Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Machine Learning – I (DJ19DSC402)

AY: 2021-22

Experiment 3 (Decision Tree)

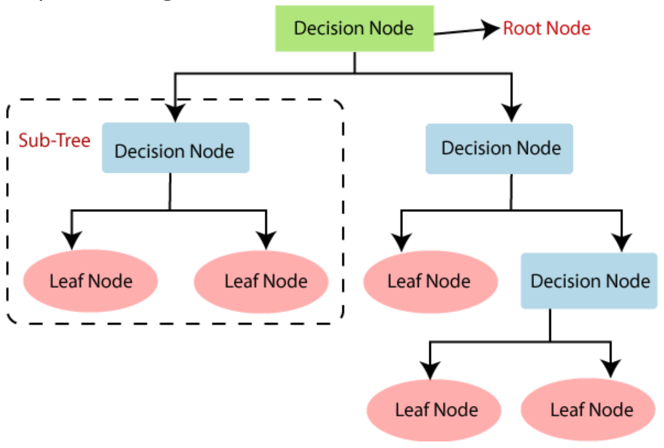
Name: Dev Patel **SAP ID: 60009200016**

Batch: K/K1 **Date: 04/04/22**

Aim: Implement Decision Tree on the given Dataset to build a classifier and Regressor.

Theory:

Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.** In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node.**
Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
The decisions or the test are performed on the basis of features of the given dataset.
It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions. It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees. Below diagram explains the general structure of a decision tree:

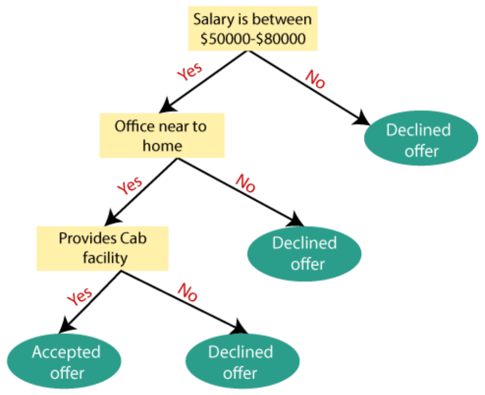


Decision Tree Terminologies

Root Node: Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.
Leaf Node: Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
Splitting: Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
Branch/Sub Tree: A tree formed by splitting the tree.
Pruning: Pruning is the process of removing the unwanted branches from the tree.
Parent/Child node: The root node of the tree is called the parent node, and other nodes are called the child nodes.

Steps in building a Tree
Step-1: Begin the tree with the root node, says S, which contains the complete dataset.
Step-2: Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**
Step-3: Divide the S into subsets that contains possible values for the best attributes.
Step-4: Generate the decision tree node, which contains the best attribute.
Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3.
Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Example: Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:



Attribute Selection Measures
While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute**

selection measure or ASM. By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

1. Information Gain:

Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute. It calculates how much information a feature provides us about a class.
According to the value of information gain, we split the node and build the decision tree.
A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:
Information Gain= Entropy(S)- [(Weighted Avg.) *Entropy(each feature)

Entropy: Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

$$\text{Entropy(s)}= -P(\text{yes})\log2\ P(\text{yes})- P(\text{no})\ \log2\ P(\text{no})$$

Where,
S= Total number of samples
P(yes)= probability of yes
P(no)= probability of no

2. Gini Index:

Gini index is a measure of impurity or purity used while creating a decision tree in the CART (Classification and Regression Tree) algorithm.
An attribute with the low Gini index should be preferred as compared to the high Gini index.
It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
Gini index can be calculated using the below formula:

$$\text{Gini Index}= 1- \sum_j P_j^2$$

Pruning: Getting an Optimal Decision tree

Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree. A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree **pruning** technology used: • Cost Complexity Pruning

- Reduced Error Pruning.

Lab Assignments to complete in this session:

Use the given dataset and perform the following tasks:

Dataset 1: PlayTennis.csv
Dataset 2: Iris.csv
Dataset 3: Diabetes.csv
Dataset 4: car prediction.csv

Department of Computer Science and Engineering (Data Science)

1. Implement Decision tree classifier from scratch on Dataset 1 by defining Node class and Tree class.
2. Use python libraries to build a decision tree classifier on Dataset 2. Analyse the results with different methods.
3. Discuss about overfitting on Dataset 3 using python libraries.
4. Implement Decision tree regressor from scratch on Dataset 4.

Code with output: (Q3, Q4)

In Q3, we observe that the values of accuracy are high in train set and low in test set, i.e., model is overfitting. We observe the effect of three attributes on the model and how it affects its performance. We try to obtain a good model with close values of accuracy on both train and test sets.

3. Discuss about overfitting on Dataset 3 using python libraries.

In [63]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier,DecisionTreeRegressor
from sklearn.metrics import accuracy_score , confusion_matrix
```

In [64]:

```
diabetes = pd.read_csv("/content/sample_data/diabetes.csv")
diabetes.head()
```

Out[64]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [65]:

```
feature_cols = ['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','
DiabetesPedigreeFunction','Age']
X = diabetes[feature_cols]
y = diabetes['Outcome']
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size =0.20,random_state =1
)
```

In [66]:

```
DT = DecisionTreeClassifier()
DT.fit(X_train , y_train)
y_pred = DT.predict(X_test)
y_trainpred = DT.predict(X_train)
```

In [67]:

```
print("Accuracy in train_set= ",accuracy_score(y_train,y_trainpred))
print("Accuracy in test_set= ",accuracy_score(y_test,y_pred))
```

```
Accuracy in train_set=  1.0
Accuracy in test_set=  0.7142857142857143
```

As the accuracy in train test is perfect, but is low in the test test, the model is overfitting.

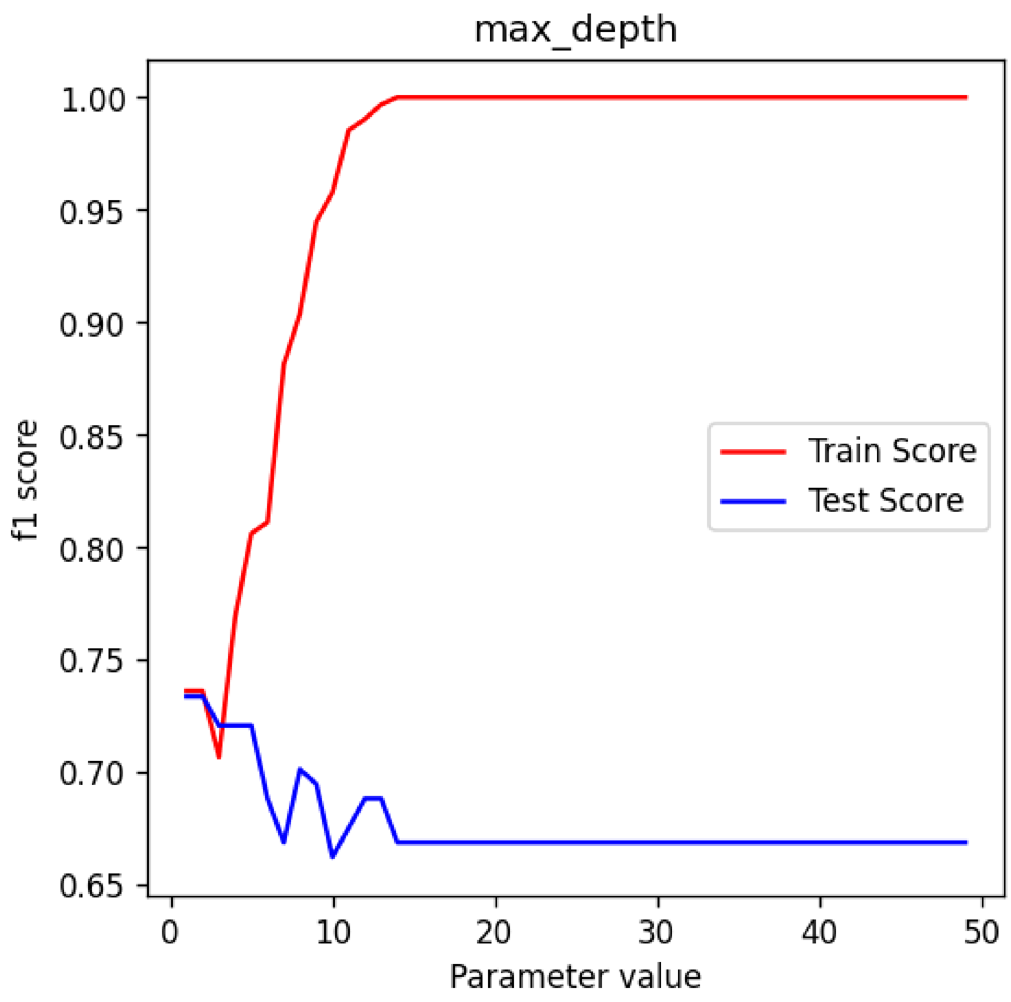
In [68]:

```
from sklearn.metrics import accuracy_score
def calc_score(model,x1,y1,x2,y2):
  model.fit(x1,y1)
  predict = model.predict(x1)
  f1 = accuracy_score(y1,predict)
  predict = model.predict(x2)
  f2 = accuracy_score(y2,predict)
  return f1,f2
```

```
def effect(train_score,test_score,x_axis, title):
  plt.figure(figsize =(5,5),dpi =120)
  plt.plot(x_axis, train_score, color='red',label = 'Train Score')
  plt.plot(x_axis, test_score, color='blue',label = 'Test Score')
  plt.title(title)
  plt.legend()
  plt.xlabel("Parameter value")
  plt.ylabel("f1 score")
  plt.show()
```

In [69]:

```
maxdepth = [i for i in range(1,50)]
train = []
test =[]

for i in maxdepth:
  model = DecisionTreeClassifier(class_weight = 'balanced', max_depth = i, random_state
= 42)
  f1,f2 = calc_score(model,X_train,y_train,X_test,y_test)
  train.append(f1)
  test.append(f2)
effect(train,test,range(1,50),'max_depth')
```

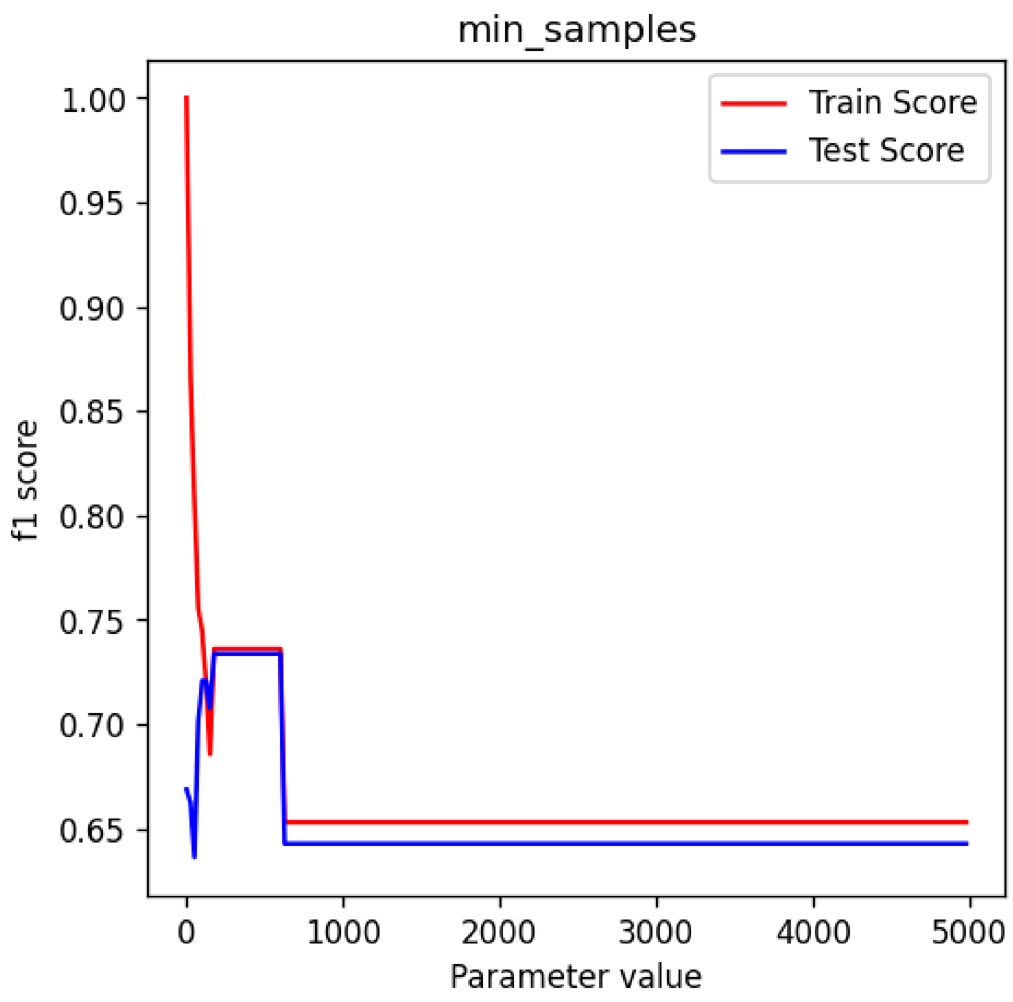


The maximum depth of the tree affects the performance of the model. It starts off well and becomes overfitting with increase in maximum depth.

In [70]:

```
min_samples = [i for i in range(2,5000,25)]
train = []
test =[]

for i in min_samples:
  model = DecisionTreeClassifier(class_weight = 'balanced', min_samples_split = i, rando
m_state = 42)
```

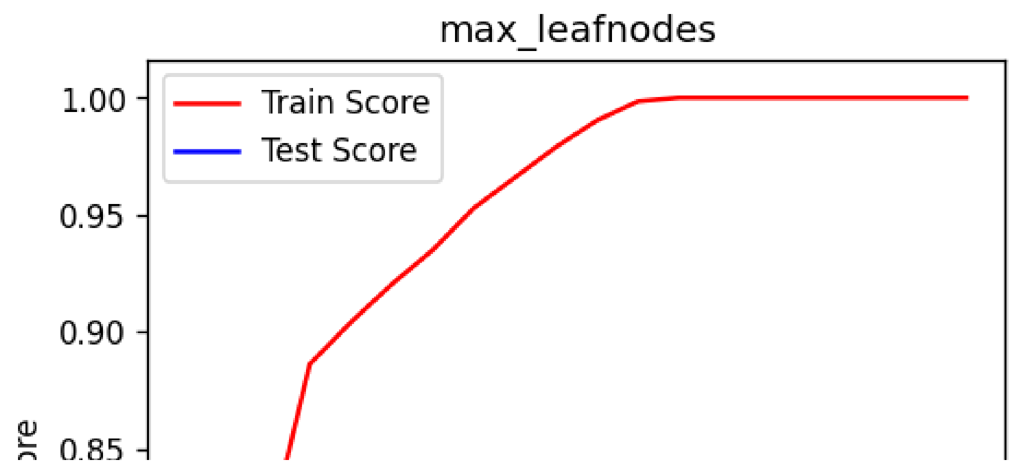
```
  f1,f2 = calc_score(model,X_train,y_train,X_test,y_test)
  train.append(f1)
  test.append(f2)
effect(train,test,range(2,5000,25),'min_samples')
```

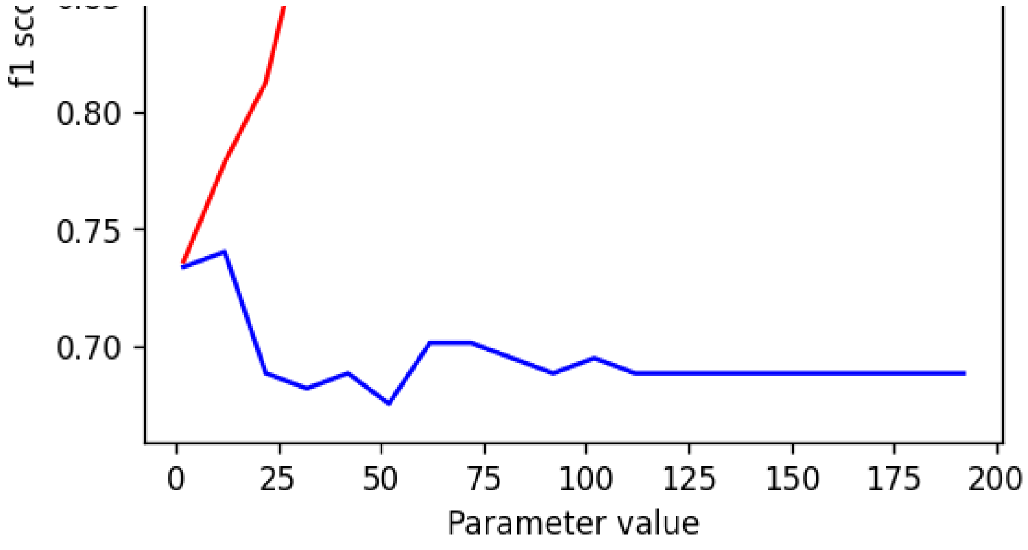


The minimum sample split checks if the samples after split is higher than the specified number. Low min_sample_split leads to an overfitting model and a higher value leads to an underfitting model.

In [71]:

```
max_leafnodes = [i for i in range(2,200,10)]
train = []
test =[]

for i in max_leafnodes:
  model = DecisionTreeClassifier(class_weight = 'balanced', max_leaf_nodes = i, random_s
tate = 42)
  f1,f2 = calc_score(model,X_train,y_train,X_test,y_test)
  train.append(f1)
  test.append(f2)
effect(train,test,range(2,200,10),'max_leafnodes')
```





High maximum number of leaf nodes leads to an overfitting model.

In [72]:

```
DT = DecisionTreeClassifier(class_weight = 'balanced', max_leaf_nodes=12, random_state =
42)
DT.fit(X_train , y_train)
y_pred = DT.predict(X_test)
y_trainpred = DT.predict(X_train)
```

In [73]:

```
print("Accuracy in train_set= ",accuracy_score(y_train,y_trainpred))
print("Accuracy in test_set= ",accuracy_score(y_test,y_pred))
```

```
Accuracy in train_set=  0.7785016286644951
Accuracy in test_set=  0.7402597402597403
```

We can treat overfitting by hypertuning, i.e., changing values of these attributes.

In [74]:

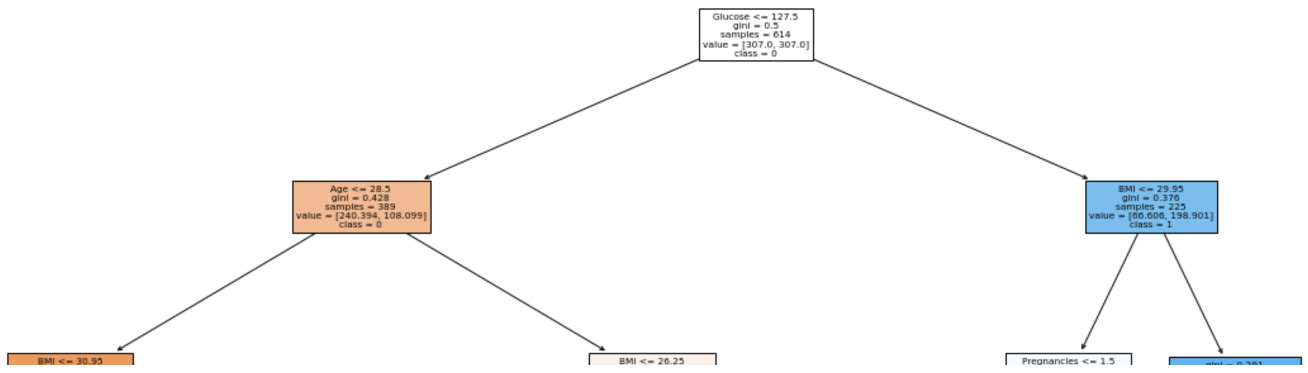
```
confusion_matrix(y_test,y_pred)
```

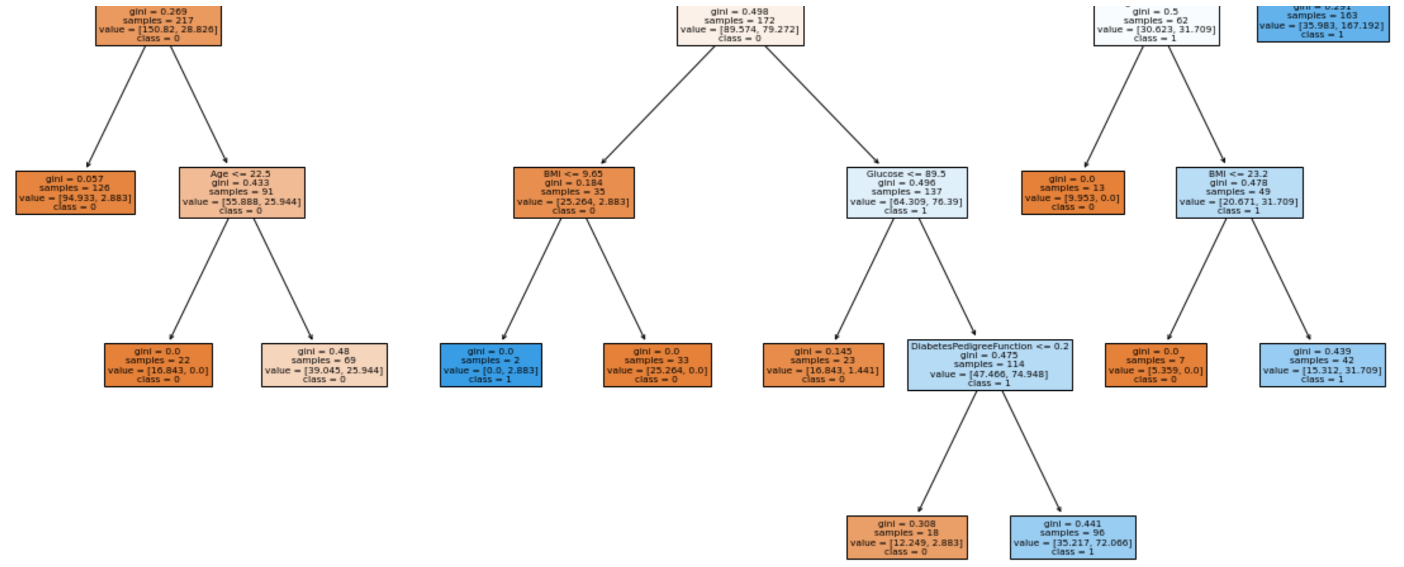
Out[74]:

```
array([[70, 29],
       [11, 44]])
```

In [75]:

```
from sklearn.tree import plot_tree
fig = plt.figure(figsize=(20,15))
_ = plot_tree(DT,
              feature_names=list(X_train),
              class_names = ["0","1"],
              filled=True)
```





4. Implement Decision tree regressor on Dataset 4.

In [76]:

```
carpred = pd.read_csv("/content/sample_data/carprediction.csv")
carpred.head()
```

Out[76]:

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	Market Category	Vehicle Size	
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	2.0	Factory Tuner,Luxury,High-Performance	Compact	
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	C
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,High-Performance	Compact	
3	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	
4	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury	Compact	C

In [77]:

```
carpred.shape
```

Out[77]:

```
(11914, 16)
```

In [78]:

```
carpred.isnull().sum()
```

Out[78]:

```
Make                   0
Model                  0
Year                   0
Engine Fuel Type       3
Engine Fuel Type           3
Engine HP                 69
Engine Cylinders          30
Transmission Type          0
Driven_Wheels              0
Number of Doors            6
Market Category         3742
Vehicle Size               0
Vehicle Style              0
highway MPG                0
city mpg                   0
Popularity                 0
MSRP                       0
dtype: int64
```

In [79]:

```
carpred.drop(columns="Market Category",inplace = True)
```

In [80]:

```
carpred.dropna(inplace = True)
```

In [81]:

```
# Import label encoder
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
carpred['Make']= label_encoder.fit_transform(carpred['Make'])
carpred['Model']= label_encoder.fit_transform(carpred['Model'])
carpred['Engine Fuel Type']= label_encoder.fit_transform(carpred['Engine Fuel Type'])
carpred['Transmission Type']= label_encoder.fit_transform(carpred['Transmission Type'])
carpred['Driven_Wheels']= label_encoder.fit_transform(carpred['Driven_Wheels'])
carpred['Vehicle Size']= label_encoder.fit_transform(carpred['Vehicle Size'])
carpred['Vehicle Style']= label_encoder.fit_transform(carpred['Vehicle Style'])
carpred.head()
```

Out[81]:

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	Vehicle Size	Vehicle Style	highway MPG	city mpg	Po
0	4	1	2011	7	335.0	6.0	3	3	2.0	0	8	26	19	
1	4	0	2011	7	300.0	6.0	3	3	2.0	0	6	28	19	
2	4	0	2011	7	300.0	6.0	3	3	2.0	0	8	28	20	
3	4	0	2011	7	230.0	6.0	3	3	2.0	0	8	28	18	
4	4	0	2011	7	230.0	6.0	3	3	2.0	0	6	28	18	

In [82]:

```
featurecols = carpred.columns[:-1]
X = carpred[featurecols]
y = carpred["MSRP"]
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size =0.20,random_state =1
)
```

In [83]:

```
DTR = DecisionTreeRegressor()
DTR.fit(X_train , y_train)
y_pred = DTR.predict(X_test)
y_trainpred = DTR.predict(X_train)
```

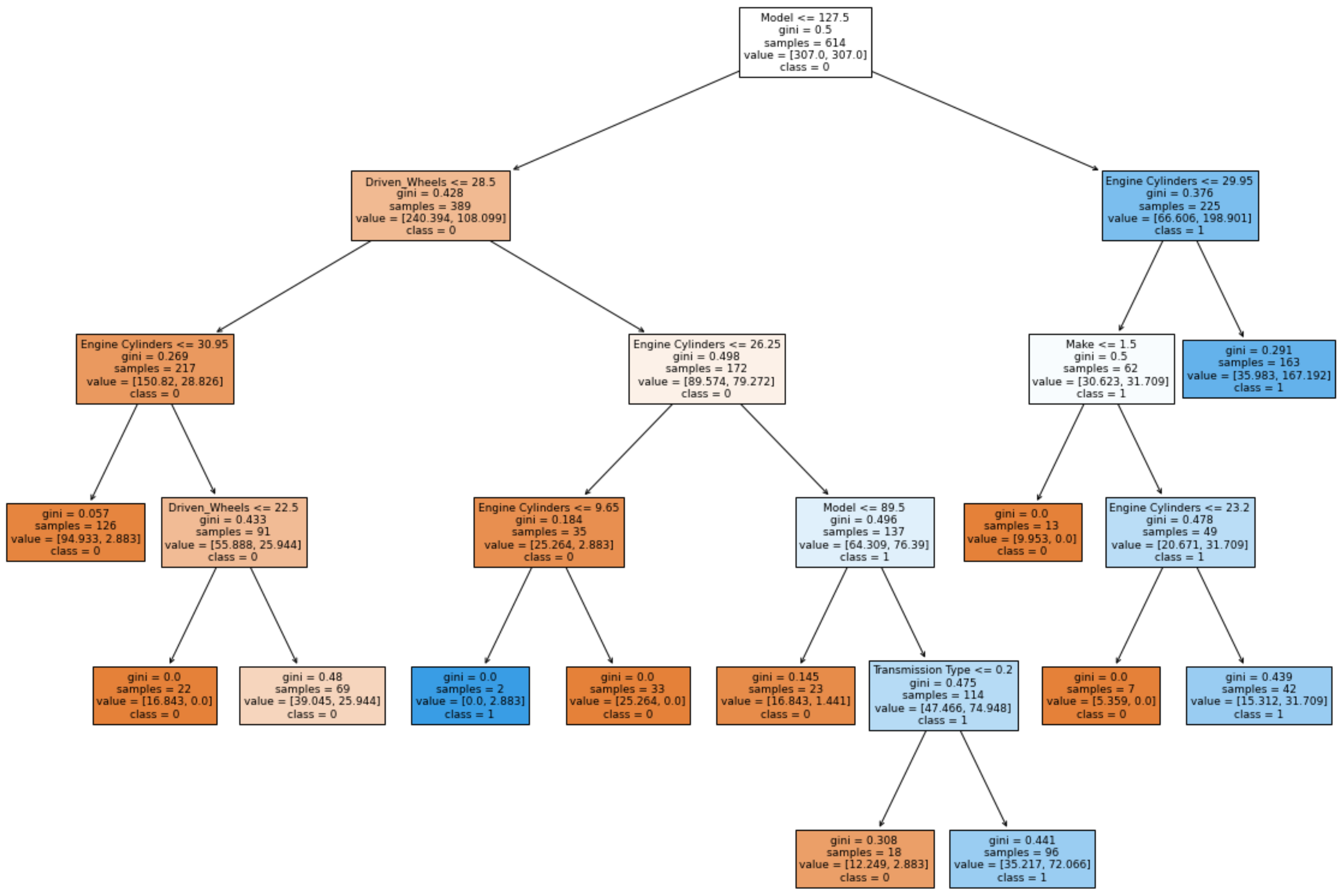
In [84]:

```python
def reg_metrics(y_test, y_pred, X_train):
    from sklearn.metrics import mean_squared_error, r2_score

    rmse = np.sqrt(mean_squared_error(y_test,y_pred))
    r2 = r2_score(y_test,y_pred)

    # Scikit-learn doesn't have adjusted r-square, hence custom code
    n = y_pred.shape[0]
    k = X_train.shape[1]
    adj_r_sq = 1 - (1 - r2)*(n-1)/(n-1-k)

    return rmse, r2, adj_r_sq

rmse, r2, adj_r_sq = reg_metrics(y_train,y_trainpred,X_train)
print("For train set\nRMSE =",rmse,"\nR-Square =", r2,"\nAdjusted R-Square =", adj_r_sq)

rmse, r2, adj_r_sq = reg_metrics(y_test,y_pred,X_train)
print("\nFor test set\nRMSE =",rmse,"\nR-Square =", r2,"\nAdjusted R-Square =", adj_r_sq)
```

```
For train set
RMSE = 4918.97306758612
R-Square = 0.9936897182329861
Adjusted R-Square = 0.9936803538123016

For test set
RMSE = 23636.141364323794
R-Square = 0.8024245569791166
Adjusted R-Square = 0.8012465091927911
```

In [85]:

```python
from sklearn.tree import plot_tree
fig = plt.figure(figsize=(20,15))
_ = plot_tree(DT,
              feature_names=list(X_train),
              class_names = ["0","1"],
              filled=True)
```



Model <= 127.5
gini = 0.5
samples = 614
value = [307.0, 307.0]
class = 0
Driven_Wheels <= 28.5
gini = 0.428
samples = 389
value = [240.394, 108.099]
class = 0
Engine Cylinders <= 29.95
gini = 0.376
samples = 225
value = [66.606, 198.901]
class = 1
Engine Cylinders <= 30.95
gini = 0.269
samples = 217
value = [150.82, 28.826]
class = 0
Engine Cylinders <= 26.25
gini = 0.498
samples = 172
value = [89.574, 79.272]
class = 0
Make <= 1.5
gini = 0.5
samples = 62
value = [30.623, 31.709]
class = 1
gini = 0.291
samples = 163
value = [35.983, 167.192]
class = 1
gini = 0.057
samples = 126
value = [94.933, 2.883]
class = 0
Driven_Wheels <= 22.5
gini = 0.433
samples = 91
value = [55.888, 25.944]
class = 0
Engine Cylinders <= 9.65
gini = 0.184
samples = 35
value = [25.264, 2.883]
class = 0
Model <= 89.5
gini = 0.496
samples = 137
value = [64.309, 76.39]
class = 1
gini = 0.0
samples = 13
value = [9.953, 0.0]
class = 0
Engine Cylinders <= 23.2
gini = 0.478
samples = 49
value = [20.671, 31.709]
class = 1
gini = 0.0
samples = 22
value = [16.843, 0.0]
class = 0
gini = 0.48
samples = 69
value = [39.045, 25.944]
class = 0
gini = 0.0
samples = 2
value = [0.0, 2.883]
class = 1
gini = 0.0
samples = 33
value = [25.264, 0.0]
class = 0
gini = 0.145
samples = 23
value = [16.843, 1.441]
class = 0
Transmission Type <= 0.2
gini = 0.475
samples = 114
value = [47.466, 74.948]
class = 1
gini = 0.0
samples = 7
value = [5.359, 0.0]
class = 0
gini = 0.439
samples = 42
value = [15.312, 31.709]
class = 1
gini = 0.308
samples = 18
value = [12.249, 2.883]
class = 0
gini = 0.441
samples = 96
value = [35.217, 72.066]
class = 1