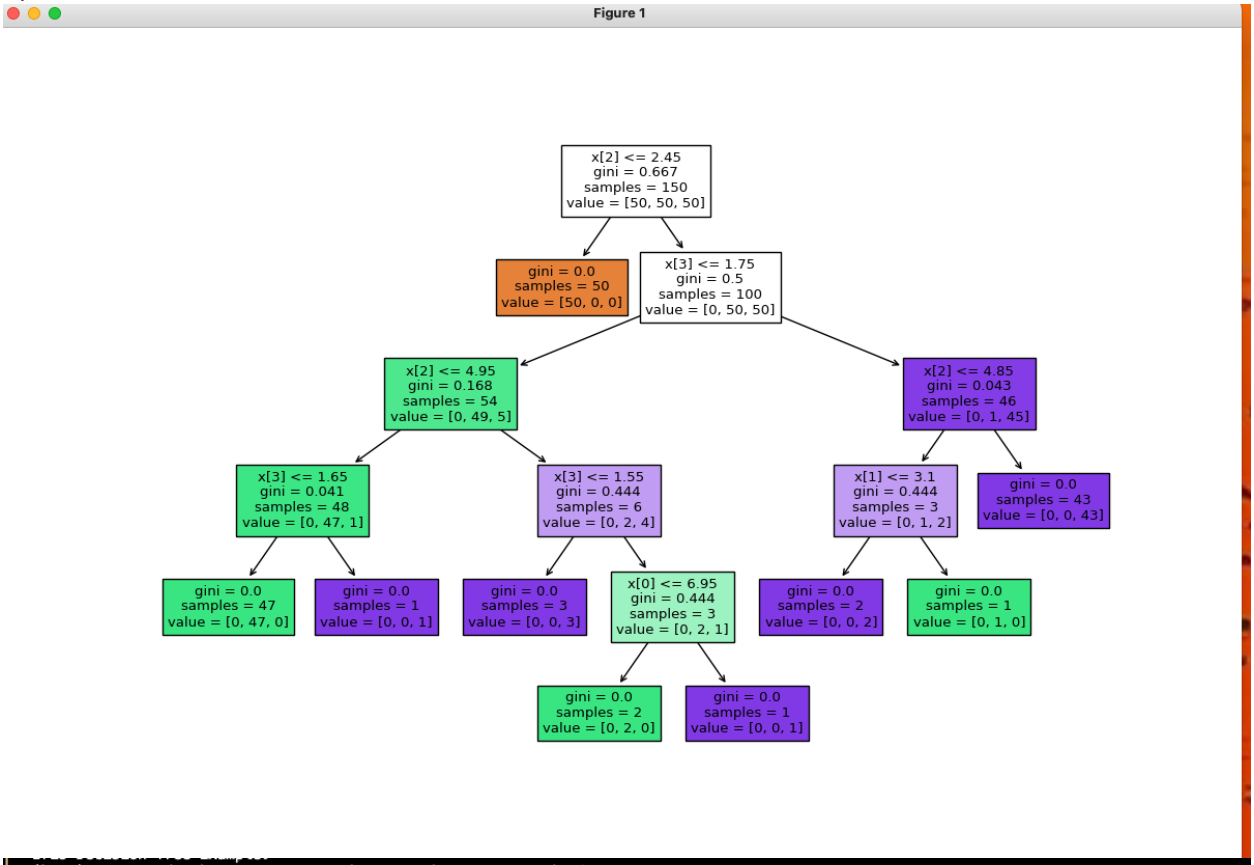
Name: Marcos Ondruska
Student # 2685885
Assignment 4

2) Decision trees:
Advantages:
Trees can be visualized – means that conceptually because you can picture it, it's something that is definable and therefore can be understood. There is a simplicity to handling the data in that it requires very little preparation, given that there are certain trees and algorithm combinations that can overcome missing values in the data. The cost of a decision tree is relatively inexpensive in that the cost of prediction is logarithmic relative to the amount of data that is used to train the tree. Trees have the advantage of being able to be categorized by both numerical and categorical data. Examples of categorical data are pretty much infinite, some of which might be items such as type of cuisine, Job Title, Sport type, care brand etc. Scikit-learn does not handle categorical data at this stage, but other libraries and algorithms do enable the categorical functionality. Multi-Output problems, which involve predicting many outputs at the same time using a given set of input features, are also supported. Any situation that is observable in a model is able to be explained through the use of Boolean logic. It is possible to test the reliability of a model using statistical methods. Performs well even in real life situations where exogenous items broaden the scope of the assumptions that were modeled.

Disadvantages:
When creating complex Decision trees, overfitting might be the result. This requires techniques like pruning, setting constraints such as maximum depth and minimum number of samples at leaf nodes. Small changes in the data can create very different trees, and cause instability. Ensemble methods mitigate this type of issue. Predictions are piecewise constant, which means that they are neither smooth nor continuous, and are therefore less useful for extrapolation. The optimal decision tree is NP-complete, meaning it relies on heuristic methods that may not find the globally optimal tree. Concepts such as XOR, parity or multiplexer problems are not expressed easily and are therefore difficult to learn. It is recommended to balance the dataset prior to training as some classes might dominate the dataset leading to a bias in the decision tree.

3) The Iris flower data set, is also known as Fisher's Iris data set and was made famous in 1936 by British statistician Ronald Fisher. The data set contains 150 samples that consist of three related Iris species and are classified according to four features – sepal length, sepal width, petal length, petal width. Fisher developed a model, called a linear discriminatory model using these features to distinguish the species. This dataset has become a core set that is used for testing various machine learning algorithms, as its class separation is obvious, and structure is straightforward making it a great choice as a beginner's dataset. This dataset is integrated in R and Python's scikit-learn library, detailing its comprehensive use for teaching and testing, particularly in classification tasks.

4)




```
(base) marcosondruska@Marcoss-MacBook-Pro assignment4 % python3
Python 3.8.1rc1 (v3.8.1rc1:b00a2b5b76, Dec  9 2019, 18:13:08)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from sklearn import tree
X = [[0, 0], [1, 1]]
Y = [0, 1]
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, Y)>>> X = [[0, 0], [1, 1]]
>>> Y = [0, 1]
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(X, Y)
>>> clf.predict([[2., 2.]])
array([1])
>>> clf.predict_proba([[2., 2.]])
array([[0., 1.]])
>>> from sklearn.datasets import load_iris
from sklearn import tree
iris = load_iris()
X, y = iris.data, iris.target
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, y)>>> from sklearn import tree
>>> iris = load_iris()
>>> X, y = iris.data, iris.target
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(X, y)
>>> tree.plot_tree(clf)
[Text(0.5, 0.9166666666666666, 'x[2] <= 2.45\ngini = 0.667\nsamples = 150\nvalue = [50, 50, 50]'), Text(0.4230769230769231, 0.75, 'gini =
 0.0\nsamples = 50\nvalue = [50, 0, 0]'), Text(0.5769230769230769, 0.75, 'x[3] <= 1.75\ngini = 0.5\nsamples = 100\nvalue = [0, 50, 50]'),
 Text(0.3076923076923077, 0.5833333333333334, 'x[2] <= 4.95\ngini = 0.168\nsamples = 54\nvalue = [0, 49, 5]'), Text(0.15384615384615385,
0.4166666666666667, 'x[3] <= 1.65\ngini = 0.041\nsamples = 48\nvalue = [0, 47, 1]'), Text(0.07692307692307693, 0.25, 'gini = 0.0\nsamples
 = 47\nvalue = [0, 47, 0]'), Text(0.23076923076923078, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [0, 0, 1]'), Text(0.46153846153846156, 0.4
166666666666667, 'x[3] <= 1.55\ngini = 0.444\nsamples = 6\nvalue = [0, 2, 4]'), Text(0.38461538461538464, 0.25, 'gini = 0.0\nsamples = 3\
nvalue = [0, 0, 3]'), Text(0.5384615384615384, 0.25, 'x[2] <= 5.45\ngini = 0.444\nsamples = 3\nvalue = [0, 2, 1]'), Text(0.46153846153846
156, 0.08333333333333333, 'gini = 0.0\nsamples = 2\nvalue = [0, 2, 0]'), Text(0.6153846153846154, 0.08333333333333333, 'gini = 0.0\nsampl
es = 1\nvalue = [0, 0, 1]'), Text(0.8461538461538461, 0.5833333333333334, 'x[2] <= 4.85\ngini = 0.043\nsamples = 46\nvalue = [0, 1, 45]')
, Text(0.7692307692307693, 0.4166666666666667, 'x[0] <= 5.95\ngini = 0.444\nsamples = 3\nvalue = [0, 1, 2]'), Text(0.6923076923076923, 0.
25, 'gini = 0.0\nsamples = 1\nvalue = [0, 1, 0]'), Text(0.8461538461538461, 0.25, 'gini = 0.0\nsamples = 2\nvalue = [0, 0, 2]'), Text(0.9
230769230769231, 0.4166666666666667, 'gini = 0.0\nsamples = 43\nvalue = [0, 0, 43]')]
>>>
```

Explanation of code:

In line 1 we import the tree module from the library sklearn

In lines 2 and 3 we define the X variable as a list of the feature vectors, and Y is defined as a list of the output data. This data is used to train the decision tree model.
In line 4 we create an object clf that is an instance of the DecisionTreeClassifier class.
In line 5 clf is = the result of the fit method that is called on the clf object with the X and Y as the arguments.

In line 6 (clf.predict([[2., 2.]])) we ask the classifier clf to predict the class label with a new data point that has the features “[2.0. 2.0]”

In line 7 (clf.predict_proba([[2., 2.]])) we ask for the predicted probabilities for a new datapoint that has the features “[2.0, 2.0]”

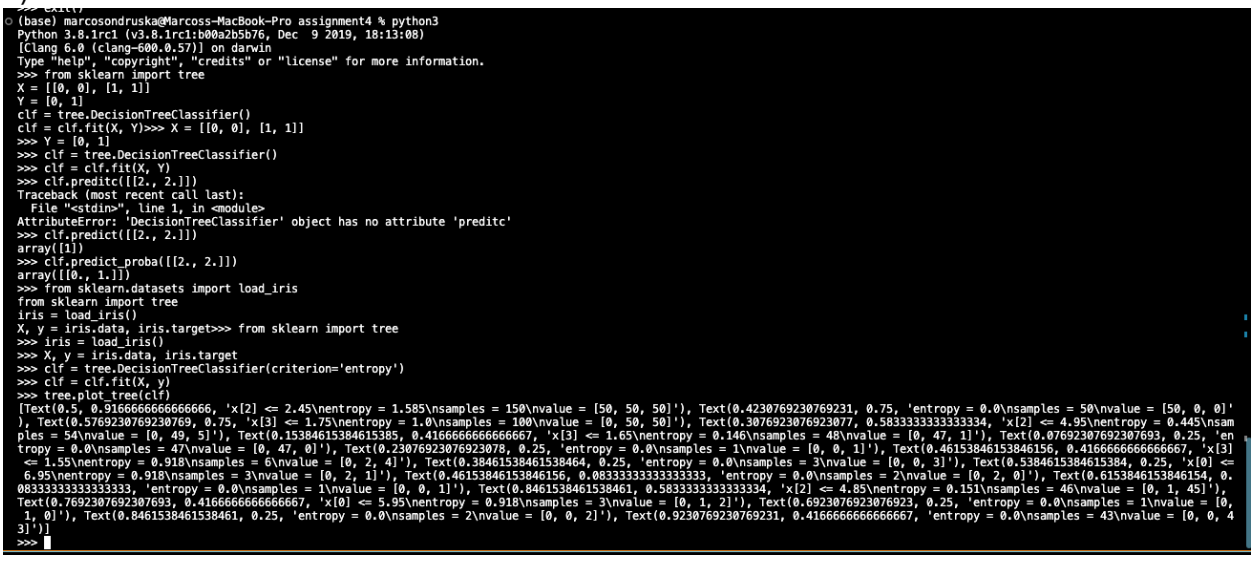
In the following section we:

```
from sklearn.datasets import load_iris
from sklearn import tree
iris = load_iris()
X, y = iris.data, iris.target
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, y)
```

Import the load_iris dataset, as well as the tree modules
Assign the iris dataset to variable iris
Assign X to the iris.data
Assign y to the iris.target
Create and instance of a DecisionTreeClasifier that is assigned to clf
Train the decision tree model with the Iris dataset.

With tree.plot_tree(clf) we call the “plot_tree” function in scikit-learn. This gives us the data output.

5)



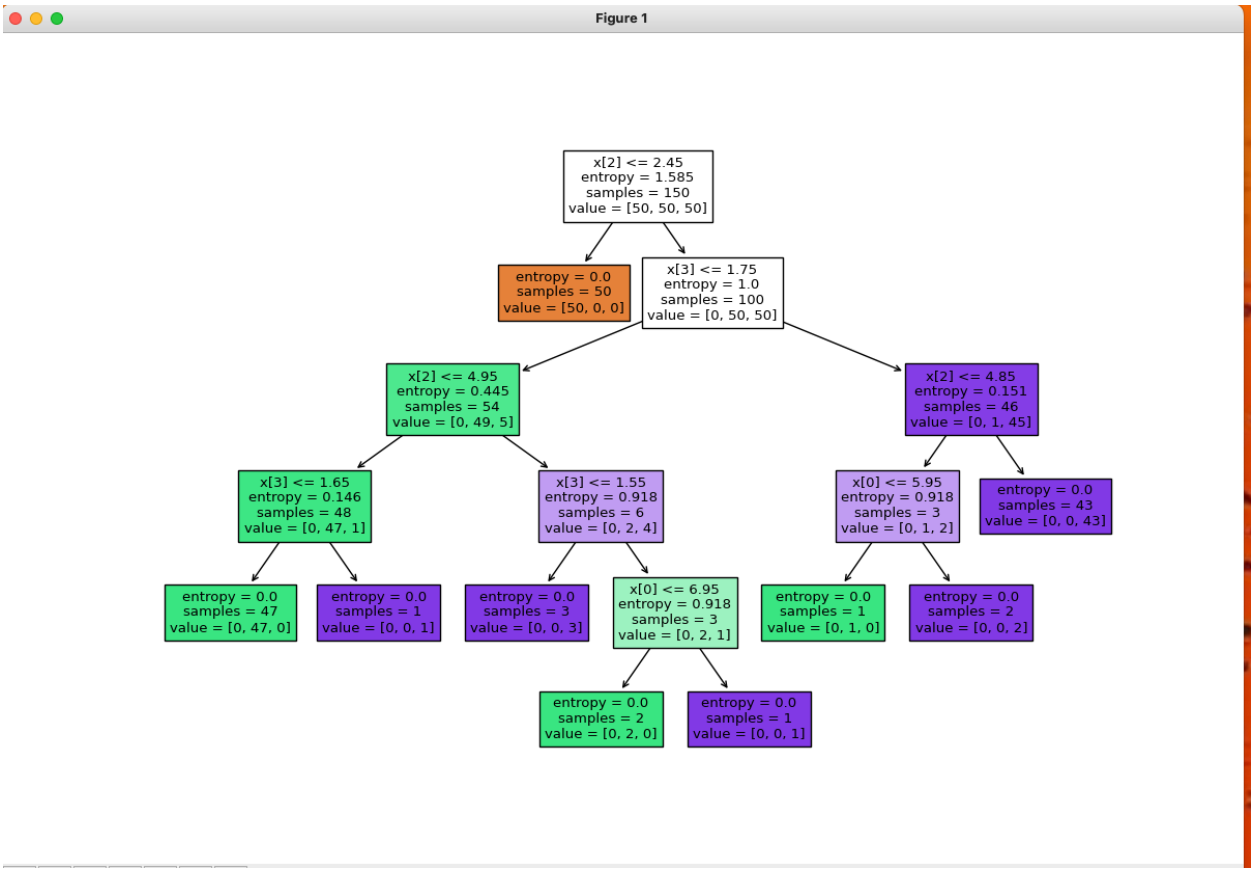


Figure 1
x[2] <= 2.45
entropy = 1.585
samples = 150
value = [50, 50, 50]
entropy = 0.0
samples = 50
value = [50, 0, 0]
x[3] <= 1.75
entropy = 1.0
samples = 100
value = [0, 50, 50]
x[2] <= 4.95
entropy = 0.445
samples = 54
value = [0, 49, 5]
x[2] <= 4.85
entropy = 0.151
samples = 46
value = [0, 1, 45]
x[3] <= 1.65
entropy = 0.146
samples = 48
value = [0, 47, 1]
x[3] <= 1.55
entropy = 0.918
samples = 6
value = [0, 2, 4]
x[0] <= 5.95
entropy = 0.918
samples = 3
value = [0, 1, 2]
entropy = 0.0
samples = 43
value = [0, 0, 43]
entropy = 0.0
samples = 47
value = [0, 47, 0]
entropy = 0.0
samples = 1
value = [0, 0, 1]
entropy = 0.0
samples = 3
value = [0, 0, 3]
x[0] <= 6.95
entropy = 0.918
samples = 3
value = [0, 2, 1]
entropy = 0.0
samples = 1
value = [0, 1, 0]
entropy = 0.0
samples = 2
value = [0, 0, 2]
entropy = 0.0
samples = 2
value = [0, 2, 0]
entropy = 0.0
samples = 1
value = [0, 0, 1]