1. Installing Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import *
from sklearn.model_selection import *
from sklearn import tree
from sklearn import metrics
import graphviz
from six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
from time import time
from scipy.stats import randint as sp_randint
```

In [2]:
```
# Loading data
df = pd.read_csv('E:\Data Scientiest\Datasets\employee.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	Educatio
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life S
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life S
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life S
4	27	No	Travel_Rarely	591	Research & Development	2	1	M

5 rows × 35 columns

Our target variable is Attrition which means we are going to predict that the employee of a company will leave or stay in a company so Attrition is our Dependent Variable

In [4]:
```
# This will give us the Dtypes of every variable and also show wether there are null
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Age                       1470 non-null   int64
 1   Attrition                 1470 non-null   object
 2   BusinessTravel            1470 non-null   object
 3   DailyRate                 1470 non-null   int64
 4   Department                1470 non-null   object
```

```
 5   DistanceFromHome          1470 non-null   int64
 6   Education                 1470 non-null   int64
 7   EducationField            1470 non-null   object
 8   EmployeeCount             1470 non-null   int64
 9   EmployeeNumber            1470 non-null   int64
 10  EnvironmentSatisfaction   1470 non-null   int64
 11  Gender                    1470 non-null   object
 12  HourlyRate                1470 non-null   int64
 13  JobInvolvement            1470 non-null   int64
 14  JobLevel                  1470 non-null   int64
 15  JobRole                   1470 non-null   object
 16  JobSatisfaction           1470 non-null   int64
 17  MaritalStatus             1470 non-null   object
 18  MonthlyIncome             1470 non-null   int64
 19  MonthlyRate               1470 non-null   int64
 20  NumCompaniesWorked        1470 non-null   int64
 21  Over18                    1470 non-null   object
 22  OverTime                  1470 non-null   object
 23  PercentSalaryHike         1470 non-null   int64
 24  PerformanceRating         1470 non-null   int64
 25  RelationshipSatisfaction  1470 non-null   int64
 26  StandardHours             1470 non-null   int64
 27  StockOptionLevel          1470 non-null   int64
 28  TotalWorkingYears         1470 non-null   int64
 29  TrainingTimesLastYear     1470 non-null   int64
 30  WorkLifeBalance           1470 non-null   int64
 31  YearsAtCompany            1470 non-null   int64
 32  YearsInCurrentRole        1470 non-null   int64
 33  YearsSinceLastPromotion   1470 non-null   int64
 34  YearsWithCurrManager      1470 non-null   int64
dtypes: int64(26), object(9)
memory usage: 402.1+ KB
```

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
Age                         0
Attrition                   0
BusinessTravel              0
DailyRate                   0
Department                  0
DistanceFromHome            0
Education                   0
EducationField              0
EmployeeCount               0
EmployeeNumber              0
EnvironmentSatisfaction     0
Gender                      0
HourlyRate                  0
JobInvolvement              0
JobLevel                    0
JobRole                     0
JobSatisfaction             0
MaritalStatus               0
MonthlyIncome               0
MonthlyRate                 0
NumCompaniesWorked          0
Over18                      0
OverTime                    0
PercentSalaryHike           0
PerformanceRating           0
RelationshipSatisfaction    0
StandardHours               0
StockOptionLevel            0
TotalWorkingYears           0
TrainingTimesLastYear       0
WorkLifeBalance             0
YearsAtCompany              0
YearsInCurrentRole          0
YearsSinceLastPromotion     0
```

```
YearsWithCurrManager        0
dtype: int64
```

In [6]:
```
# df.describe() will give us the details of our data that what is the minimum value
# what is the total count in the variable , and it also give the mean and standard d
df.describe()
```

Out[6]:

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumbe
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.0	1470.00000
mean	36.923810	802.485714	9.192517	2.912925	1.0	1024.86530
std	9.135373	403.509100	8.106864	1.024165	0.0	602.02433
min	18.000000	102.000000	1.000000	1.000000	1.0	1.00000
25%	30.000000	465.000000	2.000000	2.000000	1.0	491.25000
50%	36.000000	802.000000	7.000000	3.000000	1.0	1020.50000
75%	43.000000	1157.000000	14.000000	4.000000	1.0	1555.75000
max	60.000000	1499.000000	29.000000	5.000000	1.0	2068.00000

8 rows × 26 columns

In [7]:
```
# So here we are converting our Yes and No in Attrition as 1 and 0 respectively
df.loc[df['Attrition']=='No','Attrition']=0
df.loc[df['Attrition']=='Yes','Attrition']=1
```

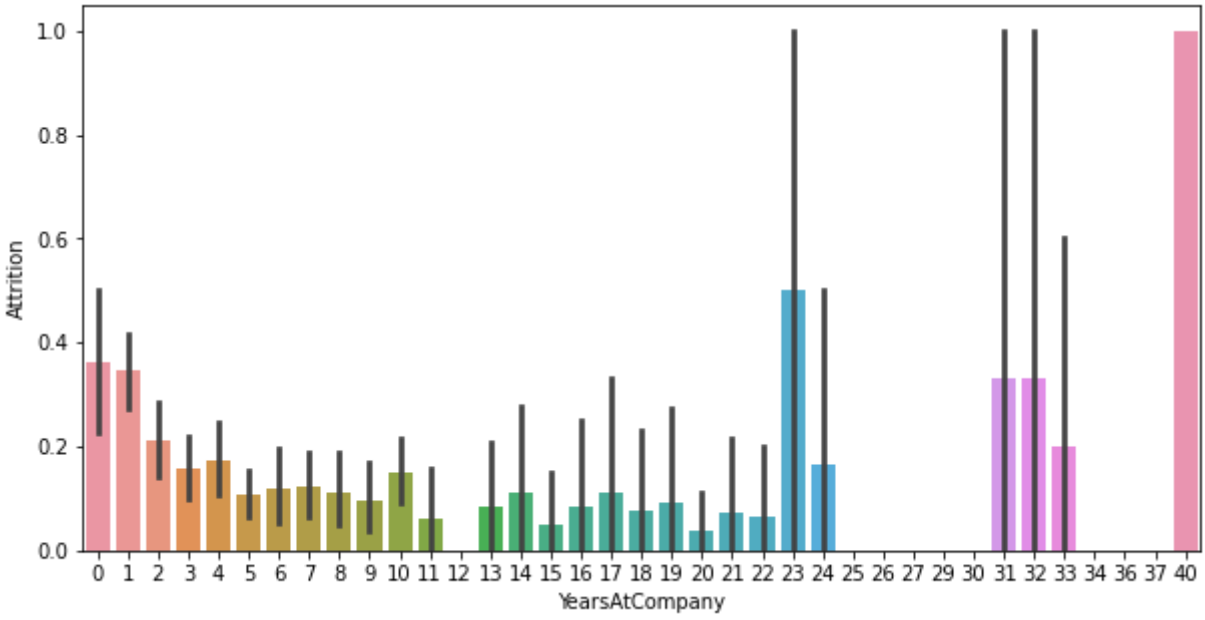
EDA

In [8]:
```
plt.figure(figsize=(20,10))
sns.heatmap(df.corr(),annot=True)
plt.show()
```

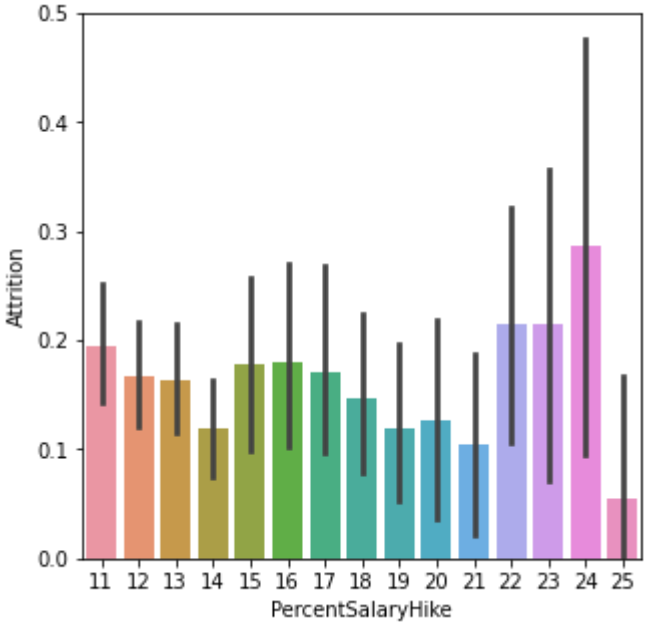
In [9]:
```
sns.pairplot(df)
```

Out[9]: <seaborn.axisgrid.PairGrid at 0x1e26cb0f040>

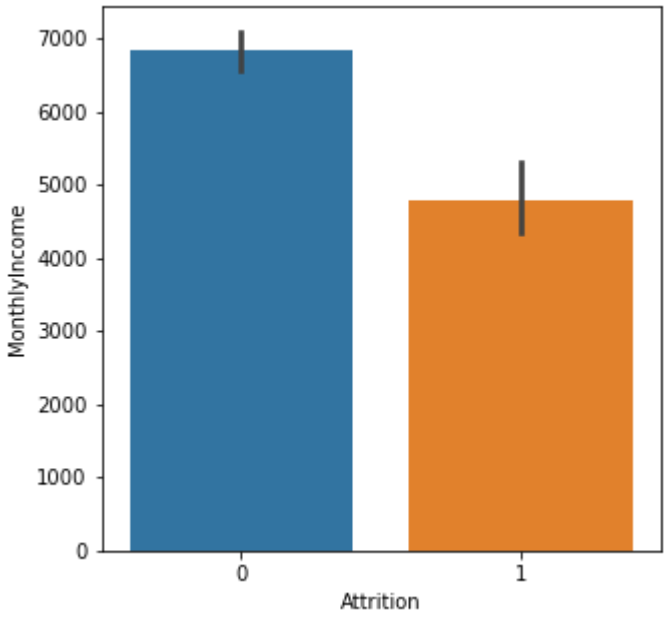
In [10]:
```
plt.figure(figsize=(10,5))
sns.barplot(x='YearsAtCompany',y='Attrition',data=df)
plt.show()
```



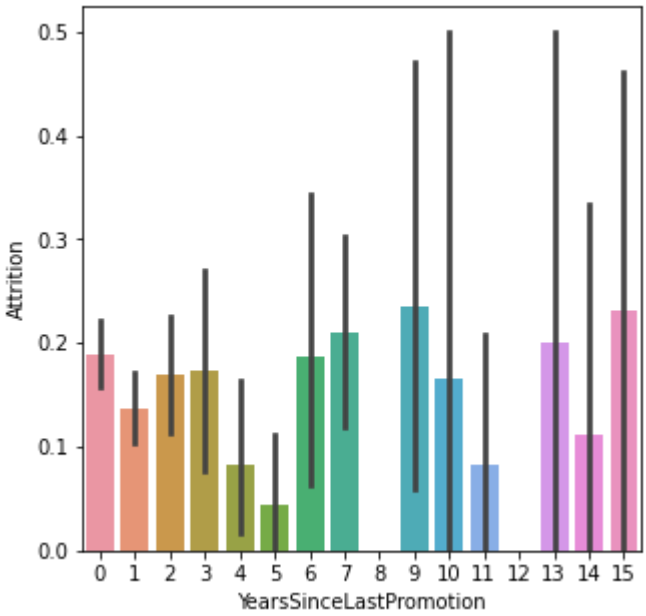
In [11]:
```
plt.figure(figsize=(5,5))
sns.barplot(x='PercentSalaryHike',y='Attrition',data=df)
plt.show()
```



In [12]:
```
plt.figure(figsize=(5,5))
sns.barplot(x='Attrition',y='MonthlyIncome',data=df)
plt.show()
```



In [13]:
```
plt.figure(figsize=(5,5))
sns.barplot(x='YearsSinceLastPromotion',y='Attrition',data=df)
plt.show()
```



In [14]:
```
df.columns
```

Out[14]:
```
Index(['Age', 'Attrition', 'BusinessTravel', 'DailyRate', 'Department',
       'DistanceFromHome', 'Education', 'EducationField', 'EmployeeCount',
       'EmployeeNumber', 'EnvironmentSatisfaction', 'Gender', 'HourlyRate',
       'JobInvolvement', 'JobLevel', 'JobRole', 'JobSatisfaction',
       'MaritalStatus', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
       'Over18', 'OverTime', 'PercentSalaryHike', 'PerformanceRating',
       'RelationshipSatisfaction', 'StandardHours', 'StockOptionLevel',
       'TotalWorkingYears', 'TrainingTimesLastYear', 'WorkLifeBalance',
       'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion',
       'YearsWithCurrManager'],
      dtype='object')
```

In [15]:
```
df.describe(include='object')
```

Out[15]:

	Attrition	BusinessTravel	Department	EducationField	Gender	JobRole	MaritalStatus	Ov
count	1470	1470	1470	1470	1470	1470	1470	
unique	2	3	3	6	2	9	3	
top	0	Travel_Rarely	Research & Development	Life Sciences	Male	Sales Executive	Married	
freq	1233	1043	961	606	882	326	673	

In [16]:
```
print("Number of people who left the Orginazation",(1-1233/1470)*100,"Percent of peo
```

```
Number of people who left the Orginazation 16.12244897959184 Percent of people
```

In [17]:
```
df.describe(include='number')
```

Out[17]:

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumbe
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.0	1470.00000
mean	36.923810	802.485714	9.192517	2.912925	1.0	1024.86530
std	9.135373	403.509100	8.106864	1.024165	0.0	602.02433
min	18.000000	102.000000	1.000000	1.000000	1.0	1.00000
25%	30.000000	465.000000	2.000000	2.000000	1.0	491.25000

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumbe
50%	36.000000	802.000000	7.000000	3.000000	1.0	1020.50000
75%	43.000000	1157.000000	14.000000	4.000000	1.0	1555.75000
max	60.000000	1499.000000	29.000000	5.000000	1.0	2068.00000

8 rows × 26 columns

In [18]:
```
df['Attrition']
```

Out[18]:
```
0       1
1       0
2       1
3       0
4       0
       ..
1465    0
1466    0
1467    0
1468    0
1469    0
Name: Attrition, Length: 1470, dtype: object
```

In [19]:
```
# Here we are storing dependent variable in another variable
attrition = df['Attrition']
```

In [20]:
```
# Now dropping Dependent variable from a main dataset
df = df.drop('Attrition',axis=1)
```

In [21]:
```
# Now here we will get the dummy values for all the categorical variable
df = pd.get_dummies(df)
```

In [22]:
```
# here we will Concat our dummy data set with the dependent variable
df = pd.concat([attrition,df],axis=1)
```

In [23]:
```
df.head()
```

Out[23]:

	Attrition	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumber	Env
0	1	41	1102	1	2	1	1	
1	0	49	279	8	1	1	2	
2	1	37	1373	2	2	1	4	
3	0	33	1392	3	4	1	5	
4	0	27	591	2	1	1	7	

5 rows × 56 columns

In [24]:
```
# Here we are seprating the Dependent Variable and the Independent Variable
x = df.drop('Attrition',axis=1)
y = df['Attrition']
```

In [25]:
```
# Now Here we will split the data as training set and the testing set with the help
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.30,random_state=123
```

In [26]:
```
# Now we will print the shape of our training set and the testing set
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1029, 55)
(441, 55)
(1029,)
(441,)
```

Creating the Decision_Tree_Clssification_Model

In [27]:
```
# creating the decision tree classifier tree model with a random stae of 123
dt = tree.DecisionTreeClassifier(random_state=123)
y_train = y_train.astype(np.number)
#  fitting the model
dt.fit(x_train,y_train)
# here storing our predicted value in the y_pred variable
y_pred = dt.predict(x_test)
```

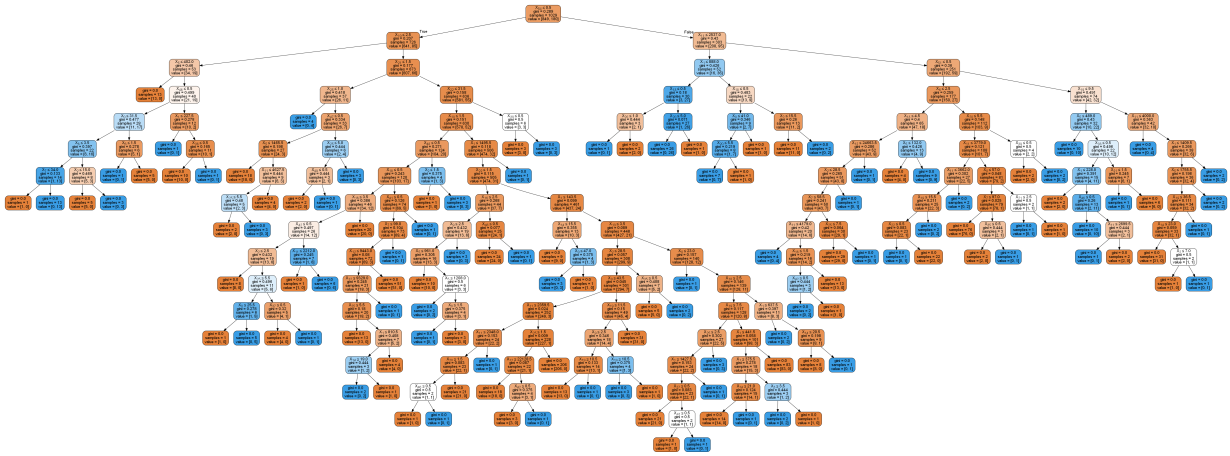
In [28]:
```
y_pred = y_pred.astype(np.number)
y_test = y_test.astype(np.number)
```

In [29]:
```
# Here we are going to display the Accuracy, Precision and Recall for you model that
print('Accuracy',metrics.accuracy_score(y_test,y_pred))
print('Precision',metrics.precision_score(y_test,y_pred))
print('recall',metrics.recall_score(y_test,y_pred))
```

```
Accuracy 0.8095238095238095
Precision 0.3150684931506849
recall 0.40350877192982454
```

In [30]:
```
# Now here we will display our decision that we have made to do the analysis
dot_data = StringIO()
export_graphviz(dt,out_file=dot_data,filled=True,rounded= True,special_characters=Tr
graph=pydotplus.graph_from_dot_data(dot_data.getvalue())
graphviz.Source(graph)
Image(graph.create_png())
```

Out[30]:



From the above model we can say that our model is having an Accuracy of 80%, but from the tree we can say that there might be a case of overfitting , so now to overcome this problem we will do hyperparameter tunnig, with the help of RandomSearchCV and the GridSearchCV

RandomSearch (Hyperparameter Tunning)

In [31]:
```
#create function to generate model report
def report(results,n_top=3):
    for i in range(1,n_top+1):
        candidates = np.flatnonzero(results['rank_test_score']==i)
        for candidate in candidates:
            print('Model with rank : {0}'.format(i))
            print('Mean Validation Score: {0:.3f} (std:{1:.3f})'.format(results['mea
                  results['std_test_score'][candidate]))
            print('Parmeters: {0}'.format(results['params'][candidate]))
            print("")
```

In [32]:
```
#specify parameters nd distrubution to sample form
param_dist={'max_depth':sp_randint(2,12),
           'max_features':sp_randint(1,11),
           'min_samples_split':sp_randint(2,20),
           'min_samples_leaf':sp_randint(1,20),
           'max_leaf_nodes':sp_randint(2,10),
           'criterion':['gini','entropy']}
```

In [33]:
```
#run randomized search
n_iter_search = 20
random_search = RandomizedSearchCV(dt,param_distributions=param_dist,n_iter=n_iter_s
```

In [34]:
```
# Here we will get our top 3 models that will have better accuracy than our previous
start = time()
random_search.fit(x_train,y_train)
print('Randomized search tooo %.2f seconds for %d candidates''parameter settings '%(
report(random_search.cv_results_)
```

```
Randomized search tooo 0.85 seconds for 20 candidatesparameter settings
Model with rank : 1
Mean Validation Score: 0.839 (std:0.007)
Parmeters: {'criterion': 'gini', 'max_depth': 10, 'max_features': 8, 'max_leaf_node
s': 8, 'min_samples_leaf': 8, 'min_samples_split': 15}

Model with rank : 2
Mean Validation Score: 0.834 (std:0.006)
Parmeters: {'criterion': 'entropy', 'max_depth': 3, 'max_features': 9, 'max_leaf_nod
es': 6, 'min_samples_leaf': 3, 'min_samples_split': 9}

Model with rank : 3
Mean Validation Score: 0.832 (std:0.008)
Parmeters: {'criterion': 'gini', 'max_depth': 10, 'max_features': 6, 'max_leaf_node
s': 9, 'min_samples_leaf': 2, 'min_samples_split': 11}
```

Now we will again fit our decision tree model with the parameters that we have got which will help us to increase our accuracy

In [57]:
```
dt_random_search = tree.DecisionTreeClassifier(criterion ='gini',
                                               max_depth= 10, max_features= 6,
                                               max_leaf_nodes= 9, min_samples_leaf=
```

In [58]:
```
dt_random_search.fit(x_train,y_train)
y_pred_random_search = dt_random_search.predict(x_test)
```

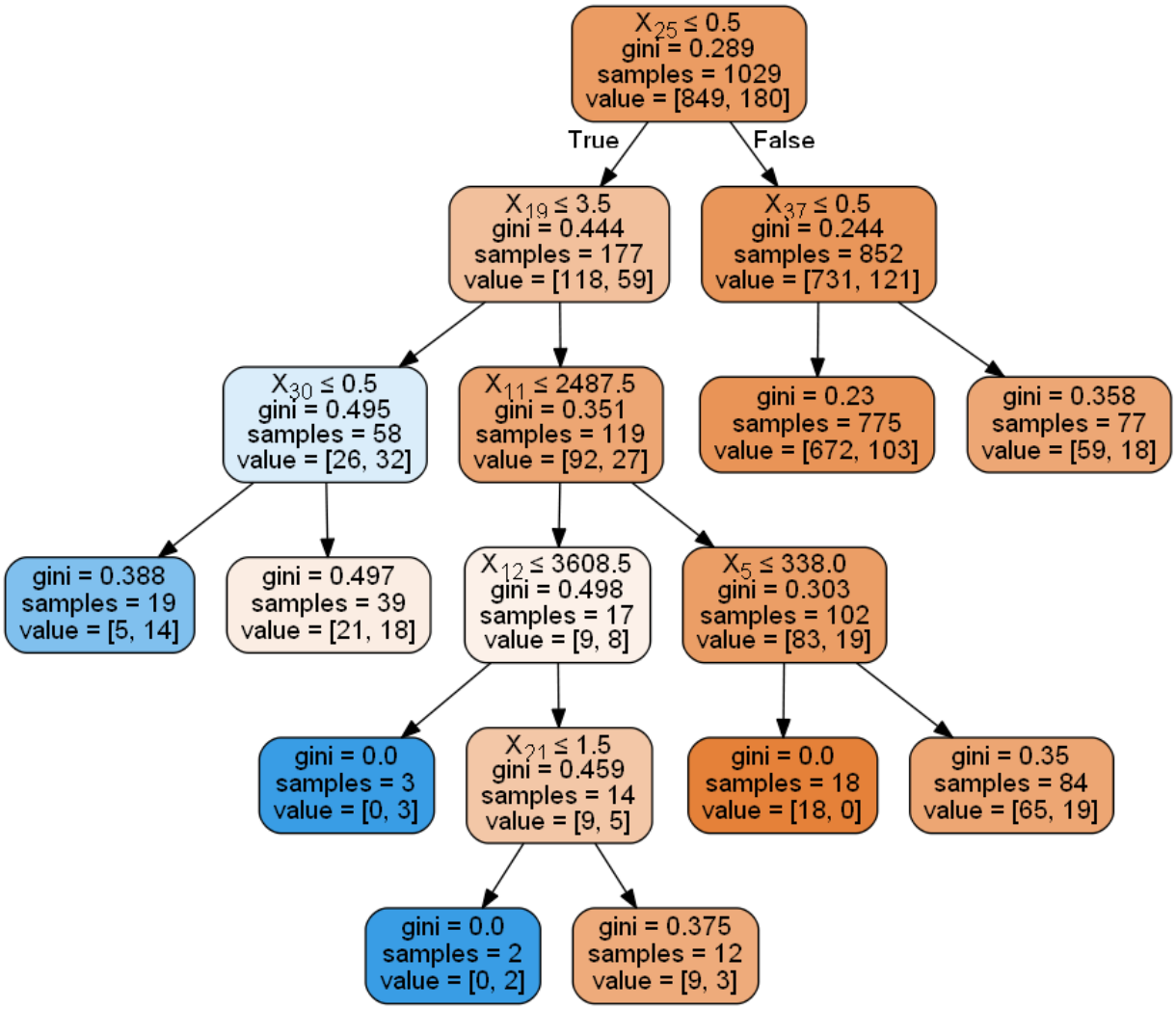
In [59]:
```
print('Accuracy',metrics.accuracy_score(y_test,y_pred_random_search))
print('Precision',metrics.precision_score(y_test,y_pred_random_search))
print('recall',metrics.recall_score(y_test,y_pred_random_search))
```

```
Accuracy 0.8752834467120182
Precision 0.5714285714285714
recall 0.14035087719298245
```

In [60]:
```
# Now here we will display our decision that we have made to do the analysis
```

```
dot_data = StringIO()
export_graphviz(dt_random_search,out_file=dot_data,filled=True,rounded= True,special
graph=pydotplus.graph_from_dot_data(dot_data.getvalue())
graphviz.Source(graph)
Image(graph.create_png())
```

Out[60]:



Now you can see that with the help of Randomsearch we have increased our Accuracy from 80% to 87% so we can say thatour model has improved and our size of tree has also reduced

GridSearchCV (Hyperparameter Tunning)

In [39]:
```
param_grid={'max_depth':[5,7,9,11,13],
            'max_features':[4,5,6,7,8,10,12],
            'min_samples_split':[2,4,6,8,10,12],
            'min_samples_leaf':[15,16,17,18,19,20],
            'max_leaf_nodes':[5,6,7,8,9],
            'criterion':['gini','entropy']}
```

In [40]:
```
grid_search=GridSearchCV(dt,param_grid=param_grid,cv=5)
```

In [41]:
```
start =time()
grid_search.fit(x_train,y_train)
print('Gridesearchscv took %.2f seconds for %d candidates parameters settings'%(time
report(grid_search.cv_results_)
```

```
Gridesearchscv took 541.69 seconds for 12600 candidates parameters settings
Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
```

```
Parmeters: {'criterion': 'gini', 'max_depth': 5, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 2}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 5, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 4}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 5, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 6}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 5, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 8}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 5, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 10}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 5, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 12}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 7, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 2}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 7, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 4}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 7, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 6}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 7, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 8}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 7, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 10}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 7, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 12}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 9, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 2}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 9, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 4}

Model with rank : 1
```

```
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 9, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 6}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 9, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 8}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 9, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 10}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 9, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 12}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 11, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 2}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 11, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 4}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 11, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 6}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 11, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 8}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 11, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 10}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 11, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 12}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 13, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 2}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 13, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 4}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 13, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 6}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 13, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 8}
```

```
Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 13, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 10}

Model with rank : 1
Mean Validation Score: 0.843 (std:0.009)
Parmeters: {'criterion': 'gini', 'max_depth': 13, 'max_features': 12, 'max_leaf_node
s': 7, 'min_samples_leaf': 17, 'min_samples_split': 12}
```

In [46]:
```
dt_grid_search = tree.DecisionTreeClassifier(criterion ='gini',
                                             max_depth= 5, max_features= 12,
                                             max_leaf_nodes= 7, min_samples_leaf=
                                            min_samples_split= 2,random_state=100)
```

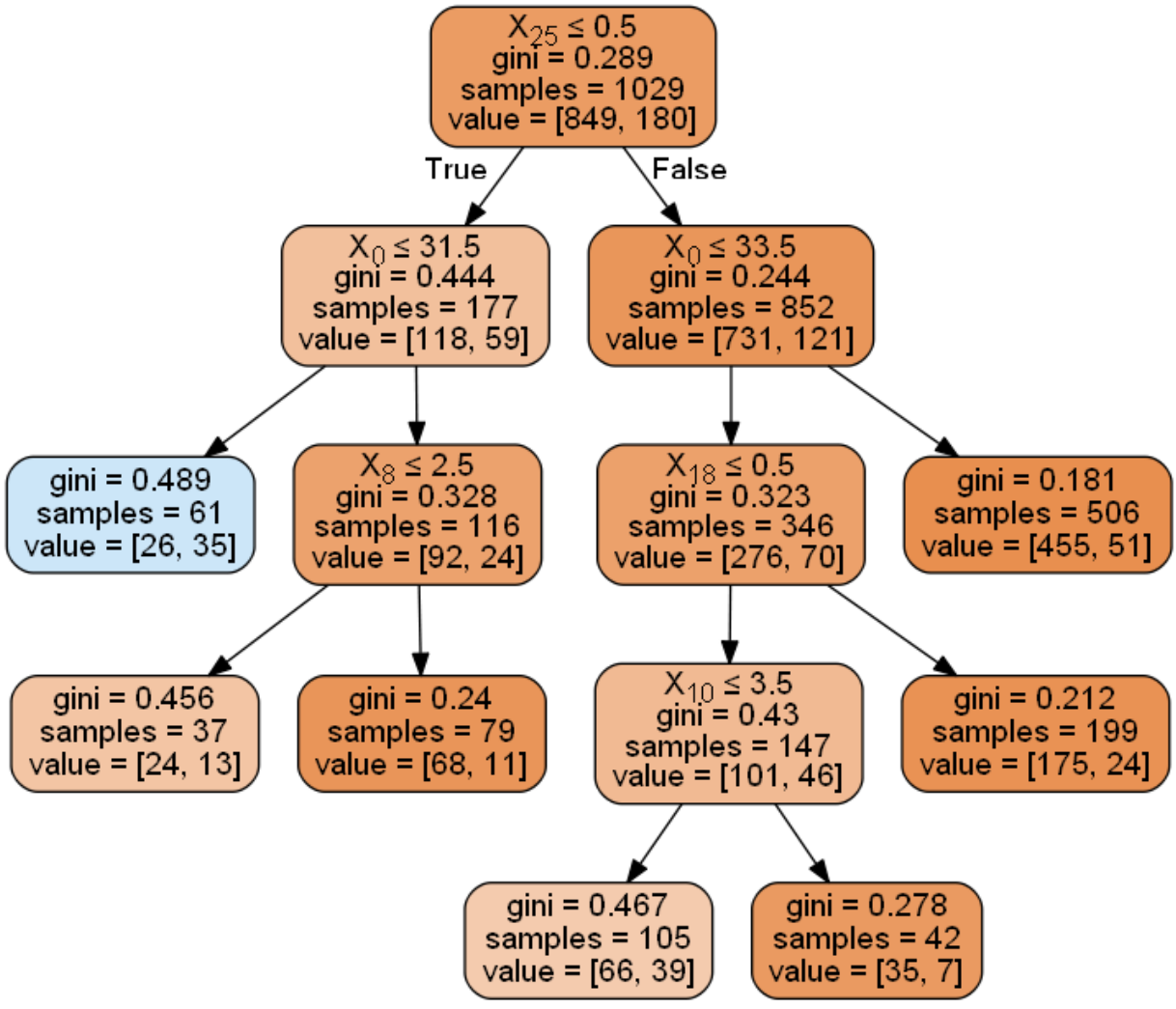
In [47]:
```
dt_grid_search.fit(x_train,y_train)
y_pred_grid_search = dt_grid_search.predict(x_test)
```

In [48]:
```
print('Accuracy',metrics.accuracy_score(y_test,y_pred_grid_search))
print('Precision',metrics.precision_score(y_test,y_pred_grid_search))
print('recall',metrics.recall_score(y_test,y_pred_grid_search))
```

```
Accuracy 0.8594104308390023
Precision 0.43902439024390244
recall 0.3157894736842105
```

In [49]:
```
dot_data = StringIO()
export_graphviz(dt_grid_search,out_file=dot_data,filled=True,rounded= True,special_c
graph=pydotplus.graph_from_dot_data(dot_data.getvalue())
graphviz.Source(graph)
Image(graph.create_png())
```

Out[49]:



so now with the help of grid search we got the accuracy of 85% which is less then the random search which was 87% so we will be selecting the random search results