# Individual Assignment: Visualising Decision Trees with the Pima Diabetes Dataset

Github link: https://github.com/orons98/Individual-assignment.git

## Introduction

In this tutorial, you will create and visualise a Decision Tree model to determine if individuals in the Pima Diabetes dataset have diabetes. You will train the model and produce a clear, easy-to-understand visualisation of the tree structure. This will help you see how the model makes decisions based on the features in the dataset.

You will learn how Decision Trees split data and why each feature matters in the decision process. You will also understand important parts of the tree, such as nodes, branches, and leaf predictions. By limiting how deep the tree goes, you will gain a better understanding of the model's behaviour and how it predicts diabetes.

### What is a Decision Tree?

"A decision tree is a non-parametric supervised learning technique used for classification and regression applications. It has a hierarchical tree structure, including a root node, branches, internal nodes, and leaf nodes" (IBM, 2024). The nodes indicate decisions made depending on conditions. The branches indicate the consequences of decisions (yes or no, 1 or 0). The leaves represent the results (predictions and labels).

### Pima Indians Diagnosis Dataset

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

## Exercises

### Exercise 1. Importing libraries

First, we must import the necessary libraries. This allows access to use pre-built libraries and functions, saving time and simplifying complex tasks. pandas is used for loading and manipulating the dataset, matplotlib.pyplot is for visualizing the Decision Tree, and tools from sklearn—like train_test_split, DecisionTreeClassifier, and plot_tree—help with splitting data, training the model, and visualizing it. accuracy_score evaluates the model's predictions against the actual outcomes.

```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score)
```

**Exercise 2. Loading and exploring the data**

This code loads the diabetes.csv dataset into a panda DataFrame. It shows the first five rows using `data.head()` to help you understand the dataset's structure. The `data.info()` method provides details about the dataset's columns, their data types, and checks for any missing values. This ensures the data is clean and ready for processing.

```
data = pd.read_csv('diabetes.csv')
print("Dataset Sample:")
print(data.head())
```

```
Dataset Sample:
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

**Exercise 3.i Splitting the dataset**

This section of the code separates the dataset into inputs and outputs for the model:

- x (Input Features): x equals data.drop(columns=['Outcome']) assigns all the columns except Outcome to x. The program will use features such as glucose levels, BMI, and age to predict diabetes.
- y = data['Outcome'] assigns the Outcome column to the target variable, y. This is the aim or value we are attempting to forecast: whether a patient has diabetes (1) or not (0).

The purpose of this stage is to organise the dataset, allowing the model to learn patterns from input features (x) and predict output (y). By separating the features from the target, the model knows what to focus on during training

```
x = data.drop(columns=['Outcome'])
y = data['Outcome']
print("\nFeature Columns:")
print(x.columns)
```

```
Feature Columns:
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age'],
      dtype='object')
```

**Exercise 3.ii Splitting the dataset into inputs and output**

This section of code separates the data into training and testing sets:
x_train and y_train: These are the features (x) and targets (y) used to train the model. This data will be used to train the model to identify patterns.

- x_test and y_test: These are the features and targets being tested. After training, the model will be evaluated with previously unseen data to determine how well it can predict.
- test_size=0.2:
  This sets aside 20% of the dataset for testing, with the remaining 80% utilised for training.
- Using random_state=42 guarantees consistent splits and reproducible results.

Purpose: Splitting data into training and testing sets ensures that the model is trained on one set and tested on another, ensuring fairness and reproducibility. This helps to determine how well the model performs on new, previously unseen data

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state= 42)

print("\nTrain-Test Split Complete")
print(f"Training Samples: {x_train.shape[0]}, Testing Samples: {x_test.shape[0]}")
```

```
Train-Test Split Complete
Training Samples: 614, Testing Samples: 154
```

**Exercise 4. Decision Tree Classifier**

The following code creates and trains a decision tree classifier:

- DecisionTreeClassifier(max_depth=3, random_state=42). This initialises the model with a maximum depth of 3. This constraint prevents the tree from becoming too deep, keeping it basic, interpretable, and easy to visualise.
- The random_state=42 ensures that the tree structure is consistent each time the code is executed.

Training the Classifier:

- tree_classifier.Fit(x_train, y_train) trains the model with the training data (x_train for input features and y_train for target values). During training, the Decision Tree discovers patterns in the data by determining the optimal conditions (splits) for classifying the target variable.
- Output: The line "\nDecision Tree Training Complete" confirms the model's successful training. At this point, the tree is prepared to make predictions based on its learnt patterns.

The purpose of this code is to train a Decision Tree model to categorise data based on basic, interpretable rules from the training dataset. Limiting the depth of the tree prevents overfitting and makes the tree easier to visualise.

```
tree_classifier = DecisionTreeClassifier(max_depth= 3, random_state= 42)
tree_classifier.fit(x_train, y_train)
print("\nDecision Tree Training Complete")
```

```
Decision Tree Training Complete
```

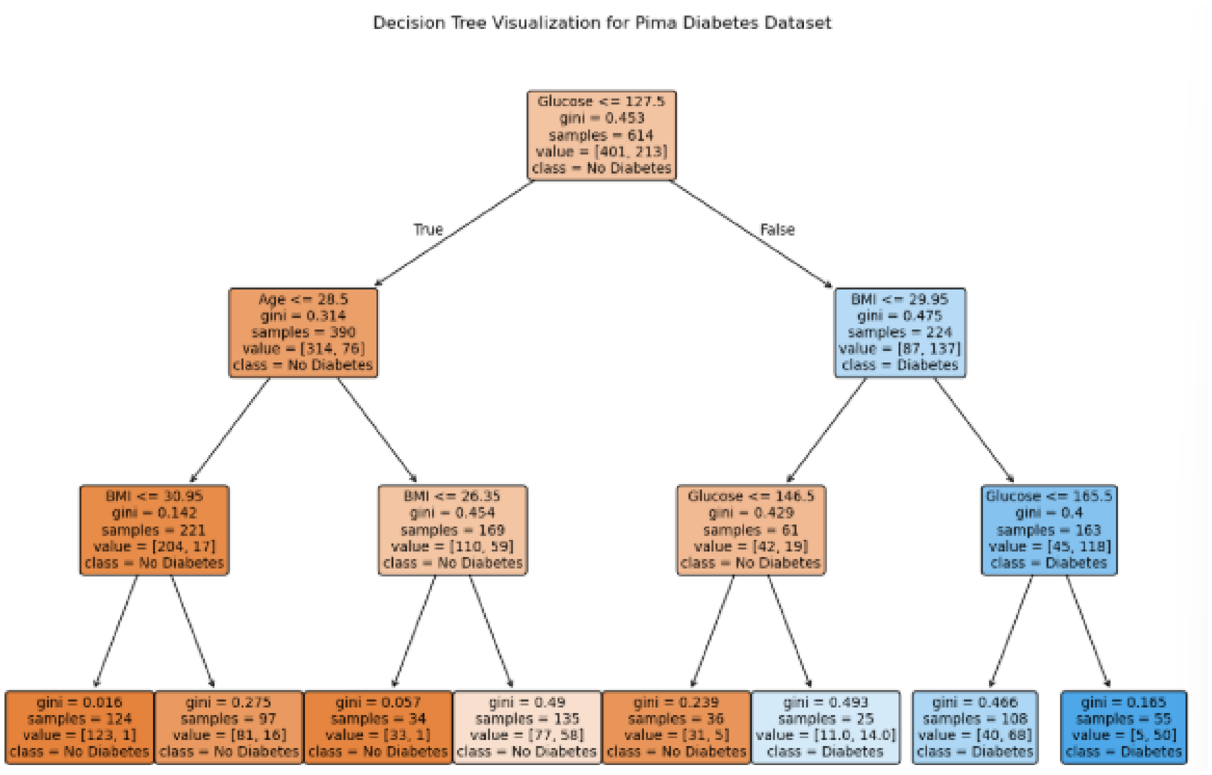
**Exercise 5. Plotting**

The trained Decision Tree is visualised with plot_tree. The feature_names argument labels each node with its relevant feature, whereas class_names labels the classes (No Diabetes or Diabetes). filled=True colours the nodes according to their majority class, and rounded=True adds rounded edges to improve clarity. The plot is displayed using plt.show().

Decision Trees split data based on the most important features using a "divide and conquer" strategy (Rokach & Maimon). For example, in our diabetes dataset, features like "Glucose" and "BMI" might play a key role in making predictions. This method performs well when the dataset contains a few highly relevant features. However, in datasets with complex relationships where many features interact, a Decision Tree might oversimplify the problem, leading to reduced performance.

```
plt.figure(figsize=(15, 10))
plot_tree(tree_classifier,
          feature_names=x.columns,
          class_names=["No Diabetes", "Diabetes"],
          filled=True,
          rounded=True)

plt.title("Decision Tree Visualization for Pima Diabetes Dataset")
plt.show()
```



**Exercise 6. Evaluate the performance of the trained Decision Tree model**

Lastly, we evaluate the performance of the trained **Decision Tree model** on the test data:

The model is evaluated by making predictions on the test dataset using tree_classifier.predict. The accuracy_score function compares these predictions with the actual test outcomes and calculates the accuracy as a percentage, providing insight into the model's performance on unseen data.

- x(Input Features): Line x equals data.drop(columns=['Outcome']) assigns all the columns except Outcome to x. The program will use features such as glucose levels, BMI, and age to predict diabetes.
- The line y = data['Outcome'] assigns the Outcome column to the target variable, y. This is the aim or value we are attempting to forecast: whether a patient has diabetes (1) or not (0).

The purpose of this stage is to organise the dataset, allowing the model to learn patterns from input features (x) and predict output (y). By separating the features from the objective, the model knows what to concentrate on during training.

```
y_pred = tree_classifier.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy on Test Data: {accuracy * 100:.2f}%")
```

```
Model Accuracy on Test Data: 75.97%
```

## References

- (No date) What is a decision tree? Available at: https://www.ibm.com/think/topics/decision-trees (Accessed: 12 December 2024).
- Blockeel, H. et al. (2023) ‘Decision trees: From efficient prediction to responsible AI’, Frontiers in Artificial Intelligence, 6. doi:10.3389/frai.2023.1124553.
- Rokach, L. and Maimon, O. (no date) ‘Decision trees’, Data Mining and Knowledge Discovery Handbook, pp. 165–192. doi:10.1007/0-387-25465-x_9.
- ‘Pima Indians Diabetes Database’ (no date).