REPORT ON

Decision Tree on Dataset_E

1. Information on the Dataset :

- Dataset consist of 8 medical predictor variables and one target variable that is **outcome**.

- All the variables are of type int ,except two BMI and DiabetesPedigreeFunction which are of type float.

- No null values are present ,so no need to deal with it .

```
In [81]: #PART1


         df = pd.read_csv("Dataset_E.csv")                    #reading first csv file
         df.info()                                               #getting info of first csv file
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

2. <u>Splitting the dataset</u>

Splitting the dataset in 80-20 as training and test data.

```
In [341]:
Target_set = np.array(df.iloc[:,-1])
Train_set = np.array(df.drop([df.columns[-1]],axis=1))
```

#Spliting the Data into training and testing set

```
In [349]: x_train, x_test, y_train, y_test = train_test_split(Train_set, Target_set, test_size = 0.2,random_state=0)
x_train = np.array(x_train)
y_train = np.array(y_train)
x_test = np.array(x_test)
y_test = np.array(y_test)
x_train
```

<u>3.Building Decision Tree Classifier</u>

- <u>Handling continuous valued attribute:-</u>

The training attributes are continuous valued so, we need to find the candidate thresholds for an attribute and repeat this for each attribute.

Procedure:

for each attribute in attribute list:

map the attribute and class in a candidate list
sort the candidate list with respect to 0*th* index
iterate from 0 to training example size and whenever class changes
append the mean of both attribute value between the change
and append to the list ‘candidate threshold’

We will use this procedure which is provided in Machine Learning by Tom Mitchell.
At last ,we will choose the best one to split.

- <u>Performance Metric: Information gain:-</u>

We are told to use Information Gain as performance measure in the question.

Information Gain use entropy of dataset and attribute to classify training examples according to their target classification

The entropy(S) relative to a Boolean classification is given by:

$$Entropy(s) = \sum_{c \in classes} -P_c * log_2 P_c$$

$$\textbf{Gain(S,A)} = Entropy(S) - \sum_{v \in values(A)} \frac{|S_v|}{|S|} * Entropy(S_v)$$

- <u>Decision Tree Building Procedure:-</u>

The following procedures explains the step-by-step algorithm to split the node and build the tree recursively.

```
BuildTree(Node):
    If the Node is pure, then
        create a leaf node with class of the dataset currently now present at the node
        return leaf node
    Find the candidate thresholds
    Best attribute and splitting threshold metric(candidate thresholds)
    Split the Tree based on the Best attribute and threshold
    root.left = BuildTree(left node)
    root.right = BuildTree(right node)
```

i. After initializing the model, we call the defined function 'model.fit()' which will build the tree.
ii. To find the accuracy of the model on the testing data, 'model.score()' should be called.
iii. We have randomly split the Dataset into 80-20 ratio of Training and Testing Dataset and run the model.

```
In [798]: #training the tree classifier we built with training data and seeing performance

model = Decision_tree(x_train, y_train, x_test, y_test,criteria = 'Information Gain')
model.fit()
print('Accuracy: ' + str(model.score()) + ', nodes in the tree: ' + str(model.total_nodes()))
file1 = open("DecisionTree_on_Training_Data.txt","w")
L = ['Accuracy: ' + str(model.score()) + ', nodes in the tree: ' + str(model.total_nodes())]
file1.writelines(L)
file1.close()
```

```
Accuracy: 0.6558441558441559, nodes in the tree: 205
```

ii) **We will run the tree for 10 random splits and the result is as shown:-**

```
In [801]: for i in range(len(score_list)):
  print('Depth of the tree: ' + str(depth_of_the_tree1[i]) + ', Number of nodes: ' +str(no_of_nodes1[i]) + '
file2 = open("DecisionTree_on_random10splits.txt","w")
for i in range(len(score_list)):
    file2.write('Depth of the tree: ' + str(depth_of_the_tree1[i]) + ', Number of nodes: ' +str(no_of_nodes1
    file2.write('\n')
file2.close()
```

```
Depth of the tree: 18, Number of nodes: 195, Accuracy: 0.7337662337662337
Depth of the tree: 15, Number of nodes: 193, Accuracy: 0.6103896103896104
Depth of the tree: 16, Number of nodes: 211, Accuracy: 0.7532467532467533
Depth of the tree: 15, Number of nodes: 193, Accuracy: 0.7272727272727273
Depth of the tree: 16, Number of nodes: 207, Accuracy: 0.6623376623376623
Depth of the tree: 15, Number of nodes: 201, Accuracy: 0.7402597402597403
Depth of the tree: 13, Number of nodes: 185, Accuracy: 0.6753246753246753
Depth of the tree: 14, Number of nodes: 183, Accuracy: 0.6948051948051948
Depth of the tree: 15, Number of nodes: 207, Accuracy: 0.7467532467532467
Depth of the tree: 15, Number of nodes: 187, Accuracy: 0.6493506493506493
```

For 10 split we get depth and accuracy and number of nodes

Out of 10 we will print best accuracy and depth :-

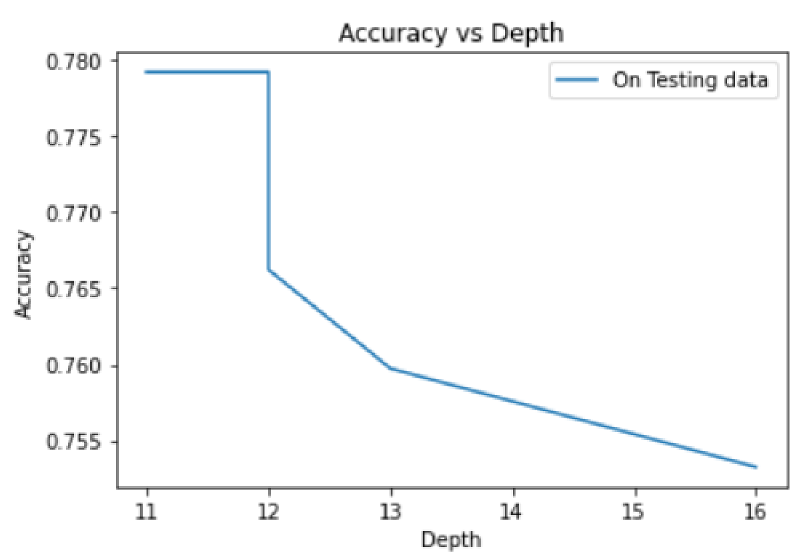
```
In [802]: print('Depth of the tree: ' + str(best_depth) +  ', Accuracy: ' + str(best_score))
          file1 = open("Best_Tree_Depth_and_Accuracy.txt","w")
          L = ['Depth of the tree: ' + str(best_depth) +  ', Accuracy: ' + str(best_score)]
          file1.writelines(L)
          file1.close()

          Depth of the tree: 16, Accuracy: 0.7532467532467533
```

3. Now for third part , we will use this model as the reference and do reduce error pruning and then plot the graph

We will use Reduce error pruning technique developed by Quinlan [1987b].

It states:

The difference between the numbers of errors (if positive) is a measure of the gain from pruning the sub-tree.

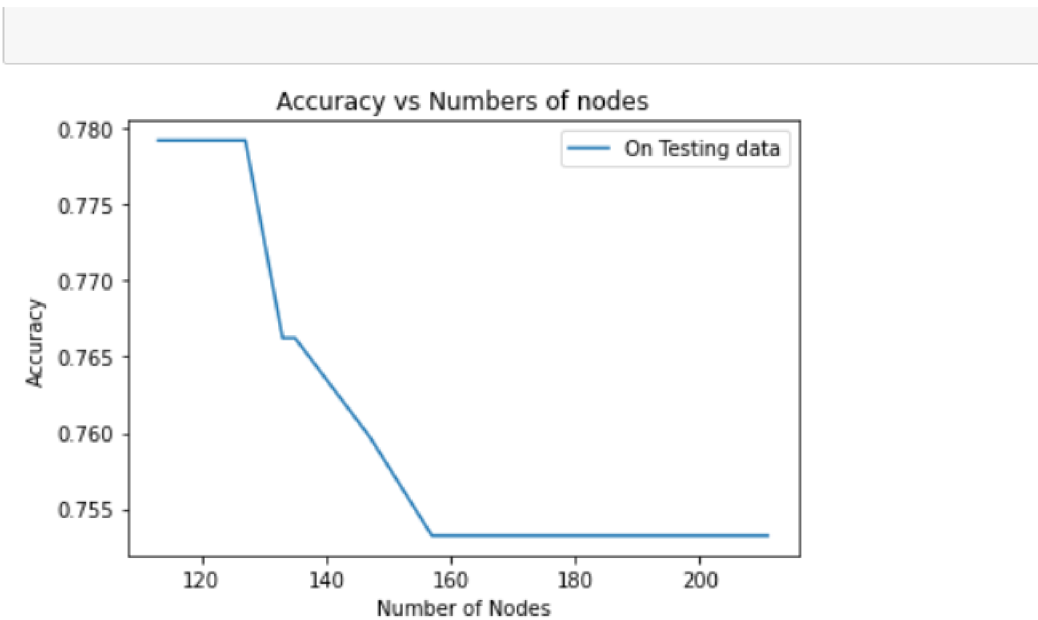


As you can see that best model has depth 16 and accuracy of 0.753.

But after reduced error pruning , as we decrease the depth from 16 to 11 ,accuracy increases from 0.753 to 0.780

And further if we will use pruning ,accuracy will decrease.

So , this is best we can get after reduced error pruning.



This is number of nodes vs Accuracy graph. Number of nodes will get decrease from around 200 to 120 in pruning . And the accuracy will also increase from 0.73 to 0.78

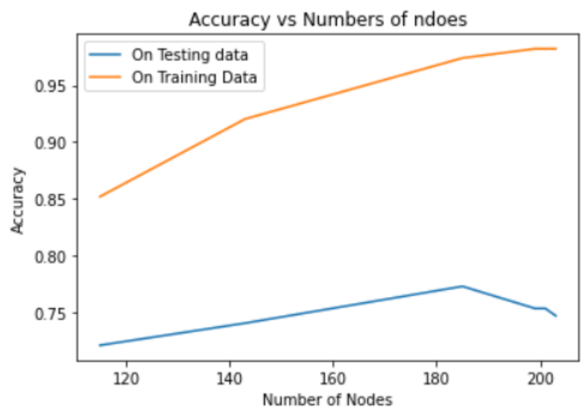
Structure of Tree After Pruning:-

In [455]:
```
best_model.print_tree()
```

```
  ||  Attribute_index[1] <= 131.0  ||   ||
  ||  Attribute_index[5] <= 26.299999999999997  ||    ||  Attribute_index[5] <= 29.9  ||    ||
  ||  Class: 1      ||  Attribute_index[7] <= 30.5  ||    ||  Attribute_index[1] <= 146.0  ||    ||  Attribute_index[1] <= 157.
5  ||   ||
  ||  Attribute_index[6] <= 0.501  ||    ||  Attribute_index[1] <= 107.0  ||    ||  Class: 1      ||  Attribute_index[7] <= 27.
5  ||    ||  Attribute_index[2] <= 62.0  ||    ||  Attribute_index[4] <= 629.5  ||   ||
  ||  Attribute_index[5] <= 45.35  ||    ||  Attribute_index[5] <= 32.2  ||    ||  Attribute_index[5] <= 38.65  ||    ||  Attri
bute_index[6] <= 0.85  ||    ||  Class: 1      ||  Attribute_index[7] <= 61.0  ||    ||  Class: 2      ||  Attribute_index[7] <
= 42.0  ||    ||  Class: 2      ||  Attribute_index[2] <= 65.0  ||   ||
  ||  Class: 1      ||  Class: 2      ||  Class: 1      ||  Attribute_index[5] <= 43.2  ||    ||  Attribute_index[6] <= 0.6375
||    ||  Attribute_index[3] <= 36.5  ||    ||  Attribute_index[7] <= 53.0  ||    ||  Class: 2      ||  Attribute_index[2] <= 7
4.0  ||    ||  Class: 1      ||  Attribute_index[5] <= 41.65  ||    ||  Attribute_index[6] <= 0.226  ||    ||  Class: 2      ||
Class: 1      ||
  ||  Attribute_index[0] <= 5.5  ||    ||  Class: 1      ||  Class: 1      ||  Attribute_index[0] <= 8.5  ||    ||  Class: 2
||  Attribute_index[0] <= 5.5  ||    ||  Attribute_index[5] <= 39.65  ||    ||  Class: 1      ||  Class: 2      ||  Attribute_i
ndex[5] <= 27.25  ||    ||  Attribute_index[5] <= 38.6  ||    ||  Attribute_index[3] <= 50.0  ||    ||  Attribute_index[4] <= 1
05.0  ||    ||  Attribute_index[6] <= 0.705  ||    ||  Attribute_index[1] <= 79.0  ||    ||  Class: 1      ||  Class: 1      ||
Class: 2      ||  Class: 1      ||  Attribute_index[5] <= 45.900000000000006  ||    ||  Attribute_index[6] <= 0.698  ||    ||
Class: 2      ||  Attribute_index[1] <= 151.0  ||    ||  Attribute_index[1] <= 149.0  ||    ||  Attribute_index[4] <= 187.5  ||
||  Class: 1      ||  Class: 2      ||  Class: 1      ||  Class: 1      ||  Class: 2      ||  Class: 2      ||  Attribute_index
[0] <= 8.0  ||    ||  Class: 1      ||  Attribute_index[5] <= 38.650000000000006  ||    ||  Class: 2      ||  Class: 1      ||
Attribute_index[1] <= 121.5  ||    ||  Class: 1      ||  Class: 1      ||  Class: 2      ||  Class: 2      ||  Class: 1      ||
Attribute_index[5] <= 33.7  ||    ||  Class: 1      ||  Class: 1      ||  Class: 2      ||  Attribute_index[1] <= 109.5  ||
||  Class: 1      ||  Attribute_index[5] <= 35.35  ||    ||  Attribute_index[7] <= 34.5  ||    ||  Attribute_index[6] <= 0.5885
||    ||  Class: 2      ||  Class: 2      ||  Attribute_index[5] <= 35.45  ||    ||  Class: 2      ||  Attribute_index[2] <= 6
9.0  ||    ||  Class: 1      ||  Attribute_index[2] <= 76.0  ||    ||  Attribute_index[7] <= 38.0  ||    ||  Class: 2      ||
Attribute_index[1] <= 112.5  ||    ||  Class: 1      ||  Class: 2      ||  Class: 1      ||  Attribute_index[0] <= 8.0  ||    |
|  Attribute_index[4] <= 361.0  ||    ||  Class: 1      ||  Class: 2      ||  Class: 1      ||  Class: 2      ||  Class: 2
||  Attribute_index[1] <= 124.0  ||    ||  Class: 1      ||  Class: 2      ||  Class: 1      ||  Class: 2
```

CONCLUSION:-

```
Accuracy on Training Data: 0.9820846905537459 On Testing Data: 0.7467532467532467 Nodes: 203
Accuracy on Training Data: 0.9820846905537459 On Testing Data: 0.7532467532467533 Nodes: 201
Accuracy on Training Data: 0.9820846905537459 On Testing Data: 0.7532467532467533 Nodes: 199
Accuracy on Training Data: 0.9739413680781759 On Testing Data: 0.7727272727272727 Nodes: 185
Accuracy on Training Data: 0.9201954397394136 On Testing Data: 0.7402597402597403 Nodes: 143
Accuracy on Training Data: 0.8517915309446255 On Testing Data: 0.7207792207792207 Nodes: 115
```



We can conclude from the graph that , accuracy for testing data will increase during pruning when we will reduce the number of nodes for the decision tree till around 185.

After that ,accuracy will decrease if we will do further pruning