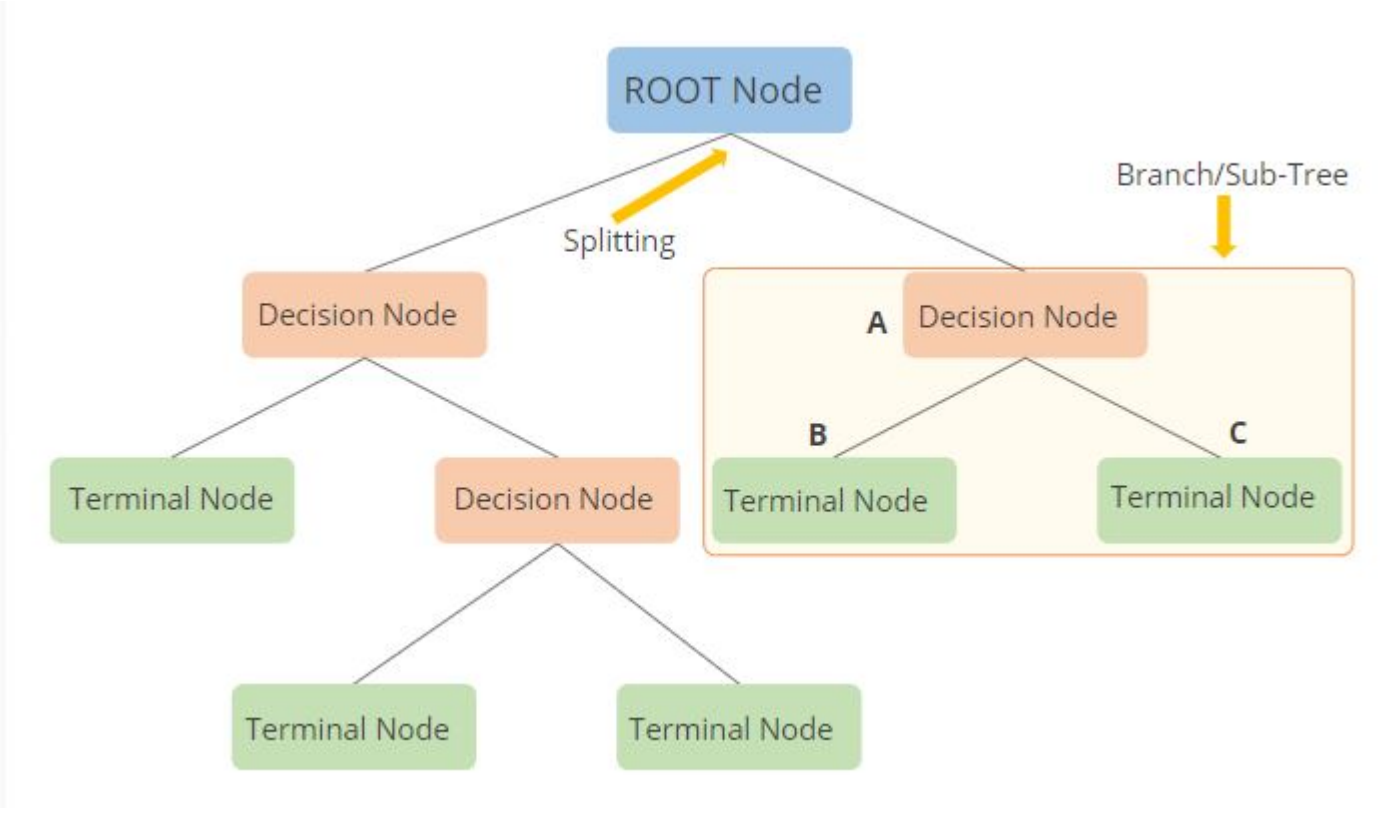
Agenda

In this lesson, we will cover the following concepts of classification with the help of a business use case:

- Decision Tree
- Random Forest
- Bagging and Bootstrapping

Decision Tree

Let us understand the basics of Decision Tree and its components with the help of a diagram.



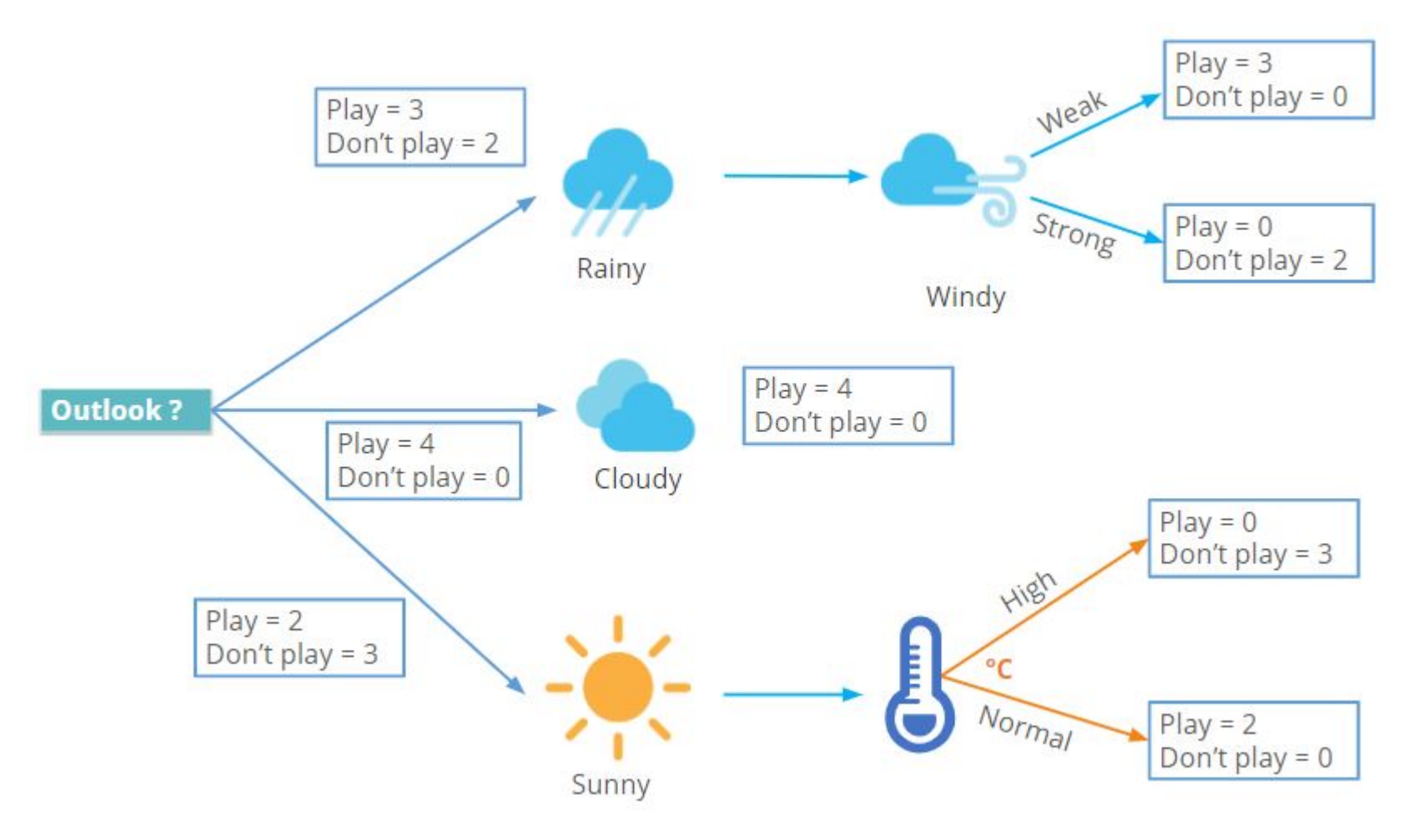
- Decision tree is a tree-like structure in which the internal node represents the test on an attribute.
- The topmost node in a decision tree is known as the root node. It learns to partition based on the attribute value.
- Each branch represents the outcome of the test and each leaf node represents the class label.
- A path from root to leaf represents the classification rules.

Splitting:

- If pure node is detected, it can be classified as leaf node.
- If impure node is detected, the best attribute is selected and splitting is continued until a pure node is detected.

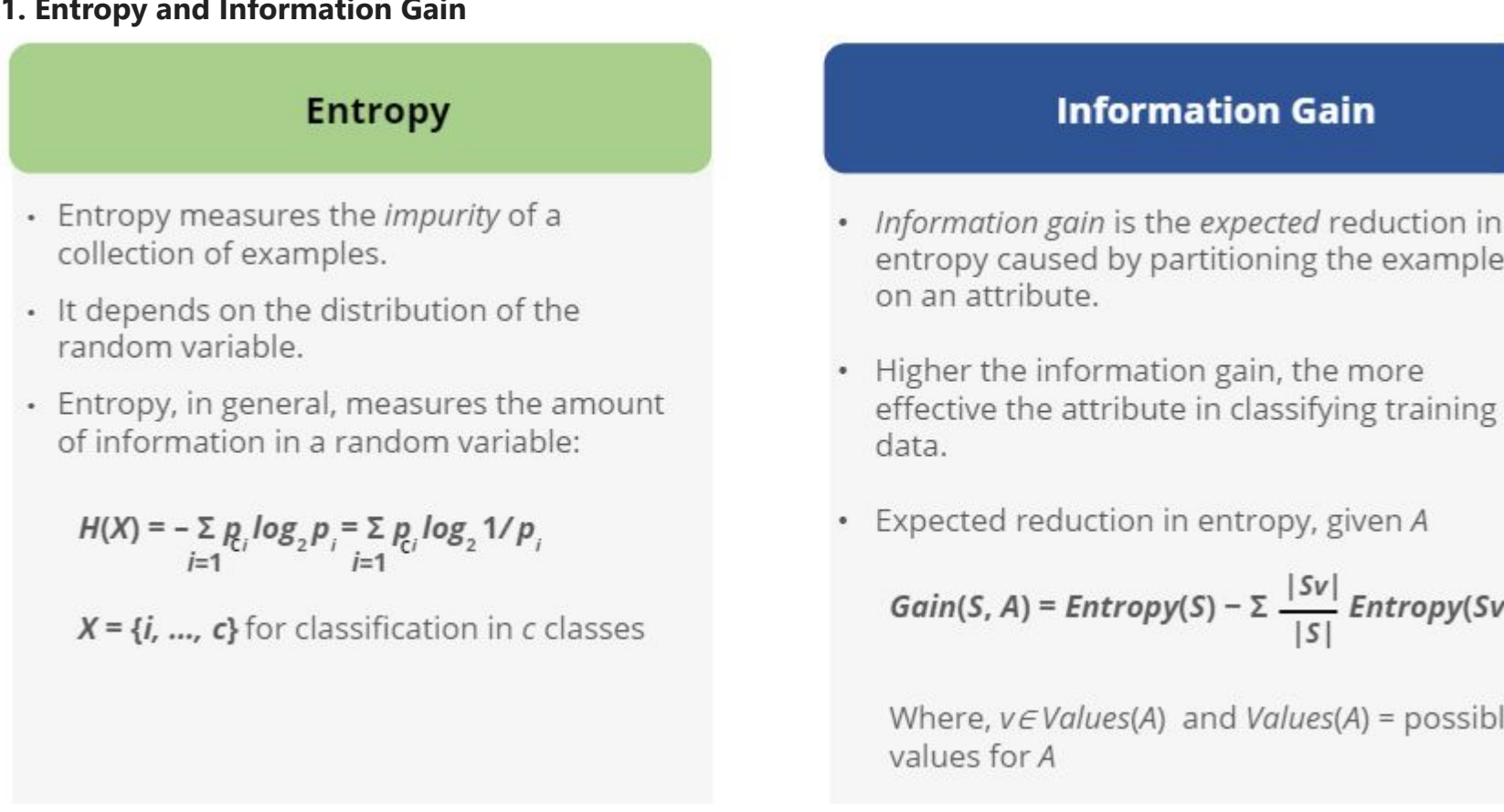
Example:

Form a decision tree to check if the match will be played or not based on climatic conditions




```
# This is formatted as code
```

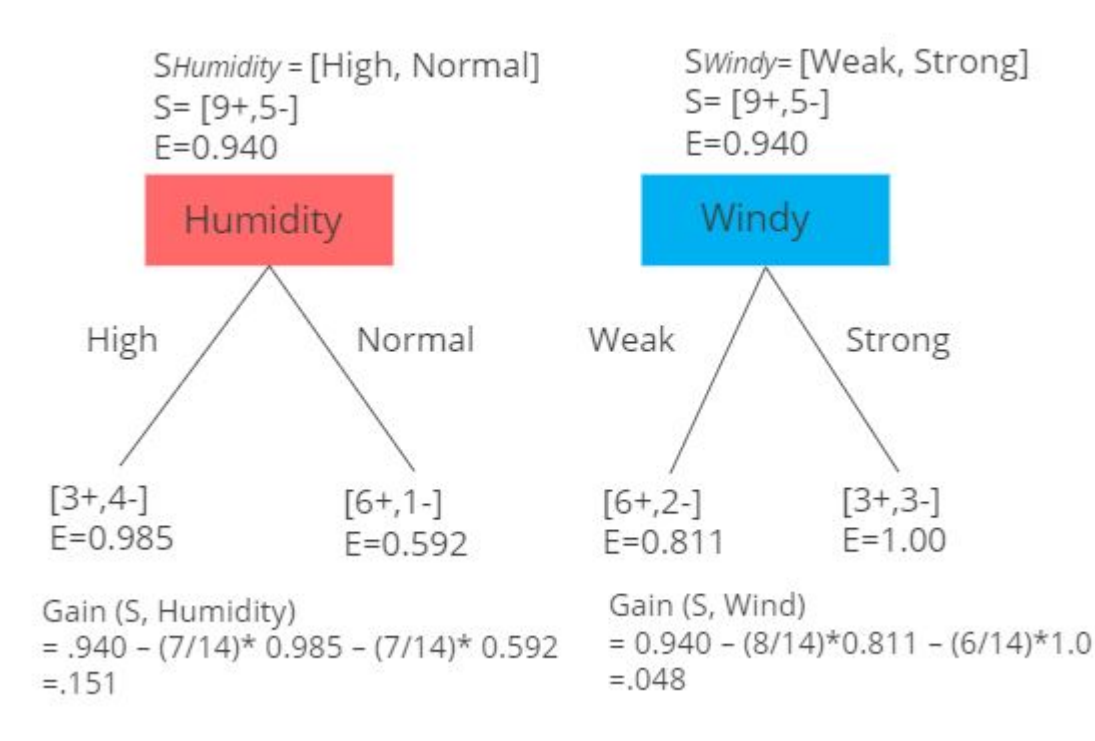
1. Entropy and Information Gain



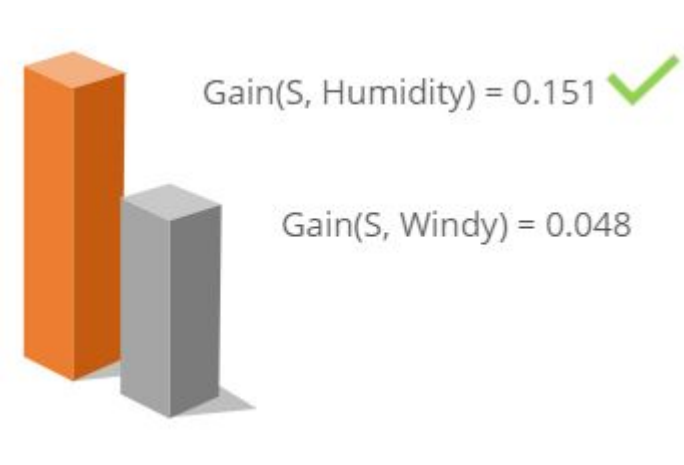
Let us consider an example where:

- Sample for humidity = [High, Normal] = [9+, 5-]
- Sample for wind = [Weak, Strong] = [9+,5-]

a. Calculate gain:



b. Select highest gain:



c. Inference:

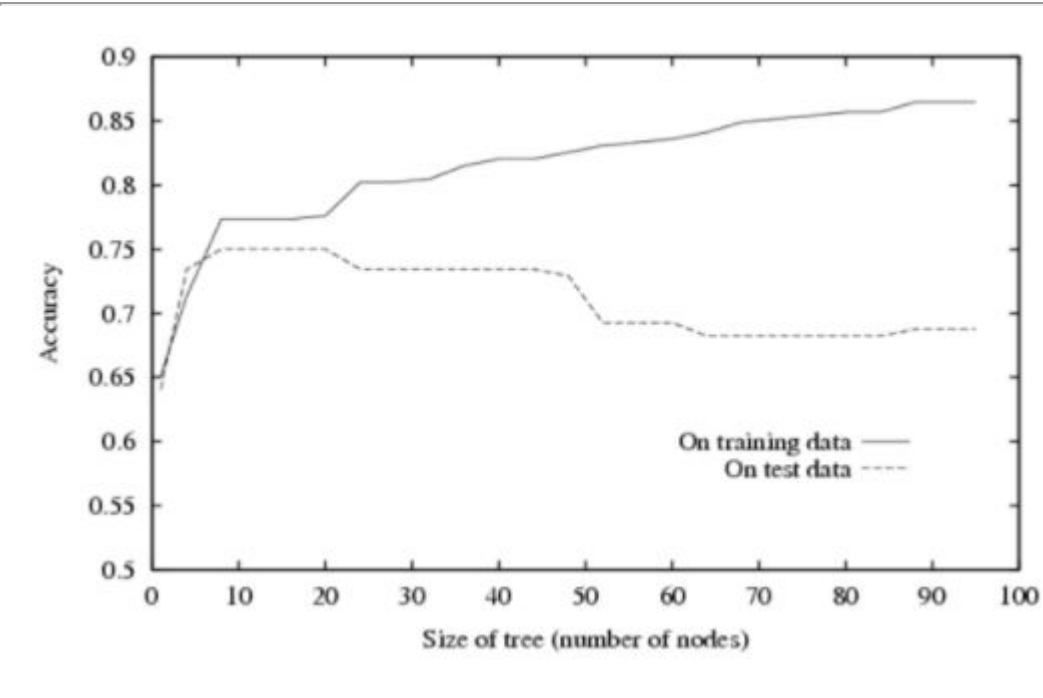
- Humidity provides the best prediction for the target
- For each possible value of Humidity, you can add a successor to the tree

Overfitting and Pruning

Overfitting:

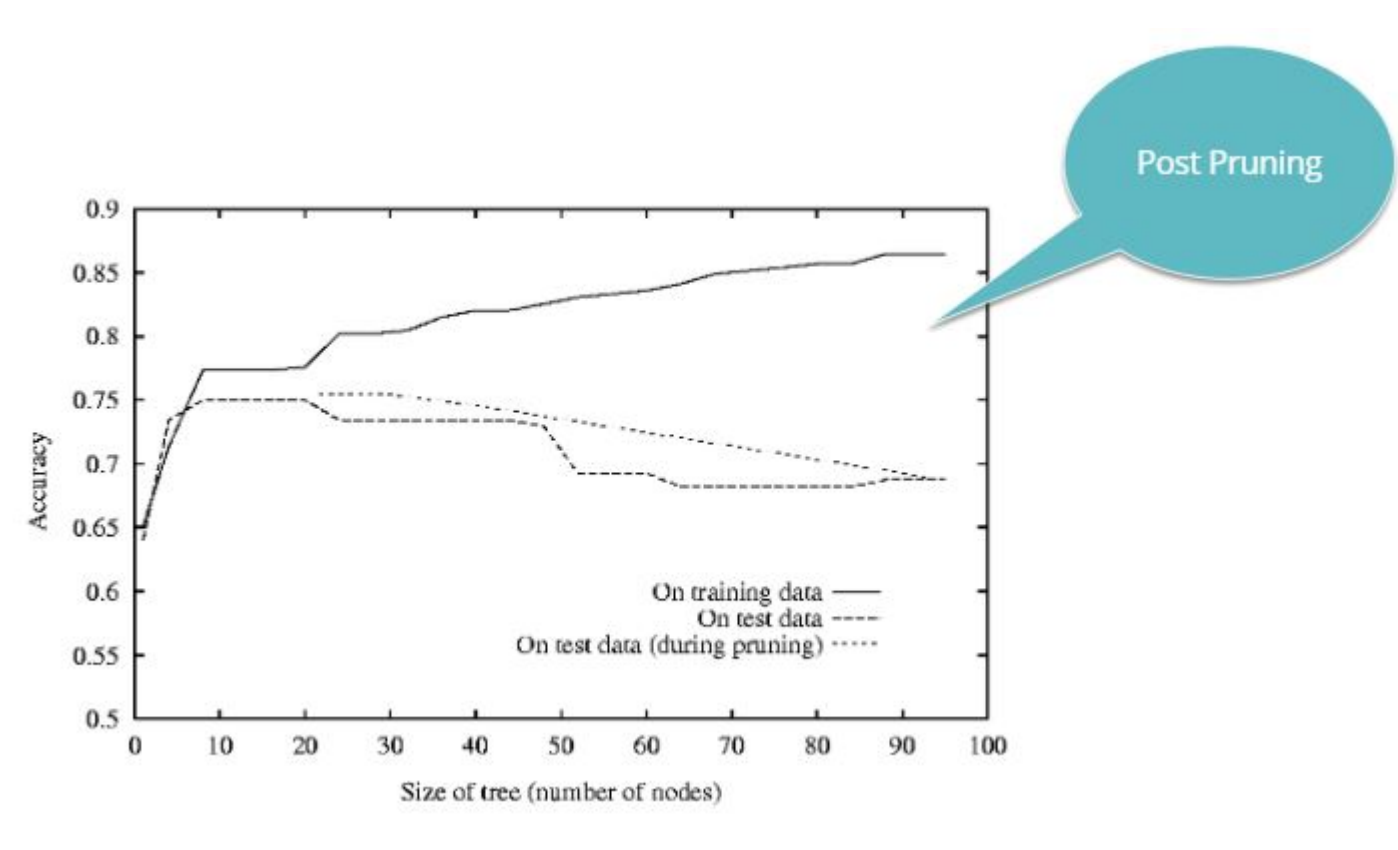
Overfitting refers to the unwanted behavior of a machine learning algorithm. It occurs when a model's performance

on the training dataset improves at the expense of poor performance on non-training data, such as a holdout test dataset or new data. By testing a machine learning model on both the training dataset and a hold-out dataset, we can determine whether a machine learning model is overfitted. If the performance of the model on the training dataset is significantly better than the performance on the test dataset, then the model may have overfitted the training dataset.



Pruning:

- The removal of the sub-tree at a node is called pruning.
- It is performed iteratively until the tree shows maximum accuracy.
- There are two ways to perform pruning:
    - Stop the development of the tree at an early stage, i.e., before perfect classification
    - Allow overfitting and later prune the tree



2. Gini Index

The Gini Index is a metric that measures the likelihood that a randomly chosen element will be incorrectly identified. It means an attribute with a low Gini index should be preferred. The decision tree algorithm aims to achieve partitions in the terminal nodes as pure as possible. The Gini index is one method used to achieve this.

- It is calculated by subtracting the sum of the squared probabilities of each class from one.

Formula:

$$Gini(D) = 1 - \sum_{i=1}^{C} (p_i)^2$$

Where, p_i is the probability that a tuple in D belongs to class C_i.

- It uses a binary split for every variable where, weighted sum of the impurity of each partition is computed in splitting.

  ##### **Example:**

  Binary split on variable A partitions data D into D1 and D2.

  What is the Gini index of D?

$$Gini_A(D) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2)$$

- In case of a discrete-valued attribute, the subset that gives the minimum gini index is selected as a splitting attribute. In the case of continuous-valued attributes, the strategy is to select each pair of adjacent values as a possible split-point and point with smaller gini index chosen as the splitting point.

Features:

- Gini index favors larger partitions.
- It uses the squared proportion of the classes.
- Gini index is perfectly classified when it is zero.

Random Forest

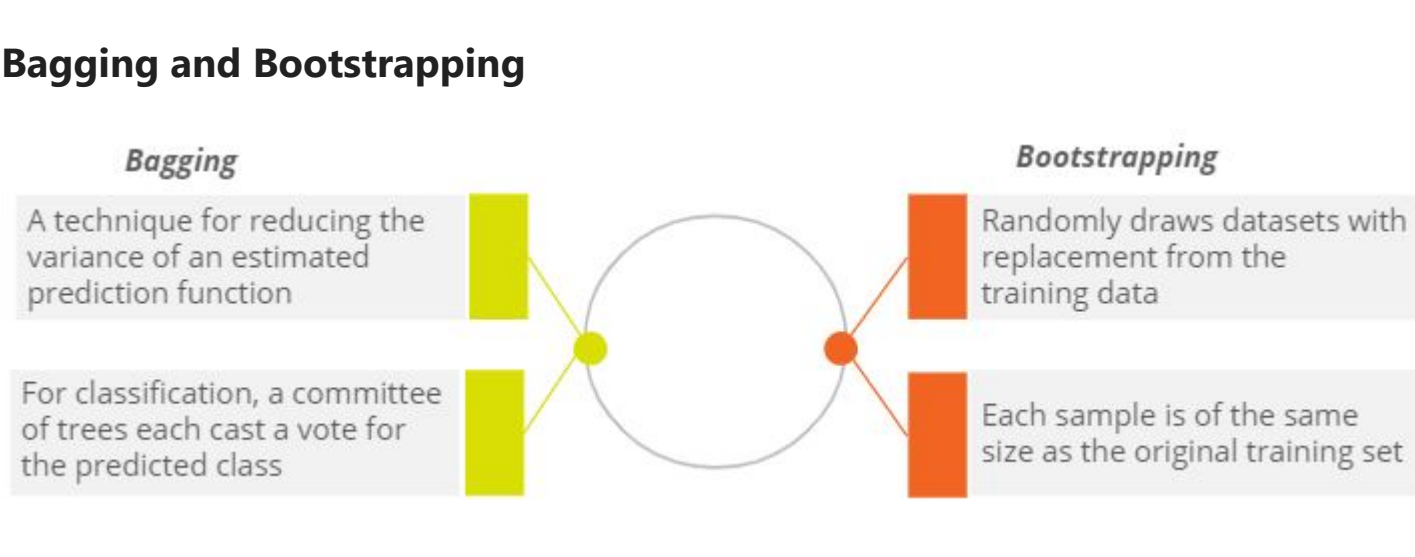
Random Forests is a supervised machine learning algorithm used in classification and regression problems. It builds decision trees for different samples and takes the majority for classification and the average for regression.

Features:

- Runs efficiently on larger databases
- Handles thousands of input variables without variable deletion
- Provides an estimate of what variables are important in the classification
- Provides a method to detect variable interactions experimentally

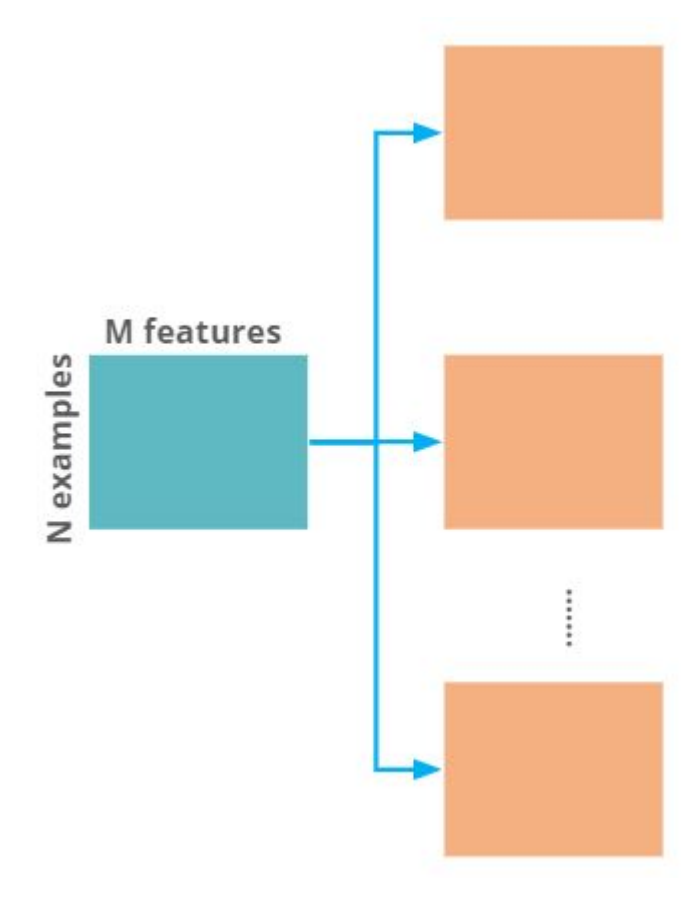
Let us extrapolate from the previous topic and discuss the concepts of random forest.

Bagging and Bootstrapping

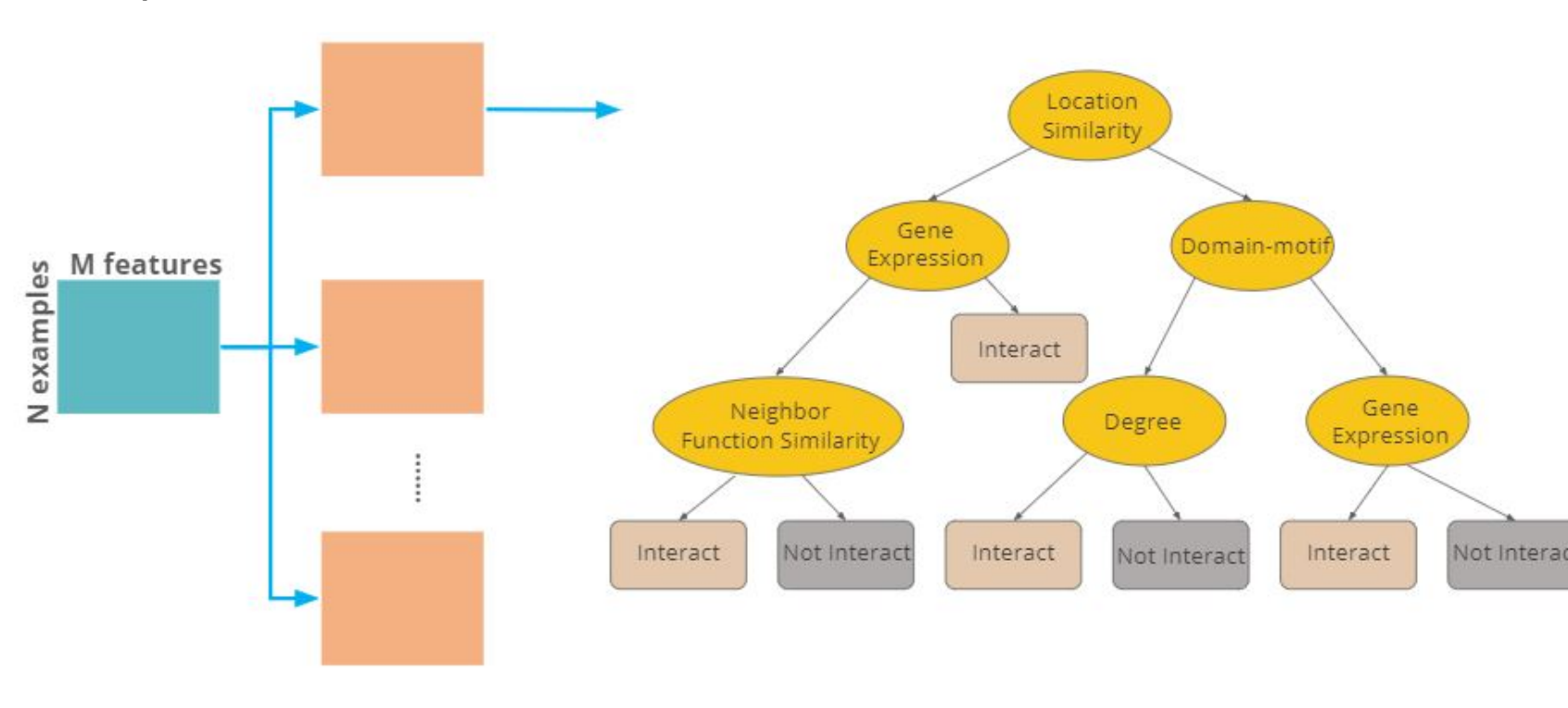


Road to random forest classifier:

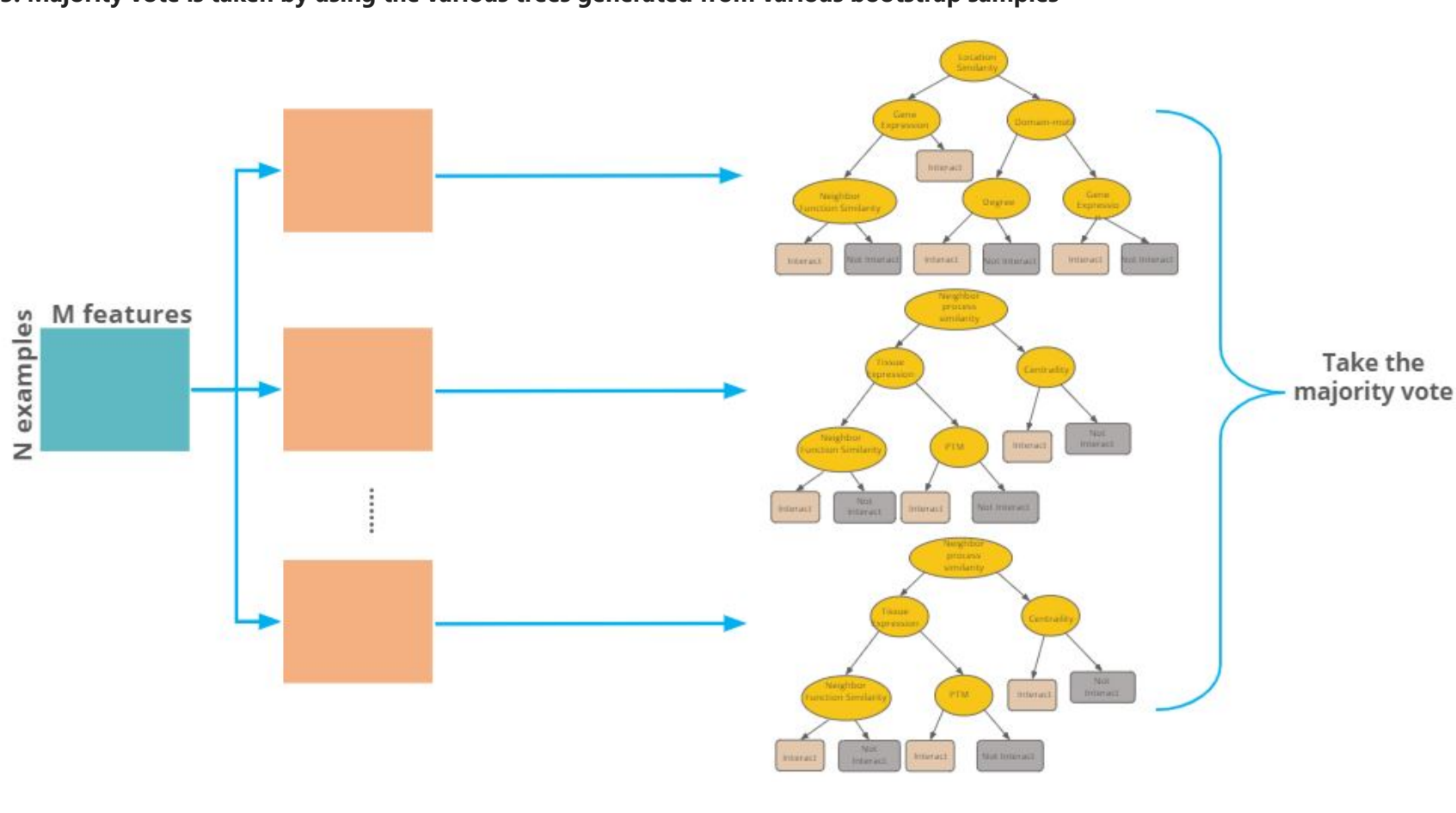
1. Create bootstrap samples from the training data



2. Each sample contributes to a decision tree classifier



3. Majority vote is taken by using the various trees generated from various bootstrap samples



Use Case: Decision Tree vs. Random Forest

Problem statement: PeerLoanKart is an NBFC (non-banking financial company) that facilitates peer-to-peer loans. It connects people who need money (borrowers) with people who have money (investors). As an investor, you would want to invest in people who showed a profile of having a high probability of paying you back. Create a model that will help predict whether a borrower will repay the loan.

Analysis to be done: Increase profits by up to 20% as NPAs will be reduced due to loan disbursal to creditworthy borrowers only

Import Libraries

In [1]:

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.model_selection import train_test_split
```

Import Data

In [2]:

```
# Importing the dataset
loans = pd.read_csv('./loan_borowwer_data.csv')
loans.describe()
```

Out[2]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6n
count	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9.578000e+03	9578.000000	9578.000
mean	0.804970	0.122640	319.089413	10.932117	12.606679	710.846314	4560.767197	1.691396e+04	46.799236	1.577
std	0.396245	0.026847	207.071301	0.614813	6.883970	37.970537	2496.930377	3.375619e+04	29.014417	2.200
min	0.000000	0.060000	15.670000	7.547502	0.000000	612.000000	178.958333	0.000000e+00	0.000000	0.000
25%	1.000000	0.103900	163.770000	10.558414	7.212500	682.000000	2820.000000	3.187000e+03	22.600000	0.000
50%	1.000000	0.122100	268.950000	10.928884	12.665000	707.000000	4139.958333	8.596000e+03	46.300000	1.000
75%	1.000000	0.140700	432.762500	11.291293	17.950000	737.000000	5730.000000	1.824950e+04	70.900000	2.000
max	1.000000	0.216400	940.140000	14.528354	29.960000	827.000000	17639.958330	1.207359e+06	119.000000	33.000

Exploratory Data Analysis

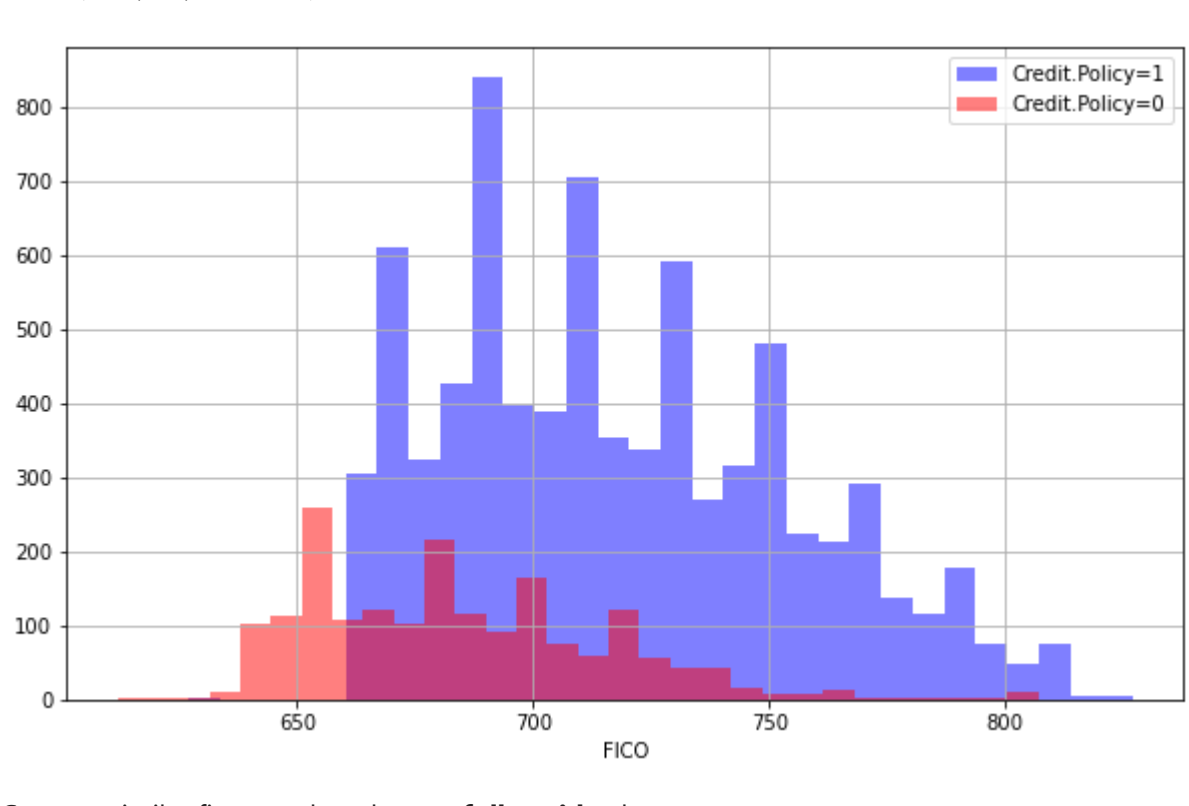
Create a histogram of two FICO distributions on top of each other, one for each credit.policy outcome.

In [4]:

```
# Creating a histogram
plt.figure(figsize=(10,6))
loans[loans['credit.policy']==1]['fico'].hist(alpha=0.5,color='blue',  bins=30,
                                              label='Credit.Policy=1')
loans[loans['credit.policy']==0]['fico'].hist(alpha=0.5,color='red', bins=30,
                                              label='Credit.Policy=0')
plt.legend()
plt.xlabel('FICO')
```

Out[4]:

```
Text(0.5, 0, 'FICO')
```



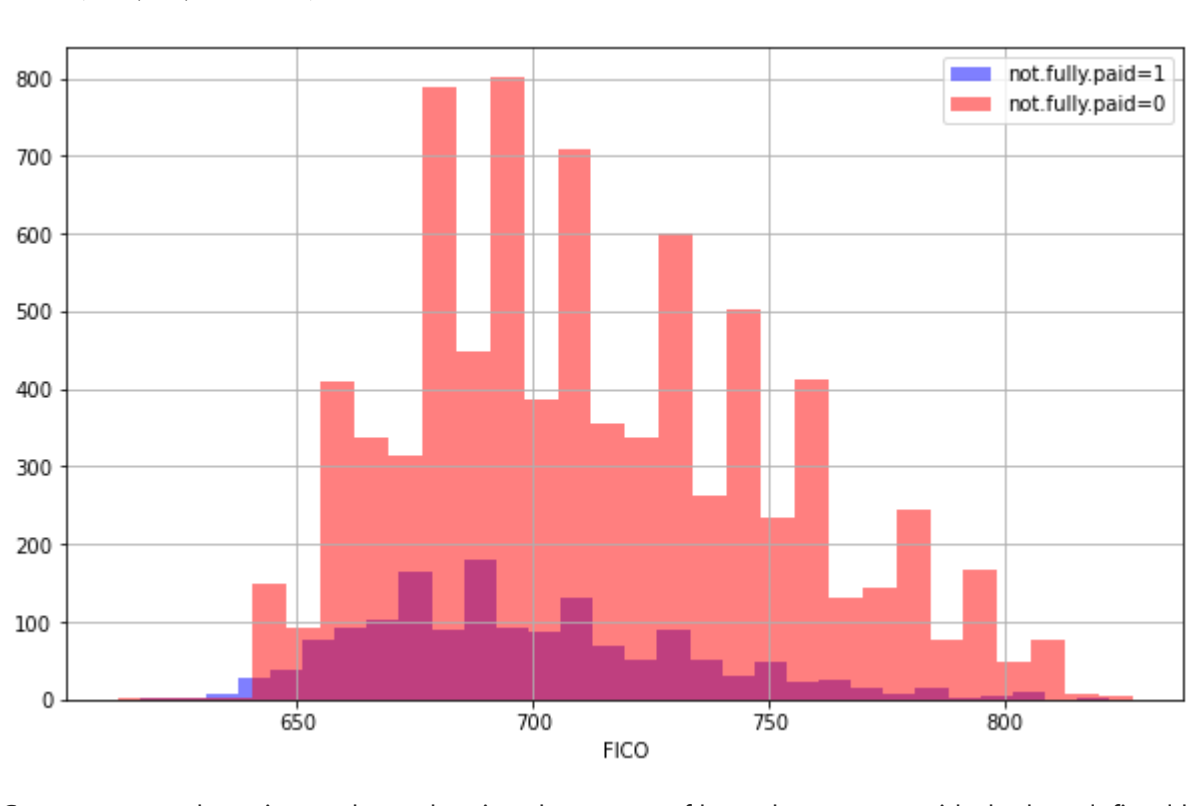
Create a similar figure; select the **not.fully.paid** column.

In [5]:

```
plt.figure(figsize=(10,6))
loans[loans['not.fully.paid']==1]['fico'].hist(alpha=0.5,color='blue',
                                              bins=30,label='not.fully.paid=1')
loans[loans['not.fully.paid']==0]['fico'].hist(alpha=0.5,color='red',
                                              bins=30,label='not.fully.paid=0')
plt.legend()
plt.xlabel('FICO')
```

Out[5]:

```
Text(0.5, 0, 'FICO')
```



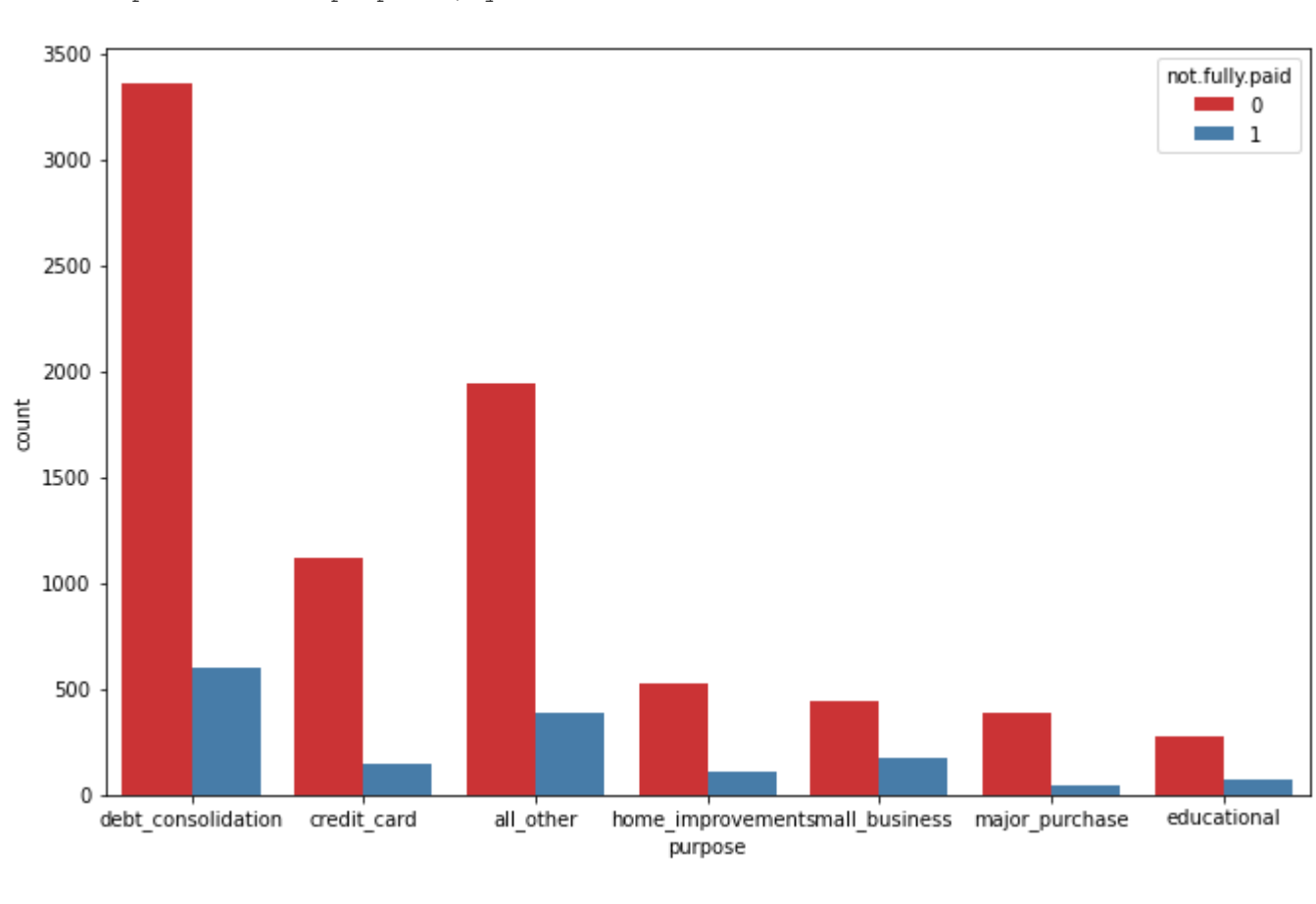
Create a countplot using seaborn showing the counts of loans by purpose with the hue defined by **not.fully.paid**.

In [6]:

```
# Creating countplot
plt.figure(figsize=(11,7))
sns.countplot(x='purpose',hue='not.fully.paid',data=loans,palette='Set1')
```

Out[6]:

```
<AxesSubplot:xlabel='purpose', ylabel='count'>
```



Setting Up the Data

Create a list of elements containing the string **purpose**. Name this list **cat_feats**.

In [7]:

```
cat_feats = ['purpose']
```

Now use **pd.get_dummies (loans,columns=cat_feats,drop_first=True)** to create a fixed and larger data frame that has new feature columns with dummy variables. Name this data frame **final_data**.

In [8]:

```
final_data = pd.get_dummies(loans,columns=cat_feats,drop_first=True)
final_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 19 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   credit.policy               9578 non-null   int64
 1   int.rate                    9578 non-null   float64
 2   installment                 9578 non-null   float64
 3   log.annual.inc              9578 non-null   float64
 4   dti                         9578 non-null   float64
 5   fico                        9578 non-null   int64
 6   days.with.cr.line           9578 non-null   float64
 7   revol.bal                   9578 non-null   int64
 8   revol.util                  9578 non-null   float64
 9   inq.last.6mths              9578 non-null   int64
 10  delinq.2yrs                 9578 non-null   int64
 11  pub.rec                     9578 non-null   int64
 12  not.fully.paid              9578 non-null   int64
 13  purpose_credit_card         9578 non-null   uint8
 14  purpose_debt_consolidation  9578 non-null   uint8
 15  purpose_educational         9578 non-null   uint8
 16  purpose_home_improvement    9578 non-null   uint8
 17  purpose_major_purchase      9578 non-null   uint8
 18  purpose_small_business      9578 non-null   uint8
dtypes: float64(6), int64(7), uint8(6)
memory usage: 1.0 MB
```

Train-Test Split

Let us distribute the data into **training** and **test** datasets using the **train_test_split()** function.

In [9]:

```
X = final_data.drop('not.fully.paid',axis=1)
y = final_data['not.fully.paid']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=101)
```

Training Decision Tree Model

Let us create a **DecisionTreeClassifier** from **sklearn.tree**.

In [10]:

```
# Decision tree model
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
dtree.fit(X_train,y_train)
```

Out[10]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

Evaluating Decision Tree

Let us create a classification report for the **DecisionTreeClassifier** we created earlier.

In [11]:

```
# Classification report
predictions = dtree.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.85      0.82      0.84      2431
           1       0.19      0.22      0.20       443

    accuracy                           0.73      2874
   macro avg       0.52      0.52      0.52      2874
weighted avg       0.75      0.73      0.74      2874
```

Confusion Matrix

In [12]:

```
# Creating a confusion matrix
print(confusion_matrix(y_test,predictions))
```

```
[[1997  434]
 [ 344   99]]
```

Training Random Forest Model

Let us create a **RandomForestClassifier** for **sklearn.ensemble** library.

In [13]:

```
# Random forest model
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=600)
rfc.fit(X_train,y_train)
```

Out[13]:

```
▾ RandomForestClassifier
RandomForestClassifier(n_estimators=600)
```

Evaluating Random Forest Model

Let us create a classification report for the **RandomForestClassifier** we created earlier.

In [14]:

```
predictions = rfc.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.85      1.00      0.92      2431
           1       0.50      0.02      0.04       443

    accuracy                           0.85      2874
   macro avg       0.67      0.51      0.48      2874
weighted avg       0.79      0.85      0.78      2874
```

Printing the Confusion Matrix

In [15]:

```
print(confusion_matrix(y_test,predictions))
```

```
[[2421   10]
 [ 433   10]]
```

Conclusion

From the above use case, we can see that the accuracy of random forest is better than decision tree.

Note: In this lesson, we saw the use of the decision trees and random forest methods, but in the next lesson we will be working on "Unsupervised Learning".

Powered by simplilearn