17.7.2

## Predict Loan Application Approval

**Now** that you have learned how decision trees work, it's time to look at using a decision tree model in practice. You will first perform the data preprocessing steps.

Let's try to predict loan application approvals using a decision tree on the `loans_data_encoded.csv` data we previously used to encode the dataset. Feel free to begin a new notebook, or to follow along the provided notebook.

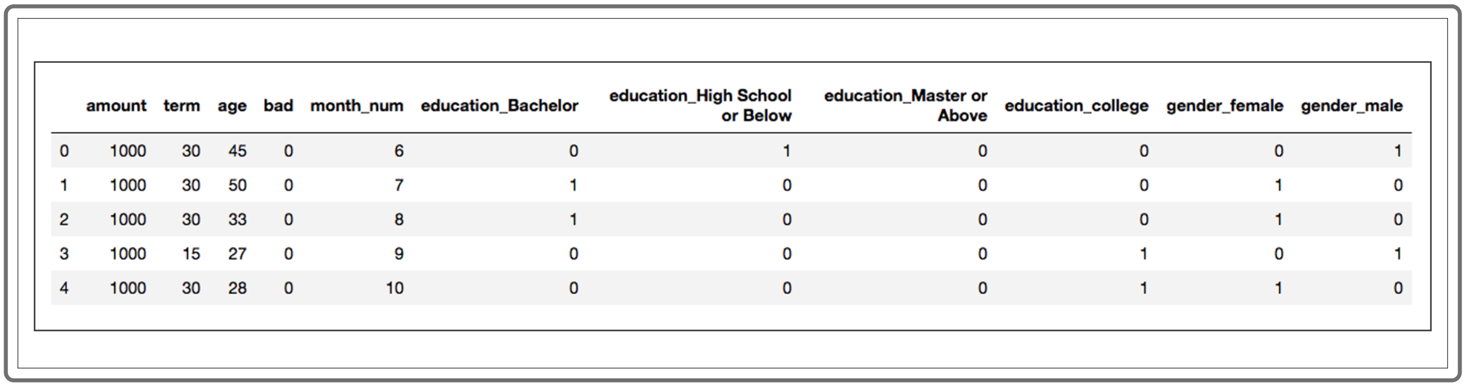
**Download 17-7-2-decision_tree.zip** (https://2u-data-curriculum-team.s3.amazonaws.com/dataviz-online/module_17/17-7-2-decision_tree.zip)

In Jupyter Notebook, import the following dependencies:

```
# Initial imports
import pandas as pd
from path import Path
from sklearn import tree
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
```

Next, read in your saved `loans_data_encoded.csv` file.

```
# Loading data
file_path = Path("../Resources/loans_data_encoded.csv")
df_loans = pd.read_csv(file_path)
df_loans.head()
```



| | amount | term | age | bad | month_num | education_Bachelor | education_High School or Below | education_Master or Above | education_college | gender_female | gender_male |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1000 | 30 | 45 | 0 | 6 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1000 | 30 | 50 | 0 | 7 | 1 | 0 | 0 | 0 | 1 | 0 |
| 2 | 1000 | 30 | 33 | 0 | 8 | 1 | 0 | 0 | 0 | 1 | 0 |
| 3 | 1000 | 15 | 27 | 0 | 9 | 0 | 0 | 0 | 1 | 0 | 1 |
| 4 | 1000 | 30 | 28 | 0 | 10 | 0 | 0 | 0 | 1 | 1 | 0 |

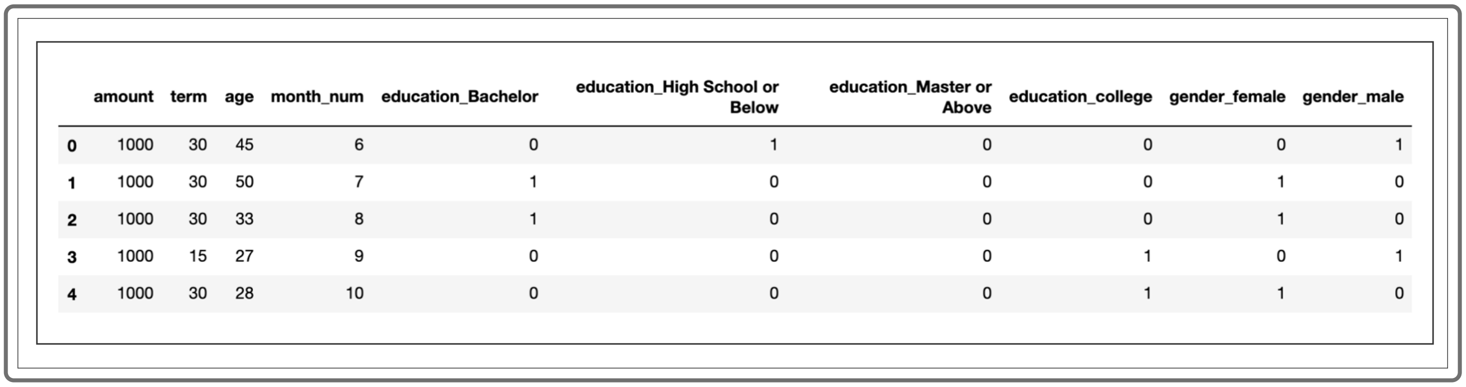
Our goal is to predict if a loan application is worthy of approval based on information we have in our `df_loans` DataFrame. To do this, we'll have to split our dataset into features (or inputs) and target (or outputs). The features set, `X`, will be a copy of the `df_loans` DataFrame without the `bad` column. These features are all the variables that help determine whether a loan application should be denied.

**REWIND**

Recall that X is the input data and y is the output data.

```
# Define the features set.
X = df_loans.copy()
X = X.drop("bad", axis=1)
X.head()
```

The output from the following code block will give us the following features set.



| | amount | term | age | month_num | education_Bachelor | education_High School or Below | education_Master or Above | education_college | gender_female | gender_male |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1000 | 30 | 45 | 6 | 0 | 1 | 0 | 0 | 0 | 1 |
| **1** | 1000 | 30 | 50 | 7 | 1 | 0 | 0 | 0 | 1 | 0 |
| **2** | 1000 | 30 | 33 | 8 | 1 | 0 | 0 | 0 | 1 | 0 |
| **3** | 1000 | 15 | 27 | 9 | 0 | 0 | 0 | 1 | 0 | 1 |
| **4** | 1000 | 30 | 28 | 10 | 0 | 0 | 0 | 1 | 1 | 0 |

The target set is the `bad` column, indicating whether or not a loan application is good (0) or bad (1). Run the following code to generate the target set data.

```
# Define the target set.
y = df_loans["bad"].values
y[:5]
```

A preview of the target set indicates five good (loan worthy) applications.

```
array([0, 0, 0, 0, 0])
```

## Split the Data into Training and Testing Sets

To train and validate our model, we'll need to split the features and target sets into training and testing sets. This will help determine the relationships between each feature in the features training set and the target training set, which we'll use to determine the validity of our model using the features and target testing sets.

In Jupyter Notebook, add the following code that will split our data into training and testing sets.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=78)
```

When the `train_test_split()` function is executed, our data is split into a specific proportion of the original data sets. By default, our training and testing data sets are 75% and 25%, respectively, of the original data. Using the following code, we can see the data's 75-25 split.

```
# Determine the shape of our training and testing sets.
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

The output from running the code above shows that the `X_train` and `y_train` is 75% of 500 and that the `X_test` and `y_test` are 25%.

```
(375, 10)
(125, 10)
(375, 1)
(125, 1)
```

We can manually specify the desired split with the `train_size` parameter.

```
# Splitting into Train and Test sets into an 80/20 split.
X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, random_state=78, train_size=0.80)
```

To see that the shape of our training and testing sets is a 80-20 split, we run the following code.

```
# Determine the shape of our training and testing sets.
print(X_train2.shape)
print(X_test2.shape)
print(y_train2.shape)
print(y_test2.shape)
```

The output from this code will give the following results.

```
(400, 10)
(100, 10)
(400, 1)
(100, 1)
```

**NOTE**

Consult the sklearn documentation for additional information about the **train_test_split()** **(https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)** function and the parameters it takes.

## Scale the Training and Testing Data

Now that we have split our data into training and testing sets, we can scale the data using Scikit-learn's `StandardScaler`.

**REWIND**

The standard scaler standardizes the data. Which means that each feature will be rescaled so that its mean is 0 and its standard deviation is 1.

**NOTE**

Typically, models that compute distances between data points, such as SVM, require scaled data. Although decision trees don't require scaling the data, it can be helpful when comparing the performances of different models.

To scale our data, we'll use the `StandardScaler` as before and fit the instance, `scaler`, with the training data and then scale the features with the `transform()` method:

```
# Creating a StandardScaler instance.
scaler = StandardScaler()
# Fitting the Standard Scaler with the training data.
X_scaler = scaler.fit(X_train)

# Scaling the data.
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

Which of the following should go in the blank spaces below to compute the mean and the standard deviation of the first column of the scaled data?

```
 Import numpy as np
np.mean(X_train_scaled___)
np.mean(X_test_scaled___)
np.std(X_train_scaled___)
np.std(X_test_scaled___)
```

- ○ [:1]
- ○ [:,0]
- ○ [1,:]
- ○ [:,1]

Check Answer

Finish