**Steps to implement decision tree using following code:**

1. After extracting the given zip folder put the ‘DecisionTreeTools.py’ , ‘Classifier.py’ and the ‘titanic.csv’ in a single folder.
2. The given dataset is of titanic which contains details of passengers and whether they survived or not. The given ‘Classifier.py’ helps to classify using the ‘DecisionTreeTools.py’ whether the passenger with given details survived or not the titanic’s accident using Decision Tree algorithm.

Functions in DecisionTreeTools.py:

1. train\_split(dataset, feature, threshold):

* Splits the given dataset on the feature given.
* Returns a vector of the different data subsets as pandas DataFrames, indexed in numerically increasing order based on the class value.

1. inform\_gain(data\_vec):

* Calculates the information gain of a given split dataset.
* Data is given as a vector of pandas DataFrames so that both continuous and nominal data can be handled the same way.
* Calculate the entropy using sum (-P(y)log.2.(P(y)) for all vals.
* inform\_gain = entropy - conditional\_entropy

1. calc\_best\_split(dataset):

* Iterate through all possible splits on the dataset provided and return the split with lowest information gain.
* Only input needed is the dataset to split.
* Output is dict: feature to split on, best threshold for cont. features, and a data vector containing the split data.

1. recursive\_split(node, max\_depth, min\_size, current\_depth):

* Recursive splitting function used to split on them splits.
* Inputs are current tree depth, minimum data size, max depth, and current node.
* Outputs are nothing, just add child\_num:child\_node key:val pairs to current node and move the data along to those children.

1. def build\_decision\_tree(dataset, max\_depth, min\_size):

* It's time to build the tree!
* Inputs include base dataset, max depth of tree, and minimum data size @ node.
* Output is the root node of the tree!

1. def print\_tree(node, depth):

* Print out the tree for visualizationz!
* Inputs are starting node for visualizing and the depth.
* Returns nothing but outputs a cool indented hierarchy of nodes on terminal!

\*\*Functions in Classifier.py are explained in the python file itself.

**Outputs:**

runfile('C:/Users/Nisar/Desktop/DecisionTreeClassifier-master/DecisionTreeClassifier-master/DecisionTreeTools.py', wdir='C:/Users/Nisar/Desktop/DecisionTreeClassifier-master/DecisionTreeClassifier-master')

runfile('C:/Users/Nisar/Desktop/DecisionTreeClassifier-master/DecisionTreeClassifier-master/Classifier.py', wdir='C:/Users/Nisar/Desktop/DecisionTreeClassifier-master/DecisionTreeClassifier-master')

Feature: Fare Values: [14] Data Len: 2

Depth: 1 Children: 2

Feature: Age Values: [23] Data Len: 2

Depth: 2 Children: 2

Feature: Age Values: [12] Data Len: 2

Depth: 3 Children: 2

Feature: Fare Values: [7] Data Len: 2

Depth: 4 Children: 2

Feature: Fare Values: [7] Data Len: 2

Depth: 5 Children: 2

Feature: Fare Values: [7] Data Len: 2

Depth: 6 Children: 2

Feature: Sex\_n Values: [1] Data Len: 2

Depth: 6 Children: 2

Feature: Fare Values: [8] Data Len: 2

Depth: 5 Children: 2

Feature: Fare Values: [8] Data Len: 2

Depth: 6 Children: 2

Feature: Fare Values: [7] Data Len: 2

Depth: 4 Children: 2

Feature: Age Values: [20] Data Len: 2

Depth: 5 Children: 2

Feature: Fare Values: [7] Data Len: 2

Depth: 6 Children: 2

Feature: Age Values: [22] Data Len: 2

Depth: 6 Children: 2

Feature: Fare Values: [9] Data Len: 2

Depth: 5 Children: 2

Feature: Age Values: [20] Data Len: 2

Depth: 6 Children: 2

Feature: Sex\_n Values: [1] Data Len: 2

Depth: 6 Children: 2

Feature: Age Values: [32] Data Len: 2

Depth: 3 Children: 2

Feature: Fare Values: [8] Data Len: 2

Depth: 4 Children: 2

Feature: Fare Values: [7] Data Len: 2

Depth: 5 Children: 2

Feature: Fare Values: [7] Data Len: 2

Depth: 6 Children: 2

Feature: Age Values: [27] Data Len: 2

Depth: 6 Children: 2

Feature: Age Values: [28] Data Len: 2

Depth: 5 Children: 2

Feature: Age Values: [25] Data Len: 2

Depth: 6 Children: 2

Feature: Age Values: [30] Data Len: 2

Depth: 6 Children: 2

Feature: Age Values: [40] Data Len: 2

Depth: 4 Children: 2

Feature: Age Values: [34] Data Len: 2

Depth: 5 Children: 2

Feature: Fare Values: [8] Data Len: 2

Depth: 6 Children: 2

Feature: Fare Values: [8] Data Len: 2

Depth: 6 Children: 2

Feature: Age Values: [48] Data Len: 2

Depth: 5 Children: 2

Feature: Age Values: [43] Data Len: 2

Depth: 6 Children: 2

Feature: Pclass Values: [3] Data Len: 2

Depth: 6 Children: 2

Feature: Age Values: [27] Data Len: 2

Depth: 2 Children: 2

Feature: Age Values: [7] Data Len: 2

Depth: 3 Children: 2

Feature: Fare Values: [27] Data Len: 2

Depth: 4 Children: 2

Feature: Fare Values: [20] Data Len: 2

Depth: 5 Children: 2

Feature: Fare Values: [15] Data Len: 2

Depth: 6 Children: 2

Feature: Fare Values: [25] Data Len: 2

Depth: 6 Children: 2

Feature: Fare Values: [46] Data Len: 2

Depth: 5 Children: 2

Feature: Age Values: [2] Data Len: 2

Depth: 6 Children: 2

Feature: Sex\_n Values: [1] Data Len: 2

Depth: 6 Children: 2

Feature: Fare Values: [36] Data Len: 2

Depth: 4 Children: 2

Feature: Age Values: [19] Data Len: 2

Depth: 5 Children: 2

Feature: Age Values: [11] Data Len: 2

Depth: 6 Children: 2

Feature: Fare Values: [24] Data Len: 2

Depth: 6 Children: 2

Feature: Age Values: [21] Data Len: 2

Depth: 5 Children: 2

Feature: Fare Values: [73] Data Len: 2

Depth: 6 Children: 2

Feature: Age Values: [24] Data Len: 2

Depth: 6 Children: 2

Feature: Fare Values: [33] Data Len: 2

Depth: 3 Children: 2

Feature: Age Values: [37] Data Len: 2

Depth: 4 Children: 2

Feature: Age Values: [32] Data Len: 2

Depth: 5 Children: 2

Feature: Fare Values: [26] Data Len: 2

Depth: 6 Children: 2

Feature: Age Values: [35] Data Len: 2

Depth: 6 Children: 2

Feature: Age Values: [45] Data Len: 2

Depth: 5 Children: 2

Feature: Age Values: [41] Data Len: 2

Depth: 6 Children: 2

Feature: Age Values: [55] Data Len: 2

Depth: 6 Children: 2

Feature: Age Values: [42] Data Len: 2

Depth: 4 Children: 2

Feature: Fare Values: [80] Data Len: 2

Depth: 5 Children: 2

Feature: Fare Values: [55] Data Len: 2

Depth: 6 Children: 2

Feature: Age Values: [36] Data Len: 2

Depth: 6 Children: 2

Feature: Fare Values: [76] Data Len: 2

Depth: 5 Children: 2

Feature: Age Values: [49] Data Len: 2

Depth: 6 Children: 2

Feature: Age Values: [52] Data Len: 2

Depth: 6 Children: 2

Output of the tree:

DecisionTreeTools.print\_tree(tree\_root, 0)

[Fare < 14.000]

[Age < 23.000]

[Age < 12.000]

[Fare < 7.000]

[Fare < 7.000]

[Fare < 7.000]

[825]

[859]

[Sex\_n < 1.000]

[727]

[828]

[Fare < 8.000]

[878]

[Fare < 8.000]

[837]

[869]

[Fare < 7.000]

[Age < 20.000]

[Fare < 7.000]

[875]

[877]

[Age < 22.000]

[762]

[554]

[Fare < 9.000]

[Age < 20.000]

[844]

[840]

[Sex\_n < 1.000]

[882]

[876]

[Age < 32.000]

[Fare < 8.000]

[Fare < 7.000]

[Fare < 7.000]

[884]

[805]

[Age < 27.000]

[870]

[756]

[Age < 28.000]

[Age < 25.000]

[864]

[886]

[Age < 30.000]

[883]

[814]

[Age < 40.000]

[Age < 34.000]

[Fare < 8.000]

[890]

[800]

[Fare < 8.000]

[847]

[812]

[Age < 48.000]

[Age < 43.000]

[865]

[873]

[Pclass < 3.000]

[772]

[851]

[Age < 27.000]

[Age < 7.000]

[Fare < 27.000]

[Fare < 20.000]

[Fare < 15.000]

[760]

[831]

[Fare < 25.000]

[888]

[711]

[Fare < 46.000]

[Age < 2.000]

[839]

[850]

[Sex\_n < 1.000]

[863]

[846]

[Fare < 36.000]

[Age < 19.000]

[Age < 11.000]

[852]

[791]

[Fare < 24.000]

[858]

[889]

[Age < 21.000]

[Fare < 73.000]

[853]

[802]

[Age < 24.000]

[742]

[710]

[Fare < 33.000]

[Age < 37.000]

[Age < 32.000]

[Fare < 26.000]

[874]

[848]

[Age < 35.000]

[657]

[701]

[Age < 45.000]

[Age < 41.000]

[885]

[854]

[Age < 55.000]

[862]

[694]

[Age < 42.000]

[Fare < 80.000]

[Fare < 55.000]

[867]

[838]

[Age < 36.000]

[759]

[835]

[Fare < 76.000]

[Age < 49.000]

[871]

[745]

[Age < 52.000]

[856]

[879]

Output of the Prediction:

example = parsed\_df.iloc[0]

def classify\_example(example, tree\_root):

question = list(tree\_root.items())[0][1]

value = list(tree\_root.items())[1][1]

# ask question

if example[question] <= float(value):

answer = list(list(tree\_root.items())[2][1].items())[0][1]

else:

answer = list(list(tree\_root.items())[2][1].items())[1][1]

# base case

if not isinstance(answer, dict):

return answer

# recursive part

else:

residual\_tree = answer

return classify\_example(example, residual\_tree)

val = classify\_example(example, tree\_root)

if (target[val] == 0):

print("The passenger died!")

else:

print("The passenger survived")

**The passenger died!**

**######################################################**