



## Decision Trees - Decision Trees - 2

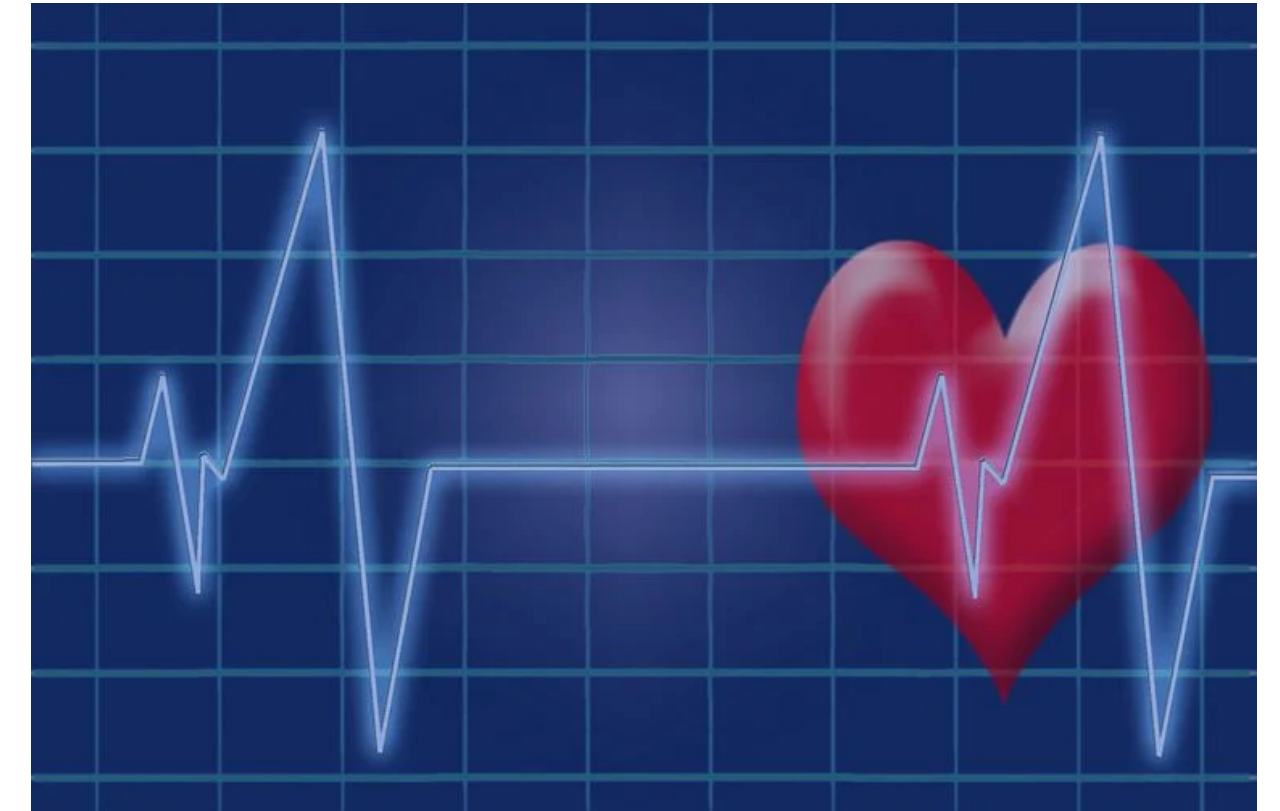
*One should look for what is and not what he thinks should be. (Albert Einstein)*

# Module completion checklist

Objectives	Complete
Transform data in order to use Decision Trees	
Implement the Decision Tree algorithm on a small subset	

# Stroke Prediction survey: case study

- According to the World Health Organization (WHO), stroke is the 2nd leading cause of death globally
- **Click here** to see a dataset showing the results of a clinical trial of a heart-disease drug survey on a sample of US adults
- Each row in the data provides relevant information about the adult, including whether they had a stroke or not
- We would like to use this data to predict whether a patient is likely to have a stroke based on their demographic information and medical history



# Dataset

- In order to implement what you learn in this course, we will be using the `healthcare-dataset-stroke-data.csv` dataset
- We will be working with columns from the dataset such as:
  - `stroke`
  - `gender`
  - `age`
  - `hypertension`
  - `heart_disease`
  - `ever_married`
- We will be using different columns of the dataset to predict `stroke` as the target variable

# Loading packages

Let's load the packages we will be using:

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pickle
from pathlib import Path
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import GridSearchCV

#import graphviz
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from matplotlib.legend_handler import HandlerLine2D
```

- To install graphviz through Anaconda, run the following code in the terminal:

```
conda install graphviz
```

- Then, install python-graphviz, which is a Python library for graphviz

```
conda install python-graphviz
```

# Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into variables
- We will use the `pathlib` library
- Let the `main_dir` be the variable corresponding to your course folder
- Let `data_dir` be the variable corresponding to your data folder

```
# Set 'main_dir' to location of the project folder
home_dir = Path(".").resolve()
main_dir = home_dir.parent.parent
print(main_dir)
```

```
data_dir = str(main_dir) + "/data"
print(data_dir)
```

```
plot_dir = str(main_dir) + "/plots"
if not os.path.exists(plot_dir):
    os.makedirs(plot_dir)
print(plot_dir)
```

# Load the dataset

- Let's load the entire dataset

```
df = pd.read_csv(str(data_dir)+"/" + 'healthcare-dataset-stroke-data.csv')
print(df.head())
```

```
   id  gender  age  ...  bmi  smoking_status  stroke
0  9046   Male  67.0  ...  36.6  formerly smoked      1
1  51676  Female  61.0  ...   NaN      never smoked      1
2  31112   Male  80.0  ...  32.5      never smoked      1
3  60182  Female  49.0  ...  34.4           smokes      1
4   1665  Female  79.0  ...  24.0      never smoked      1

[5 rows x 12 columns]
```

# Subset data

- Remove any columns from the dataframe that are not numeric or categorical as we will not be using them in our models

```
df = df[['age', 'avg_glucose_level', 'heart_disease', 'ever_married', 'hypertension',  
'Residence_type', 'gender', 'smoking_status', 'work_type', 'stroke', 'id']]  
print(df.head())
```

	age	avg_glucose_level	heart_disease	...	work_type	stroke	id
0	67.0	228.69	1	...	Private	1	9046
1	61.0	202.21	0	...	Self-employed	1	51676
2	80.0	105.92	1	...	Private	1	31112
3	49.0	171.23	0	...	Private	1	60182
4	79.0	174.12	0	...	Self-employed	1	1665

[5 rows x 11 columns]



# Data prep: check for NAs

- We now check for NAs and there are multiple methods to deal with them

```
# Check for NAs.  
print(df.isnull().sum())
```

```
age                0  
avg_glucose_level  0  
heart_disease      0  
ever_married       0  
hypertension       0  
Residence_type     0  
gender             0  
smoking_status     1544  
work_type          0  
stroke            0  
id                0  
dtype: int64
```

- If we do have NAs, we could replace them with a mean or 0

```
percent_missing = df.isnull().sum() * 100 /  
len(df)  
print(percent_missing)
```

```
age                0.000000  
avg_glucose_level  0.000000  
heart_disease      0.000000  
ever_married       0.000000  
hypertension       0.000000  
Residence_type     0.000000  
gender             0.000000  
smoking_status     30.215264  
work_type          0.000000  
stroke            0.000000  
id                0.000000  
dtype: float64
```

# Data prep: check for NAs

```
# Delete columns containing either 50% or more
than 50% NaN Values
perc = 50.0
min_count = int(((100-perc)/100)*df.shape[0] +
1)
df = df.dropna(axis=1,
               thresh=min_count)
print(df.shape)
```

```
(5110, 11)
```

```
# Function to impute NA in both numeric and
categorical columns
def fillna(df):
    # Fill numeric columns with mean value
    df = df.fillna(df.mean())
    # Fill categorical columns with mode value
    df = df.fillna(df.mode().iloc[0])
    return df

df = fillna(df)
```

# Data prep: target

- The next step of our data cleanup is to ensure the target variable is binary and has a label
- Let's look at the `dtype` of `stroke`

```
print(df['stroke'].dtypes)
```

```
int64
```

- We want to convert this to `bool` so that is a binary class

```
# Identify the the two unique classes
threshold = df['stroke'].mean()
df['stroke'] = np.where(df['stroke'] > threshold, 1, 0)
```

```
unique_values = sorted(df['stroke'].unique())
df['stroke'] = np.where(df['stroke'] == unique_values[0], False, True)
# Check class again.
print(df['stroke'].dtypes)
```

```
bool
```

# Summarize the data

- Let's use the `.describe()` function within Pandas to summarize our data

```
print(df.describe())
```

```
count    5110.000000    age    avg_glucose_level    ...    hypertension    id
mean      43.226614    106.147677    ...      0.097456    36517.829354
std       22.612647     45.283560    ...      0.296607    21161.721625
min        0.080000     55.120000    ...      0.000000     67.000000
25%       25.000000     77.245000    ...      0.000000    17741.250000
50%       45.000000     91.885000    ...      0.000000    36932.000000
75%       61.000000    114.090000    ...      0.000000    54682.000000
max       82.000000    271.740000    ...      1.000000    72940.000000

[8 rows x 5 columns]
```

# scikit-learn: tree

- We will be using the `DecisionTreeClassifier` library from `scikit-learn`

**sklearn.tree.DecisionTreeClassifier**

```
class sklearn.tree. DecisionTreeClassifier (criterion='gini', splitter='best', max_depth=None,
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None,
max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False)
```

[\[source\]](#)

A decision tree classifier.

Read more in the [User Guide](#).

**Parameters:**

**criterion : string, optional (default="gini")**  
The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

**splitter : string, optional (default="best")**  
The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

**max\_depth : int or None, optional (default=None)**  
The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples.

**min\_samples\_split : int, float, optional (default=2)**  
The minimum number of samples required to split an internal node:

- If int, then consider min\_samples\_split as the minimum number.
- If float, then min\_samples\_split is a fraction and  $\text{ceil}(\text{min\_samples\_split} * \text{n\_samples})$  are the minimum number of samples for each split.

Changed in version 0.18: Added float values for fractions.

- For all the parameters of the `tree` package, visit [scikit-learn's documentation](#)

# Trees: pre-processing steps

- Now that we know the problem and have our `df` dataset loaded, it's time to get the data ready to model
- Let's start with data pre-processing.
  - We don't need to scale data since trees are not sensitive to unscaled data
  - We don't need to look at the number of NAs because we already did this and have loaded the cleaned dataset
- **However, on your own data, these are steps you must walk through before modeling**

# Module completion checklist

Objectives	Complete
Transform data in order to use decision trees	✓
Implement the decision tree algorithm and predict on test	

# Decision Tree: splitting the data

- Let's split our dataset `df` into `X` and `y`
- `X` will contain the predictors
- `y` will contain the target variable

```
# Split the data into X and y
columns_to_drop_from_X = ['stroke'] + ['id']
X = df.drop(columns_to_drop_from_X, axis = 1)
y = np.array(df['stroke'])
```



# Data prep: numeric variables

- In Decision Trees, we use **numeric data** as predictors
- We can **convert categorical data to integer values**

```
X = pd.get_dummies(X, columns = ['heart_disease', 'ever_married', 'hypertension', 'Residence_type',  
'gender', 'smoking_status', 'work_type'], dtype=float, drop_first=True)  
print(X.dtypes)
```

```
age                float64  
avg_glucose_level  float64  
heart_disease_1    float64  
ever_married_Yes   float64  
hypertension_1     float64  
Residence_type_Urban float64  
gender_Male        float64  
gender_Other       float64  
smoking_status_never smoked float64  
smoking_status_smokes float64  
work_type_Never_worked float64  
work_type_Private  float64  
work_type_Self-employed float64  
work_type_children float64  
dtype: object
```

# Decision Tree: running the algorithm

- Now let's run our tree on the entire X dataset

```
# Implement the decision tree on X.  
clf = tree.DecisionTreeClassifier()  
clf_fit = clf.fit(X, y)  
  
# Look at our generated model:  
print(clf_fit)
```

```
DecisionTreeClassifier()
```

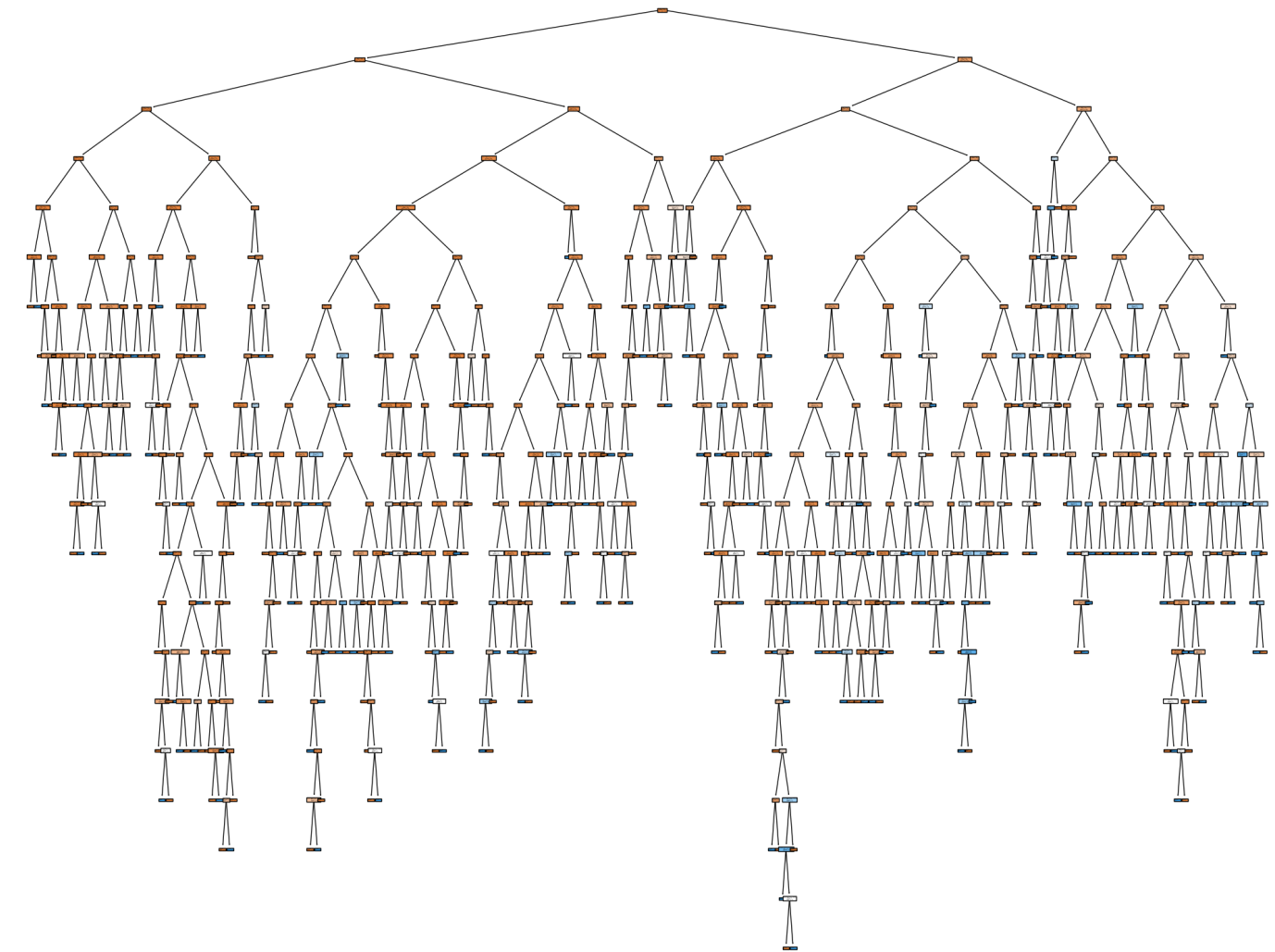
# Visualize: plot\_tree

- We can visualize trees by using `sklearn.tree.plot_tree`
- You can read more about it [here](#)
- We will use it in this module to illustrate the tree we are building

```
# Set figure size
fig = plt.figure(figsize=(25,20))
# Visualize `clf_fit_small`
tree.plot_tree(clf_fit,
               feature_names= X.columns,
               filled=True)

# Save figure
plt.savefig(str(plot_dir)+'tree.png', format='png', bbox
           = "tight")
```

```
plt.show()
```



# Decision Tree:

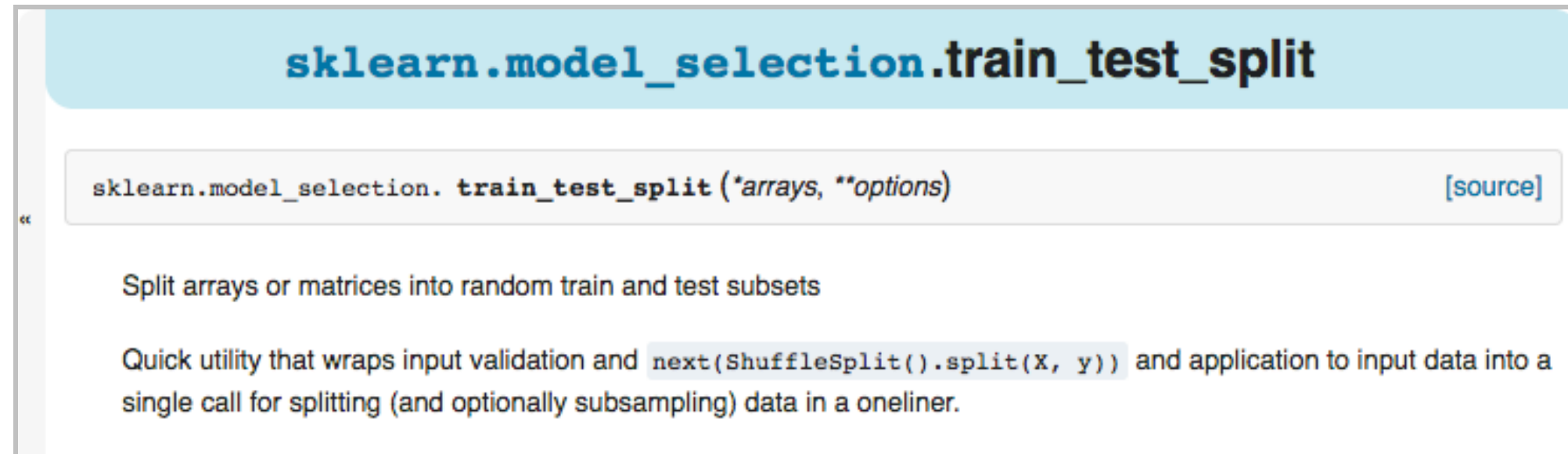
`healthcare-dataset-stroke-data.csv`

- Now, let's run the tree on the full `healthcare-dataset-stroke-data.csv` dataset
- We will evaluate the model and add the results to our `model_final` dataframe



# scikit-learn: train\_test\_split

- We will be using the `train_test_split` library from `scikit-learn`



- Inputs are:
  - Lists, NumPy arrays, scipy-sparse matrices, or Pandas DataFrames
- For all the parameters of the `tree` package, visit [scikit-learn's documentation](#)

# Split into train and test sets

- Split the cleaned data into a train set and test set
- Remember, we already imported the package at the start of this lesson
- Otherwise, we would have to import it now

```
# Split into train and test.  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)  
  
print(X_train.shape, y_train.shape)
```

```
(3577, 14) (3577,)
```

```
print(X_test.shape, y_test.shape)
```

```
(1533, 14) (1533,)
```

# Fit Decision Tree and predict

- Now we will run `tree` on `X_train` and `y_train` and then predict on `X_test`

```
# Implement the decision tree on X_train.
clf = tree.DecisionTreeClassifier()
clf_fit = clf.fit(X_train, y_train)

# Predict on X_test.
y_predict = clf_fit.predict(X_test)
```

- The result is an **array of predictions**

```
y_predict[:20]
```

```
array([False, False, False, False, False, False, False, False, False,
       False, False, False, False, False,  True, False, False, False,
       False, False])
```

- We can determine the accuracy by comparing the predictions against `y_test`

# Knowledge check





# Module completion checklist

Objectives	Complete
Transform data in order to use decision trees	✓
Implement the decision tree algorithm and predict on test	✓

# Congratulations on completing this module!

You are now ready to try Tasks 1-6 in the Exercise for this topic

