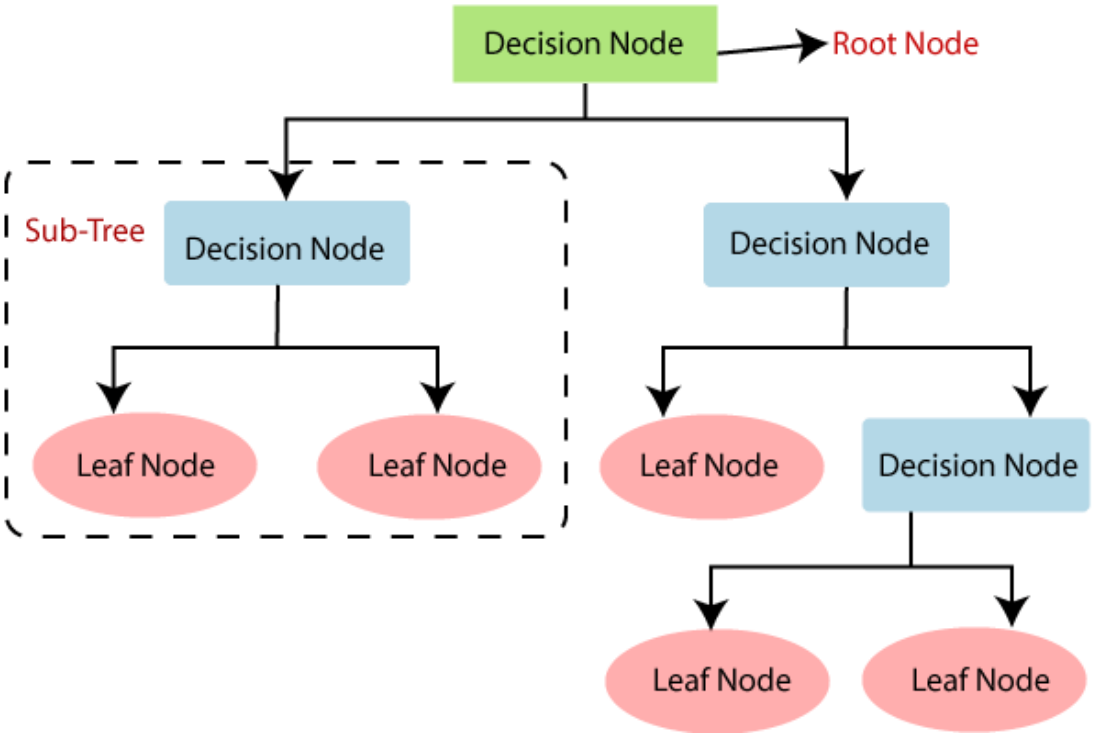
3. Decision Tree Classifier

- Supervised Learning Model
- Tree structure Model

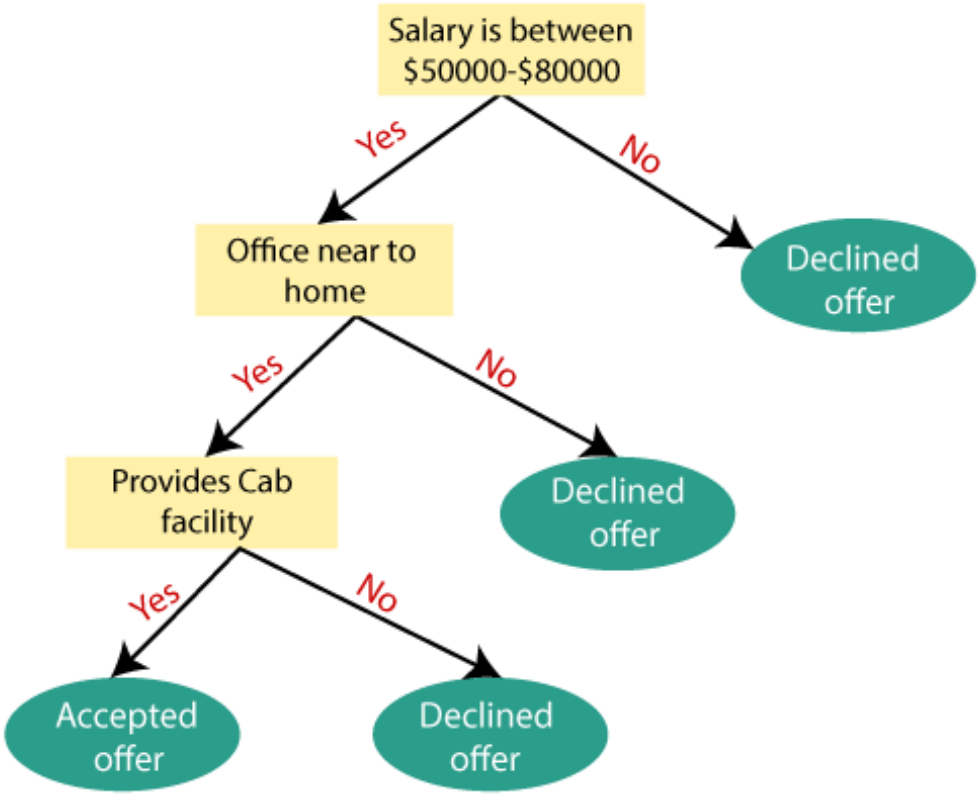
A decision tree classifier is a supervised machine learning algorithm that uses a tree-like structure to make predictions or classify input data. It recursively partitions the input space based on the features to create a tree of decision nodes and leaf nodes. Each decision node represents a feature and a threshold, while each leaf node represents a class label or a prediction.



The decision tree classifier operates by recursively splitting the input data based on the values of different features. It partitions the data into subsets at each decision node based on the selected feature and its threshold value. This process continues until the algorithm reaches a stopping criterion, such as reaching a maximum depth, a minimum number of samples, or when all samples in a subset belong to the same class.

The splitting process aims to maximize the homogeneity or purity of the subsets. Various splitting criteria can be used, with the most common being Gini impurity and entropy. Gini impurity measures the probability of misclassifying a randomly chosen sample if it were

labeled randomly according to the distribution of classes in the subset. Entropy, on the other hand, measures the level of impurity or randomness in the subset.



Advantages of Decision Tree Classifier:

- **Interpretability:** Decision trees are easy to understand and interpret. The structure of the tree allows for transparent decision-making, as each path from the root to a leaf represents a set of rules that lead to a particular prediction or classification.
- **Handling Non-linearity:** Decision trees can handle non-linear relationships between features and the target variable without requiring complex transformations. They can capture interactions and non-linear decision boundaries by recursively splitting the data based on the features' values.
- **Feature Importance:** Decision trees provide a measure of feature importance by evaluating the influence of each feature in the tree structure. This information can be valuable for feature selection and understanding the underlying factors driving the predictions.
- **Handling Missing Values:** Decision trees can handle missing values in the dataset. They can evaluate the available features and select the optimal split based on the available data.

Limitations of Decision Tree Classifier:

- **Overfitting:** Decision trees have a tendency to overfit the training data, especially when the tree becomes deep and complex. Overfitting occurs when the tree captures noise or irrelevant patterns in the training data, leading to poor generalization on unseen data.
- **Instability:** Decision trees can be sensitive to small changes in the training data, leading to different tree structures and predictions. This instability can be reduced by using ensemble methods like random forests or boosting.
- **Lack of Smoothness:** Decision trees produce piecewise constant predictions, meaning they create boundaries between regions with different predictions. This lack of smoothness may not be suitable for problems where a smooth decision boundary is desired.

Applications of Decision Tree Classifier:

- **Credit Scoring:** Decision trees are commonly used in credit scoring to assess the creditworthiness of individuals or businesses. By considering factors such as income, credit history, and debt-to-income ratio, decision trees can predict the likelihood of a borrower defaulting on a loan.
- **Customer Churn Prediction:** Decision trees can predict customer churn by analyzing factors like customer demographics, purchase behavior, and service usage. This helps businesses identify customers at risk of churning and take proactive measures to retain them.
- **Disease Diagnosis:** Decision trees are employed in medical diagnosis to predict the presence of certain diseases or conditions. By considering symptoms, patient characteristics, and medical test results, decision trees can assist in diagnosing diseases and recommending appropriate treatments.
- **Fraud Detection:** Decision trees are used in fraud detection systems to identify fraudulent transactions or activities. By analyzing transaction patterns, user behavior, and other relevant features, decision trees can flag suspicious activities for further investigation.

Implementing Decision Tree Classifier using Python

Dataset Required

https://drive.google.com/file/d/1V6yFU3nDdx9R56yOzy6GxHPq-Dav2A4K/view?usp=share_link

Importing Required Libraries

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
import seaborn as sn
```

Importing (Reading) Datasets

```
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin',
             'bmi', 'pedigree', 'age', 'label']
data = pd.read_csv('/content/diabets.csv', header= None,
       names=col_names)
print(data.shape)
data.head()
```

```
(768, 9)
```

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
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

Checking for any null values in dataset

```
data.isnull().sum()
```

```
pregnant    0
glucose     0
bp          0
skin        0
insulin     0
bmi         0
pedigree    0
age         0
label       0
dtype: int64
```

Assigning dependent and independent variables

```
feature_cols = ['pregnant','insulin', 'bmi',
                'age','glucose','bp', 'pedigree']
x=data[feature_cols]
y=data.label
```

splitting the dataset into Training and Testing Dataset

```
x_train, x_test, y_train, y_test = train_test_split(x,y,
     test_size=0.2, random_state=5)
display(x_train.shape, y_train.shape, x_test.shape,
     y_test.shape)
```

```
(614, 7)
(614,)
(154, 7)
(154,)
```

Fitting the Model (Decision Tree Classifier)

```
model=  DecisionTreeClassifier(criterion='entropy',random_state=5)
model.fit(x_train, y_train)
y_pred=model.predict(x_test)
print('y_pred: ', y_pred)
```

```
y_pred:  [1 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 1 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 0 1 0
1
 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 1 1 1 1 1 0 1
 1 1 0 1 1 0 1 1 1 1 0 0 0 0 0 1 0 0 0 1 0 1 1 1 0 0 0 0 1 1 0 1 0 0 1 0 0
 0 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 1 0 1 0 0 0 1
 0 0 1 1 0 1]
```

Evaluation Metrics

```
conf_mat=metrics.confusion_matrix(y_test, y_pred)
print('Confusion Matrix : ', conf_mat)
Accuracy_score=metrics.accuracy_score(y_test, y_pred)
print('Accuracy Score : ', Accuracy_score)
print('Accuracy in Percentage : ', int(Accuracy_score*100),'%')
```

```
Confusion Matrix :  [[77 23]
 [19 35]]
Accuracy Score :  0.7272727272727273
Accuracy in Percentage :  72 %
```

```
conf_mat=pd.crosstab(y_test,  y_pred,  rownames=['Actual'],
    colnames=['Predicted'])
sn.heatmap(conf_mat, annot=True)
```