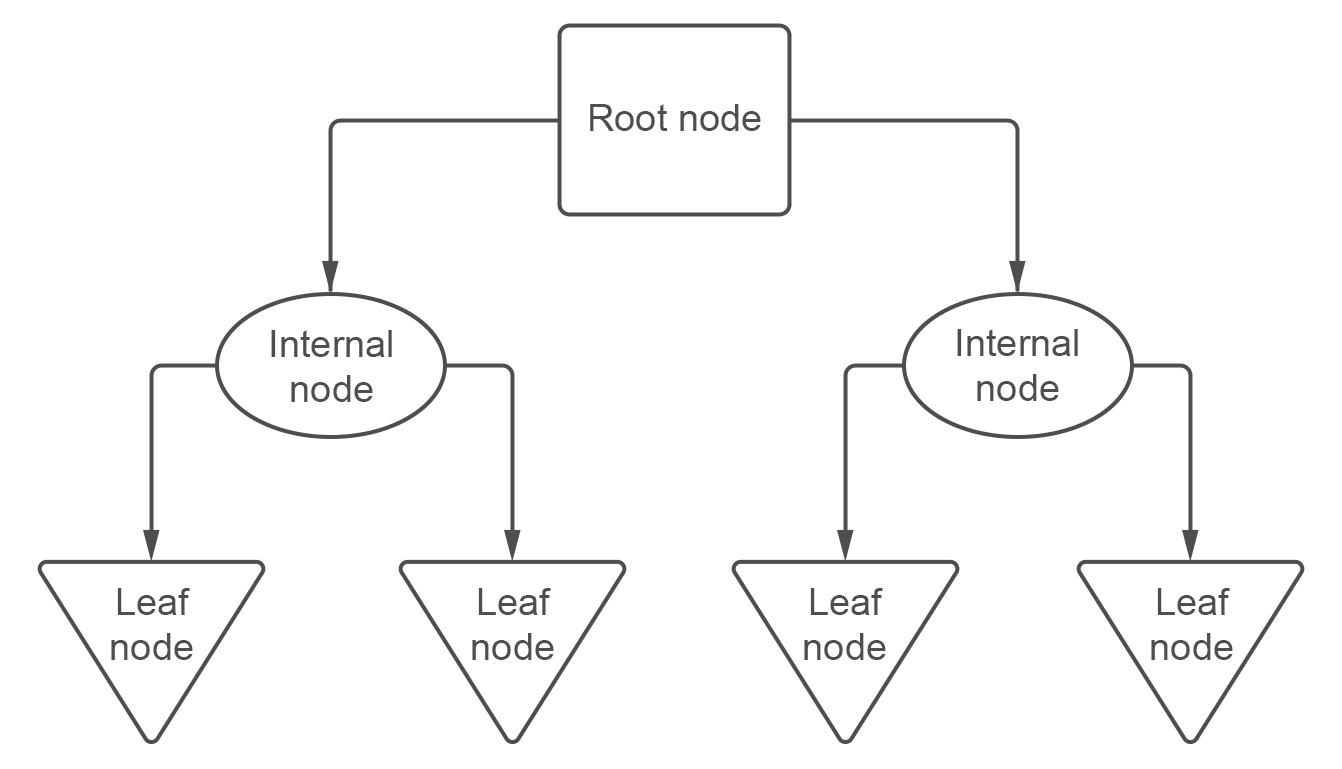
**Module 14**

**Decision Trees**

1. Build a decision tree manually
2. Train a decision tree model with desired hyperparameters using scikit-learn
3. Visualize a decision tree
4. Evaluate overfitting of decision trees
5. Evaluate decision tree splits
6. Implement the decision tree algorithm
7. Compare the performance of different grid search algorithms

* [Video Transcripts](https://student.emeritus.org/courses/4765/files/3353899?wrap=1)
* [Download Video Transcripts](https://student.emeritus.org/courses/4765/files/3353899/download?download_frd=1)
* [Quick Reference Guide](https://student.emeritus.org/courses/4765/files/3353900?wrap=1)



**Notes:**

Glossary

**Branch**

A connection between nodes

**Entropy**

The degree of impurity or uncertainty in a set of observations

**Internal Node**

A node that has two or more branches and can be split further

**Leaf Node**

A node that can no longer be split

**Pruning**

A set of techniques to ‘trim’ branches from the decision tree, making the tree faster to compute

**Root Node**

The beginning node of a decision tree

**Weighted Entropy**

The entropy of a node multiplied by the fraction of samples in that node

X2 = X[['flipper\_length\_mm', 'bill\_length\_mm']]

dtree = DecisionTreeClassifier(max\_depth = 2).fit(X2, y)

tree2 = export\_text(dtree, feature\_names = list(X2.columns))

### ANSWER CHECK

print(tree2)

sns.scatterplot(data=X2, x='flipper\_length\_mm', y='bill\_length\_mm', hue=y)

plt.axvline(x=206.50, color="black")

plt.axhline(y=43.35, xmin=0, xmax=0.575, color="black")

plt.axhline(y=40.85, xmin=0.575, xmax=1, color="black")

dtree3 = DecisionTreeClassifier(max\_depth = 3).fit(X2, y)

tree3 = export\_text(dtree3, feature\_names = list(X2.columns))

prediction = dtree3.predict(pd.DataFrame([[209, 41.2]], columns=X2.columns))

import ssl

ssl.\_create\_default\_https\_context = ssl.\_create\_unverified\_context

**Module Issues:**

Codio 14.4 Problem 4: must to declare those variables:

**ent\_age1 = -(1 \* np.log2(1))**

**ent\_age2 = -(1/4 \* np.log2(1/4) + 3/4 \* np.log2(3/4))**

Codio 14.4 Problem 5: parameter should be **criterion = 'entropy'**

Codio 14.6 Problem 3: Do not include the last element in ccp\_alphas in the for loop: **for i in ccp\_alphas[:-1]**

Codio 14.6 Problem 4: Change to **plt.step(ccp\_alphas, depths, '--o')**

Codio 14.6 marked as Codio 14.7 in the activity.

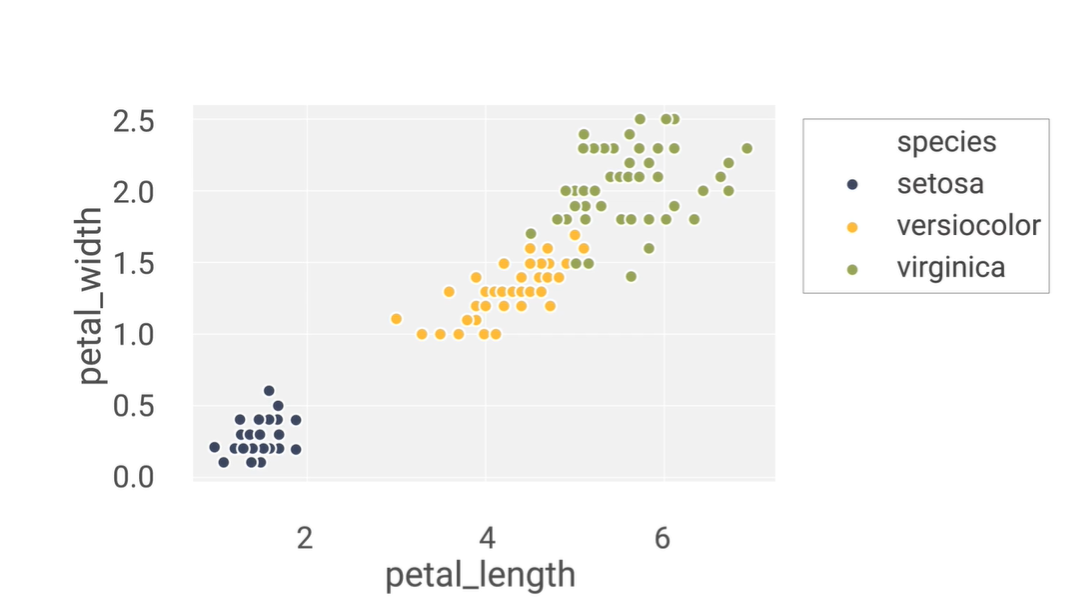
**Quizes:**

A decision tree is a tree of questions that should be answered arbitrarily to yield a predictive classification. : False

*You are correct! The answer “*False*” is correct because a decision tree is a tree of questions that should be answered in sequence to yield a predictive classification.*

The decision boundaries of a logistic regression model are linear. : True

*You are correct! The answer “*True*” is correct because the decision boundaries of a logistic regression model are linear.*



Consider this plot with petal\_length on the x-axis and petal\_width on the y-axis. For a decision rule “petal\_width < 0.75 and petal\_length < 2” equal to yes, what class would be chosen? : setosa

*You are correct! The answer “*setosa*” is correct because in this range of decision rules, setosa is the only class present, thus making setosa the best prediction.*

What is the correct Python statement to import a decision tree model from scikit-learn? : from sklearn import tree

*You are correct! The answer “*from sklearn import tree*” is correct because this is the correct Python statement to import decision tree models from scikit-learn.*

“d\_t=tree.DecisionTree(criterion=’entropy’)”

The given Python code is the correct code for building a decision tree classifier. : False

*You are correct! The answer “*False*” is correct because the correct function for building a decision tree classifier is* *“*DecisionTreeClassifier(criterion=’entropy’)*”.*

Which of the following is not a constructor used in “tree.plot\_tree()” function? : criterion

*You are correct! The answer “*criterion*” is correct because this is the constructor of the “*DecisionTreeClassifier()*” not for “*plot\_tree()” *function.*

Seaborn is the best library to visualize a decision tree. : False

*You are correct! The answer “*False*” is correct because the Python library used to visualize the decision tree plot with better layout is Graphviz.*

A node from a decision tree

|  |
| --- |
| petal\_width ≤ 1.75 |
| entropy = 1.0 |
| samples = 100 |
| value = [0, 50, 50] |
| class = versicolor |

Given a single node of a decision tree built with class\_names=[“a”,“b”,“c”], how many samples belong to class “a” at this node? : 0

*You are correct! The answer “*0*” is correct because the parameter ‘value’ at the specific node tells the number of samples belonging to each class. Since there are three classes (“*a*”, “*b*”, and “*c*”) and the ‘value’ parameter here says [0, 50, 50], that means there are 0 samples in class “*a*” (and 50 each in “*b*” and “*c*”).*

What is the correct Python function that can be used to measure the accuracy of a decision tree classifier? : accuracy\_score(predictions,original\_data)

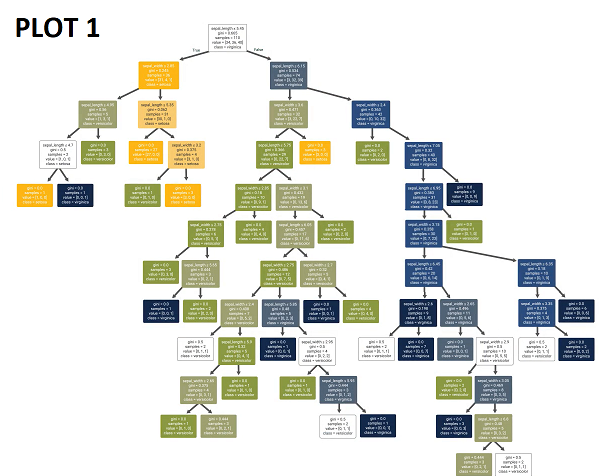
*You are correct! The answer “*accuracy\_score(predictions,original\_data)*” is correct because this is the correct Python function used to measure the accuracy of a decision tree classifier.*

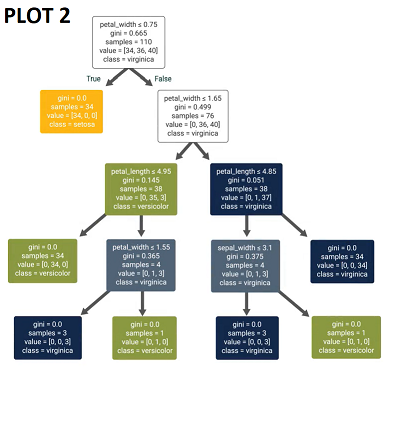
Scikit-learn decision trees always have perfect accuracy on the training data, except when there are samples from the same classes with the exact same features. : False

*You are correct! The answer “*False*” is correct because scikit-learn decision trees always have perfect accuracy on the training data except when there are samples from different classes with the exact same features.*

The training accuracy for a decision tree classifier can be 100% when the data points overlap exactly. : False

*You are correct! The answer “*False*” is correct because the training accuracy for a decision tree classifier can be 100% when the data points do not overlap exactly.*





Consider the two decision trees above. Which of the plots seems to show overfitting? : Plot 1

*You are correct! The answer “*Plot 1*” is correct because it is highly complex, and the more complex the plot, the higher the chance of overfitting. Plot 1 seems to be more complex with more chances of overfitting.*

For decision tree classifiers, more features do not necessarily lead to overfitting. : True

*You are correct! The answer “*True*” is correct because more features do not lead to overfitting but can help in resolving the difference between the classes.*

The very first node of a decision tree is known as the leaf node. : False

*You are correct! The answer “*False*” is correct because the very first node of a decision tree is known as the root node.*

In the decision tree algorithm, after selecting the best feature “x” and the best split value “β”, the data is split into two nodes. One of the nodes has a rule where x < β. What is the rule for the other node? : x >= β

*You are correct! The answer “*x >= β*” is correct because after selecting “x” and “*β”,*the data is split into two nodes, one where x < β and one where*x >= β.

A node in a decision tree that has overlapping data points from different classes and thus cannot be split is called a “pure” node. : False

*You are correct! The answer “*False*” is correct because a node in the decision tree that has overlapping data points from different classes and thus cannot be split is called an ‘unsplittable’ node*.

A node from a decision tree

|  |
| --- |
| sepal\_length ≤ 5.45 |
| gini = 0.665 |
| samples = 110 |
| value = [24, 36, 40] |
| class = virginica |

For this given node of a decision tree classifier, what would the value of “p2” be? : 0.36

*You are correct! The answer “*0.36*” is correct because p2 would be calculated as 40/110 which comes out to be “0.36”.*

What is the entropy (S) for a node with p values of [“p0=0.31”, “p1=0.33”, “p2=0.36”]? : 1.58

*You are correct! The answer “*1.58*” is correct because with the given values, the formula for entropy gives*

*−0.31 log2 0.31 − 0.33 log2 0.33 − 0.36 log2 0.36, which comes out to be 1.58.*

A node in a decision tree where data is evenly split between two classes has entropy 1. : True

*You are correct! The answer “*True*” is correct because with evenly split data for two classes, the formula for entropy is*

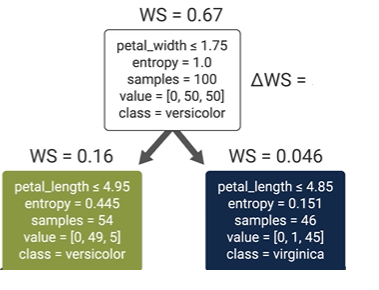
*−0.5 log2 0.5 − 0.5 log2 0.5, which is equal to 1.*

The weighted entropy (WS) of a node is its entropy scaled by the fraction of samples in that node. : True

*You are correct! The answer “*True*” is correct because the weighted entropy (WS) of a node is its entropy multiplied by the fraction of samples in that node divided by the total samples.*

The weighted entropy of the root node is equal to its entropy. : True

*You are correct! The answer “*True*” is correct because the weighted entropy is given as the entropy multiplied by the fraction of samples in that node, and the root node's fraction is 1 (n/n). Therefore, the entropy and the weighted entropy of the root node will always be the same.*



From the given information, what would ΔWS be for the root node? : 0.46

*You are correct! The answer “*0.46*” is correct because the ΔWS is given as the WS of the node minus the WS of the child nodes, which is 0.67 − (0.16 + 0.046) = 0.46.*

The decision tree classifier chooses the best feature and the best split such that the ΔWS is minimized. : False

*You are correct! The answer “*False*” is correct because the decision tree classifier chooses the best feature and the best split such that the ΔWS is maximized.*

The decision tree classifier can use L1 and L2 regularization. : False

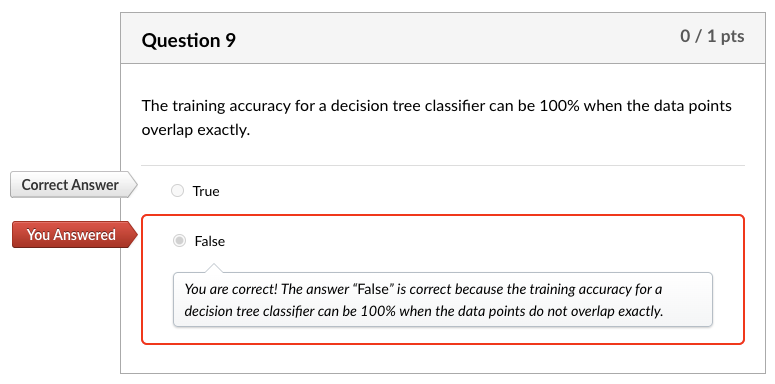
*You are correct! The answer “*False*” is correct because decision trees do not have parameters that act as weights, so the algorithm cannot use L1 or L2 regularization.*

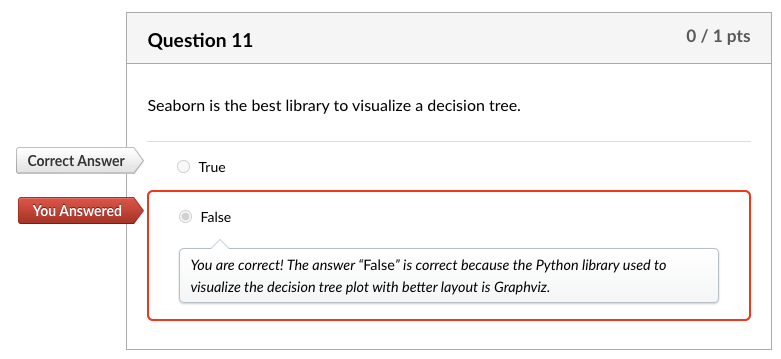
The default option for min\_samples\_split is two. : True

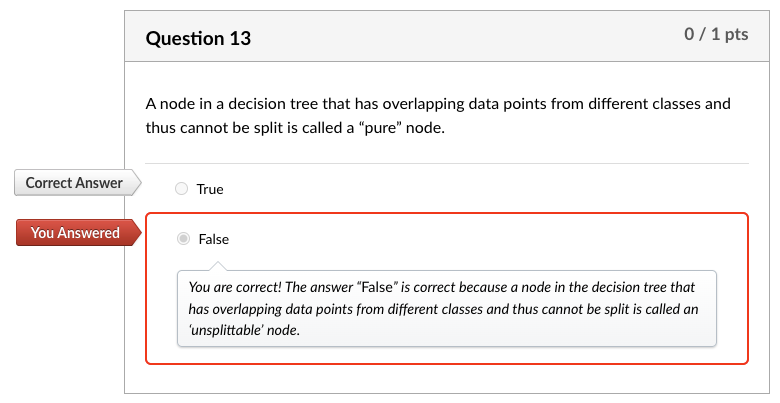
*You are correct! The answer “*True*” is correct because the default option for min\_samples\_split is 2.*

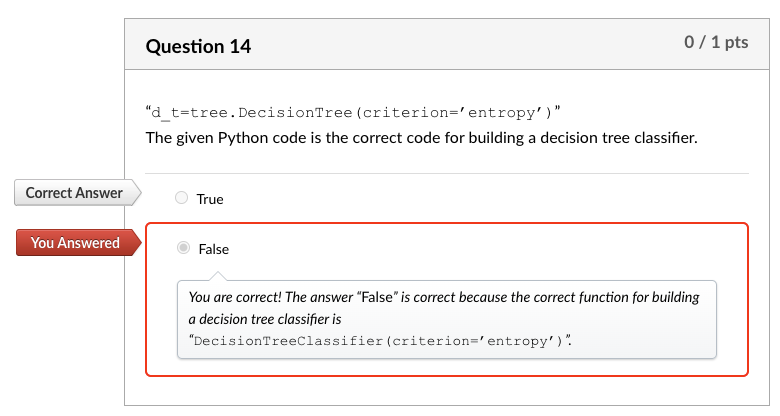
**Quiz Problems:**

Wrong answers set in the template for 4 questions, please see the attachments and correct the answer template as well as my grade. Thanks.









**Discussion 14.1: You Already Use Decision Trees**

Decision Tree for enjoying a (state) park visit

Do you want to visit a close by park?

Yes No

| |

Do you want to see peacocks? Do you want to drive less than 3 hours?

Yes No Yes No

| | | |

Go to Mayfield Park and Nature Preserve Emma Long Metropolitan Park Blue hole Park

How far you want to drive?

Do you want to drive less than 2 hours?

**Yes**

Do you want amenities and restaurants

Yes : Muller Park

Do you want to exercise but still in downtown?

Yes: Lady Bird Lake @ Cesar Chavez

No: McKinney Falls

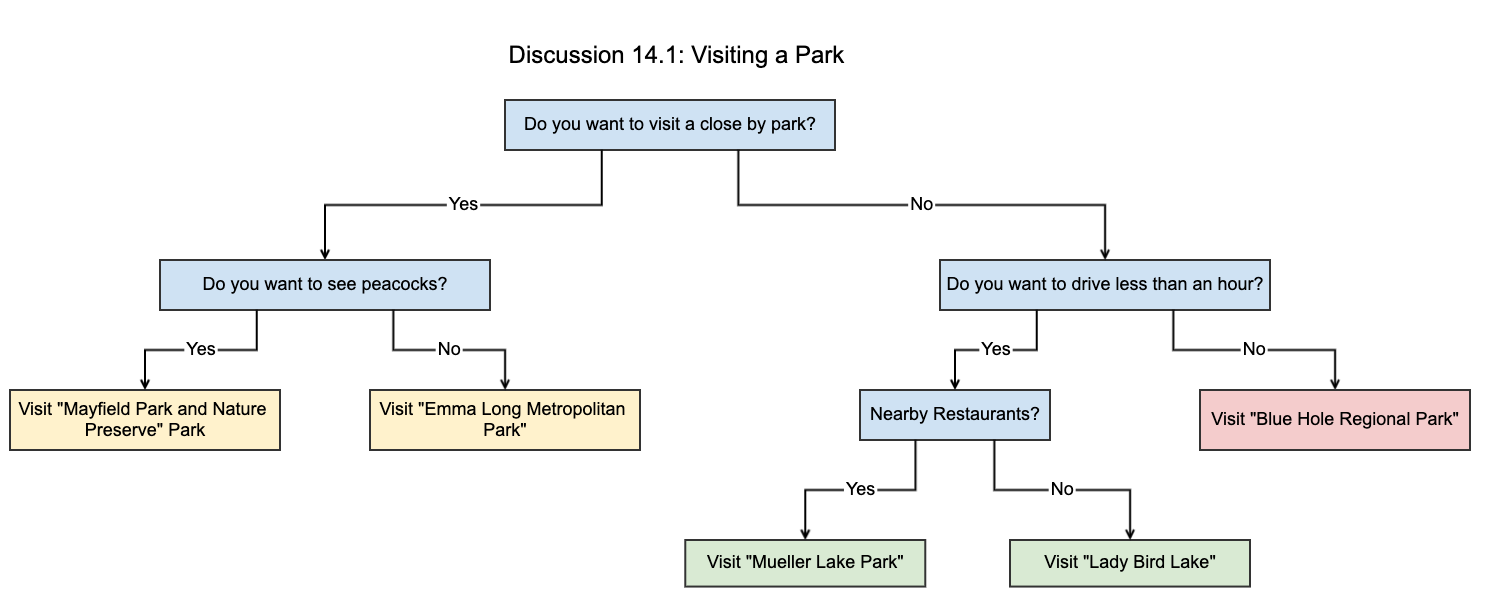
**No:**

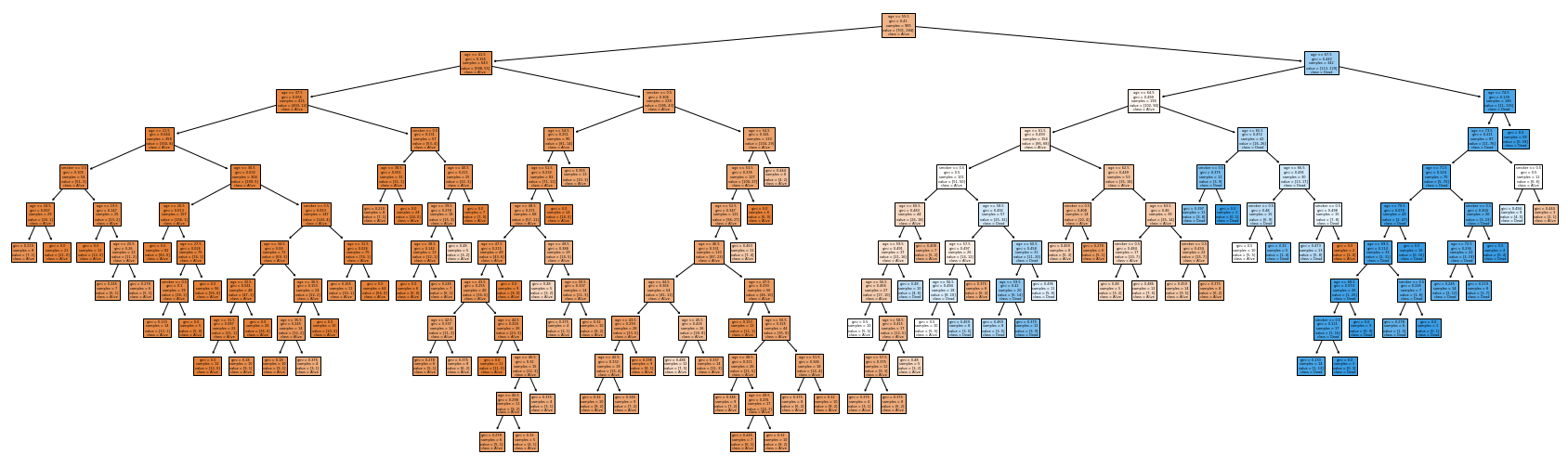
Do you want to stay in close proximity of a downtown?

Yes: Blue Hole Regional Park

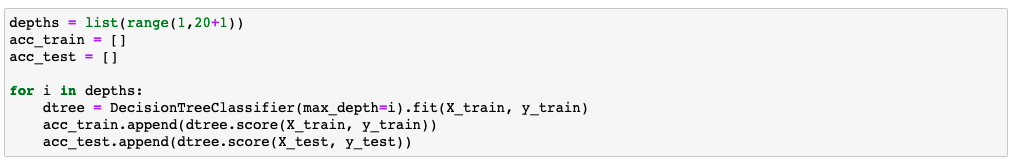
No:

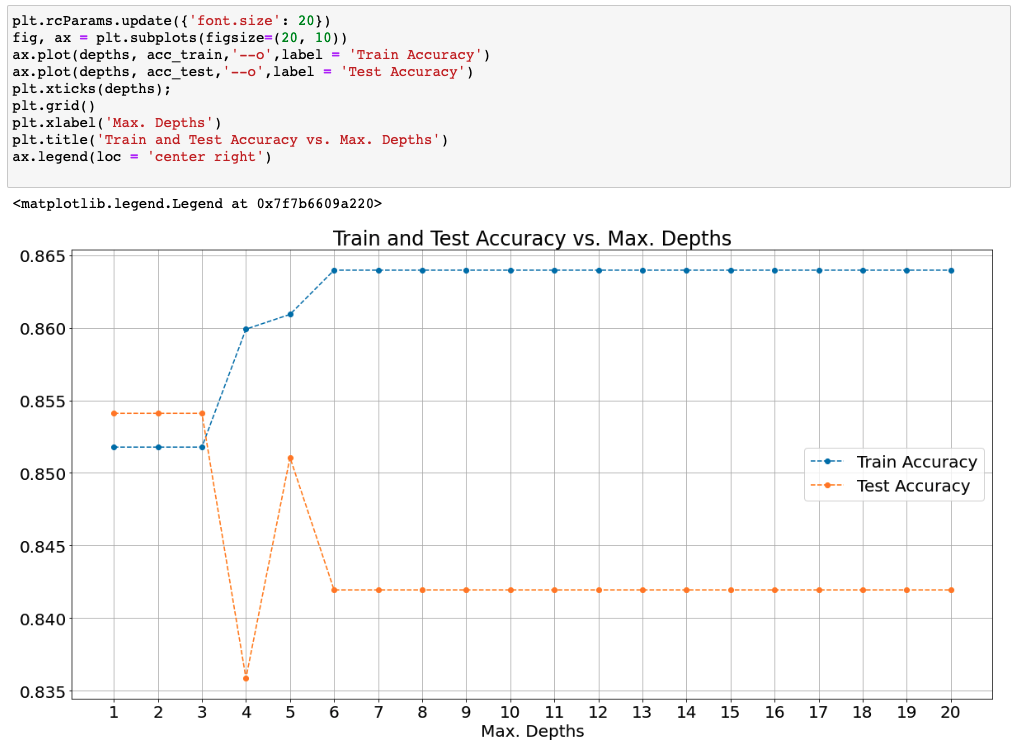
My decision tree is for choosing a park for recreational activities, sometimes I go restaurants after those places, other times just hiking, sometimes we do some country side driving. This is a simple decision tree what I want to do that day, yellow is close by parks, green is moderate drive, red is farther park:











**Try-It Activity 14.1: Generating Overfit Models - Section B**

**Dataset Analysis**

Before iterating through depths until the maximum, I analyzed the dataset. Here is my findings about it:

There are no null values, 1314 entries, 2 string columns.

**Feature Overlapping**

I got curious about feature overlapping for different classes, sure enough there are 72 feature overlapping cases affecting ~11% of the entire population, or model cannot perform for those:

# number of overlapping feature with different classes

data\_size = data.groupby(['smoker', 'age', 'outcome']).size().groupby(['smoker', 'age']).size().reset\_index()

data\_size[data\_size[0]>1]

**Data Preparation**

The rest is transforming smoker column to integer 1/0 binary values, checking if the dataset is balanced, it is 72% and 28% distribution per outcome target column.

y.value\_counts(normalize=True)

**Maximum Depth for Upper Bound**

I ran a default model to get maximum depth for upper bound:

# get maximum depth with default parameters:

max\_tree = DecisionTreeClassifier(random\_state=93).fit(X\_train, y\_train)

max\_depth = max\_tree.get\_depth()

**Iteration for Depth**

Iterating the model over while incrementing the max\_depth parameter to analyze overfitting

# Try out all depths to capture scores

train\_accs = []

test\_accs = []

for d in range(1,max\_depth+1,1):

#create decision tree and fit it

tree = DecisionTreeClassifier(random\_state=93, max\_depth=d).fit(X\_train, y\_train)

# populate train and test accuracy lists

train\_accs.append(tree.score(X\_train, y\_train))

test\_accs.append(tree.score(X\_test, y\_test))

**Accuracy Scores**

# plot scores

plt.subplots(figsize = (10, 7))

plt.step(range(1,max\_depth+1,1), train\_accs, '-o', label = 'Train')

plt.step(range(1,max\_depth+1,1), test\_accs, '-o', label = 'Test')

plt.plot(np.argmax(test\_accs)+1, max(test\_accs), 'ro', markersize = 12, alpha = 0.4, label = 'Best Score')

plt.xticks(range(1,max\_depth+1,1), rotation = 0)

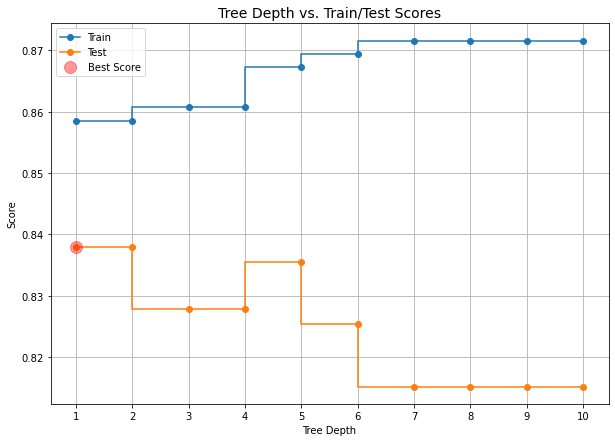
plt.legend()

plt.grid()

plt.xlabel('Tree Depth')

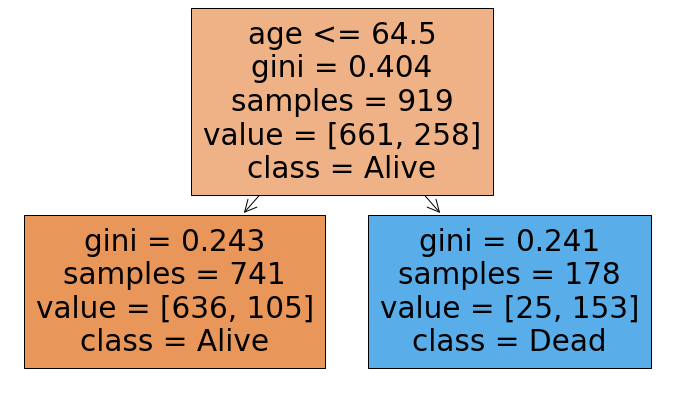
plt.ylabel('Score')

plt.title('Tree Depth vs. Train/Test Scores', fontsize=14)

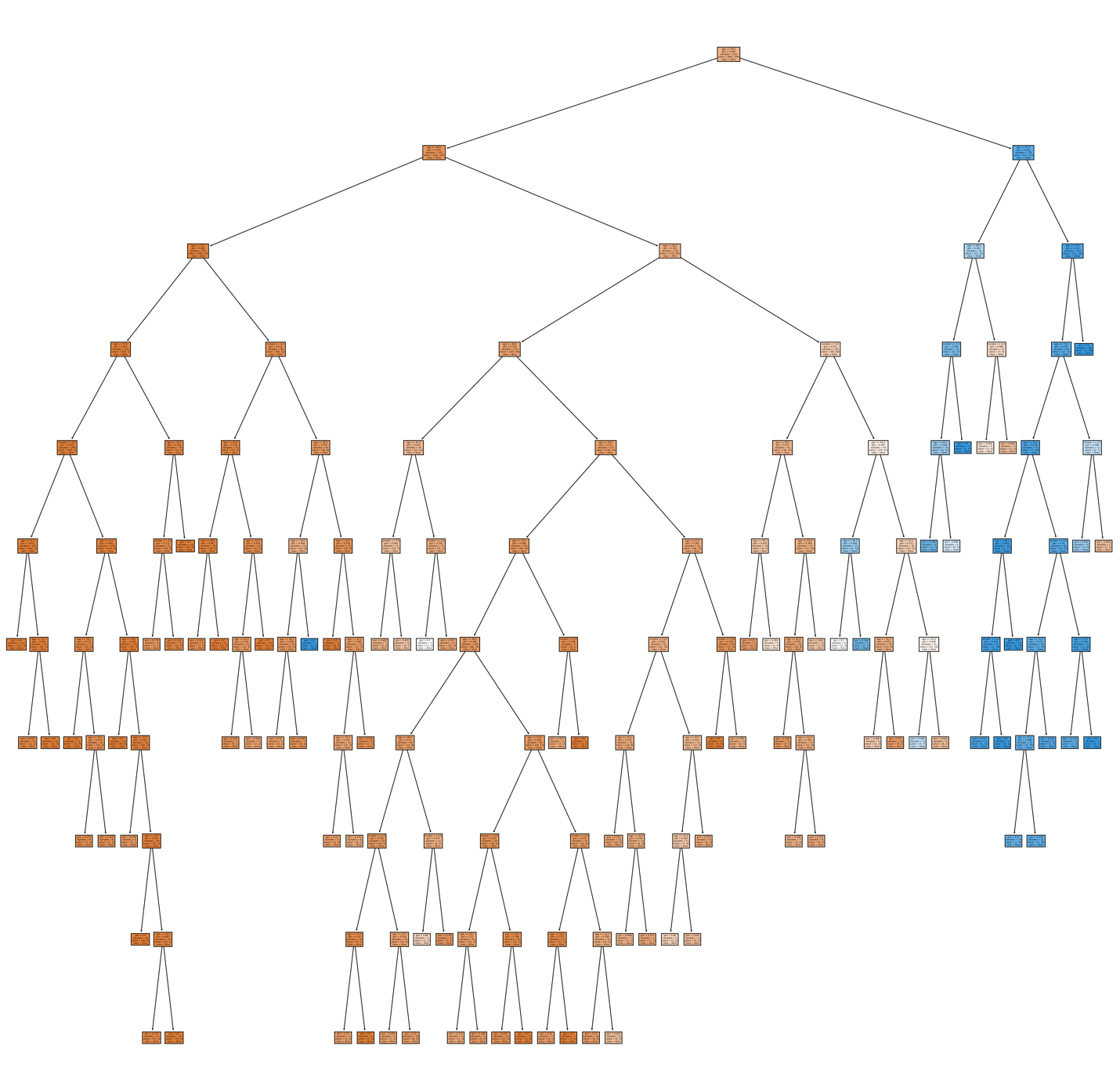


Overfitting begins after max\_depth=1 in this case as test scores decreasing. Please note, depth=1 and 2 scores are the same.

**Optimum Tree:**



**Maximum Depth Tree:**



**Conclusion**

Best accuracy score is an indicator we can spot where overfitting begins on the plot per test accuracy scores. The score degraded when increasing the tree depth, the best score is about 84%, I computed the feature overlapping as 11% which may yield false classification, the best we can get from the model is 89% anyway. In contrast to that, the train accuracy increases with tree-depth increase which is another sign of overfitting.

**Try-It Activity 14.2: Grid Searching Decision Trees - Section B**

I used the same dataset we used in the previous try-t activity. Same analysis, data is imbalanced 72% and 28% distribution per *outcome* target column, 72 feature overlapping, etc.

**Parameters for Grid Search**

# Put some values to try out

params = {'max\_depth': [2, 3, 4, 5, 6, 7, 10],

'min\_samples\_split': [0.1, 0.2, 0.05],

'criterion': ['gini', 'entropy'],

'min\_samples\_leaf': [2, 3, 4, 5]

}

**Model**

# initialize

dtree = DecisionTreeClassifier(random\_state=93)

**Comparing Grid Searches**

I built 4 grid search for:

* GridSearchCV
* RandomizedSearchCV
* HalvingGridSearchCV
* HalvingRandomSearchCV

# GridSearchCV

grid = GridSearchCV(dtree, param\_grid=params).fit(X\_train, y\_train)

grid\_train\_acc = grid.score(X\_train, y\_train)

grid\_test\_acc = grid.score(X\_test, y\_test)

best\_params = grid.best\_params\_

### Results

print(f'Training Accuracy: {grid\_train\_acc: .2f}')

print(f'Test Accuracy : {grid\_test\_acc: .8f}')

print(f'Best parameters of tree: {best\_params}')

Training Accuracy: 0.86

Test Accuracy : 0.83797468

Best parameters of tree: {'criterion': 'gini', 'max\_depth': 2, 'min\_samples\_leaf': 2, 'min\_samples\_split': 0.1}

# RandomizedSearchCV

rgrid = RandomizedSearchCV(dtree, param\_distributions=params).fit(X\_train, y\_train)

rgrid\_train\_acc = rgrid.score(X\_train, y\_train)

rgrid\_test\_acc = rgrid.score(X\_test, y\_test)

rbest\_params = rgrid.best\_params\_

### Results

print(f'Training Accuracy: {rgrid\_train\_acc: .2f}')

print(f'Test Accuracy : {rgrid\_test\_acc: .8f}')

print(f'Best parameters of tree: {rbest\_params}')

Training Accuracy: 0.86

Test Accuracy : 0.83797468

Best parameters of tree: {'min\_samples\_split': 0.2, 'min\_samples\_leaf': 2, 'max\_depth': 7, 'criterion': 'gini'}

# HalvingGridSearchCV

hgrid = HalvingGridSearchCV(dtree, param\_grid=params).fit(X\_train, y\_train)

hgrid\_train\_acc = hgrid.score(X\_train, y\_train)

hgrid\_test\_acc = hgrid.score(X\_test, y\_test)

hbest\_params = hgrid.best\_params\_

### Results

print(f'Training Accuracy: {hgrid\_train\_acc: .2f}')

print(f'Test Accuracy : {hgrid\_test\_acc: .8f}')

print(f'Best parameters of tree: {hbest\_params}')

Training Accuracy: 0.86

Test Accuracy : 0.83797468

Best parameters of tree: {'criterion': 'entropy', 'max\_depth': 3, 'min\_samples\_leaf': 4, 'min\_samples\_split': 0.05}

# HalvingRandomSearchCV

hrgrid = HalvingRandomSearchCV(dtree, param\_distributions=params).fit(X\_train, y\_train)

hrgrid\_train\_acc = hrgrid.score(X\_train, y\_train)

hrgrid\_test\_acc = hrgrid.score(X\_test, y\_test)

hrbest\_params = hrgrid.best\_params\_

### Results

print(f'Training Accuracy: {hrgrid\_train\_acc: .2f}')

print(f'Test Accuracy : {hrgrid\_test\_acc: .8f}')

print(f'Best parameters of tree: {hrbest\_params}')

Training Accuracy: 0.86

Test Accuracy : 0.83797468

Best parameters of tree: {'min\_samples\_split': 0.1, 'min\_samples\_leaf': 3, 'max\_depth': 2, 'criterion': 'gini'}

Their train and test accuracy scores came out the same for all execution results but each search method picked different best parameters.

I used *mean\_fit\_time* in cv\_results\_ to computed cumulated elapsed time for the plots when I need time comparison among all:

# Time spent

# set metrics!

grid\_options=['GridSearchCV','RandomizedSearchCV','HalvingGridSearchCV','HalvingRandomSearchCV']

train\_accs = [grid\_train\_acc, rgrid\_train\_acc, hgrid\_train\_acc, hrgrid\_train\_acc]

test\_accs = [grid\_test\_acc, rgrid\_test\_acc, hgrid\_test\_acc, hrgrid\_test\_acc]

elapsed\_times = [np.sum(grid.cv\_results\_['mean\_fit\_time']), np.sum(rgrid.cv\_results\_['mean\_fit\_time']),

np.sum(hgrid.cv\_results\_['mean\_fit\_time']), np.sum(hrgrid.cv\_results\_['mean\_fit\_time'])]

depths = [best\_params['max\_depth'], rbest\_params['max\_depth'], hbest\_params['max\_depth'], hrbest\_params['max\_depth']]

splits = [best\_params['min\_samples\_split'], rbest\_params['min\_samples\_split'],

hbest\_params['min\_samples\_split'], hrbest\_params['min\_samples\_split']]

leaves = [best\_params['min\_samples\_leaf'], rbest\_params['min\_samples\_leaf'],

hbest\_params['min\_samples\_leaf'], hrbest\_params['min\_samples\_leaf']]

# plot accuracy and time elapsed

fig, ax = plt.subplots(1, 2, figsize = (15, 8))

ax[0].plot(grid\_options, train\_accs, '--o', label = 'Training Accuracy')

ax[0].plot(grid\_options, test\_accs, '--o', label = 'Testing Accuracy')

ax[0].plot(grid\_options, [max(test\_accs)]\*len(grid\_options), 'ro', markersize = 12, alpha = 0.4, label = 'Best Score')

ax[0].tick\_params(axis='x', rotation=90)

ax[0].set\_xlabel('Grid Search Options')

ax[0].set\_ylabel('Accuracy Score')

ax[0].set\_title('Grid Search Options vs. Accuracy Score all is best')

ax[0].legend()

# time plot

ax[1].plot(grid\_options, elapsed\_times, '--o', label = 'Elapsed Time')

ax[1].tick\_params(axis='x', rotation=90)

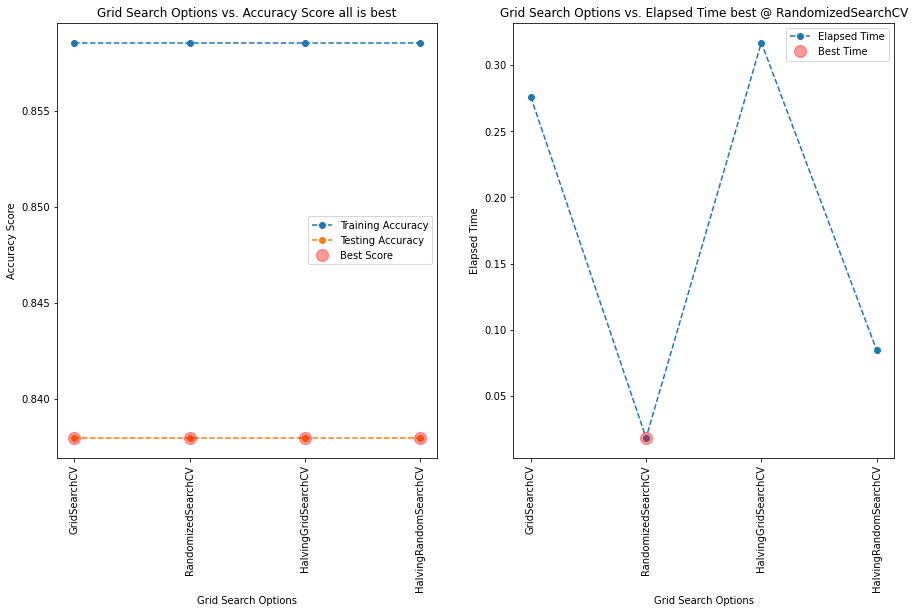
ax[1].set\_xlabel('Grid Search Options')

ax[1].set\_ylabel('Cumulative Elapsed Time')

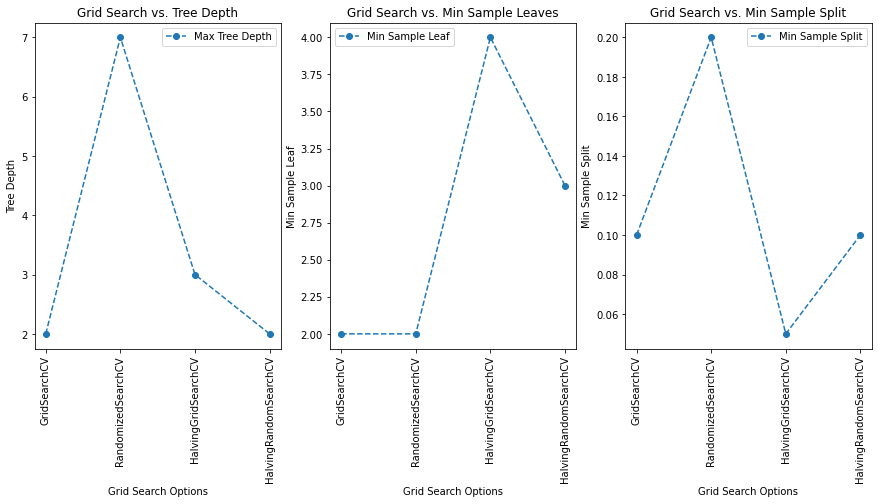
ax[1].set\_title(f'Grid Search Options vs. Elapsed Time best @ {grid\_options[np.argmin(elapsed\_times)]}')

ax[1].plot(np.argmin(elapsed\_times), min(elapsed\_times), 'ro', markersize = 12, alpha = 0.4, label = 'Best Time')

ax[1].legend()



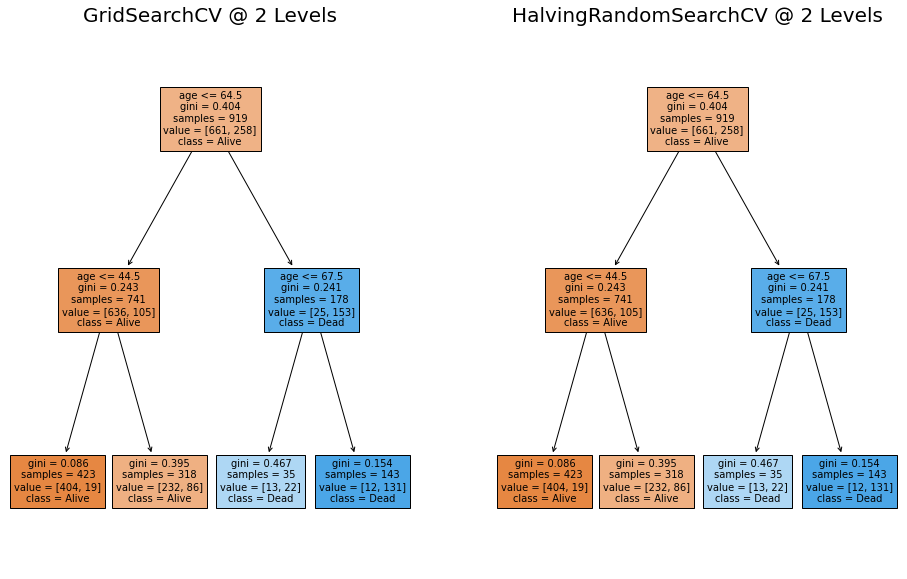
Consensus on all methods for best score, however, RandomizedSearchCV is winner in elapsed time! However, GridSearchCV and HalvingGridSearchCV are significantly higher cumulative times than RandomizedSearchCV and HalvingRandomSearchCV.



Best max\_depth=2, two methods GridSearchCV and HalvingRandomSearchCV achieved it. Best min\_samples\_leaf=2, by GridSearchCV and RandomizedSearchCV. Best min\_samples\_split=0.05 by HalvingGridSearchCV.

**Conclusion**

Best parameters from GridSearchCV and HalvingRandomSearchCV have the minimum max\_depth=2 which reduces complexity.



Also, the other best parameters are the same {'min\_samples\_split': 0.1, 'max\_depth': 2, 'criterion': 'gini'} except 'min\_samples\_leaf': 3 by HalvingRandomSearchCV and 'min\_samples\_leaf': 2 by GridSearchCV.

Both produced the exact same decision tree but HalvingRandomSearchCV is more than 3 times faster than GridSearchCV.

Miguel:

n\_splits  = tree\_grid\_1.n\_splits\_

n\_iter = pd.DataFrame(tree\_grid\_1.cv\_results\_).shape[0]

mean\_time=np.mean(tree\_grid\_1.cv\_results\_['mean\_fit\_time'] + tree\_grid\_1.cv\_results\_['mean\_score\_time'])

print(f'Elapsed Time using RandomizedSearchCV: {mean\_time \* n\_splits \* n\_iter: .3f}')

 ————— o —————