# Predicting IBM Employee Attrition Python Jupyter Notebook

## Part 4 - Build a Decision Tree Model

**Import numpy and pandas.**

In [1]:
```
import numpy as np
import pandas as pd
```

**Import data visualization libraries and set %matplotlib inline.**

In [2]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

**Import churn modeling pickle file into a Pandas dataframe called churn_model2.**

In [3]:
```
churn_model2 = pd.read_pickle('../data/churn_modeling_data.pickle')
```

**Define X and y to split data into training and test data sets, and construct decision tree model on entire modeling data set.**

In [4]:
```
X = churn_model2.drop(['Churn'], axis=1)
y = churn_model2['Churn']
```

**Tune decision tree model to avoid overfitting.**

In [5]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.cross_validation import cross_val_score
```

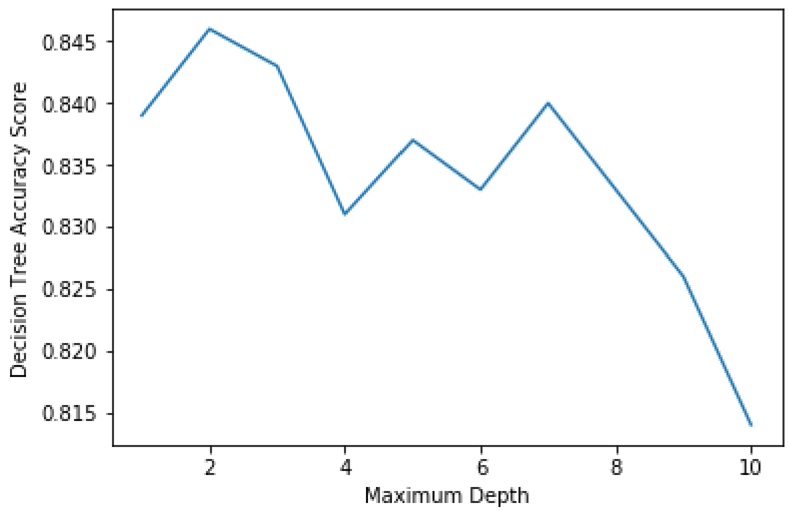
```
C:\Users\kyrma\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\cros
s_validation.py:41: DeprecationWarning: This module was deprecated in version
0.18 in favor of the model_selection module into which all the refactored class
es and functions are moved. Also note that the interface of the new CV iterator
s are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [6]:
```
max_depth_range1 = range(1, 11)

acc_scores3 = []

for depth in max_depth_range1:
    treereg = DecisionTreeClassifier(max_depth=depth, random_state=1)
    scores3 = cross_val_score(treereg, X, y, cv=14, scoring='accuracy')
    acc_scores3.append(scores3.mean().round(3))

print(acc_scores3)
```

```
[0.839, 0.846, 0.843, 0.831, 0.837, 0.833, 0.84, 0.833, 0.826, 0.814]
```

In [7]:
```
plt.plot(max_depth_range1, acc_scores3)
plt.xlabel('Maximum Depth')
plt.ylabel('Decision Tree Accuracy Score')
```

Out[7]:
```
Text(0,0.5,'Decision Tree Accuracy Score')
```




- The maximum depth of the decision tree should be 2 since the tree has the maximum accuracy score of 0.846.

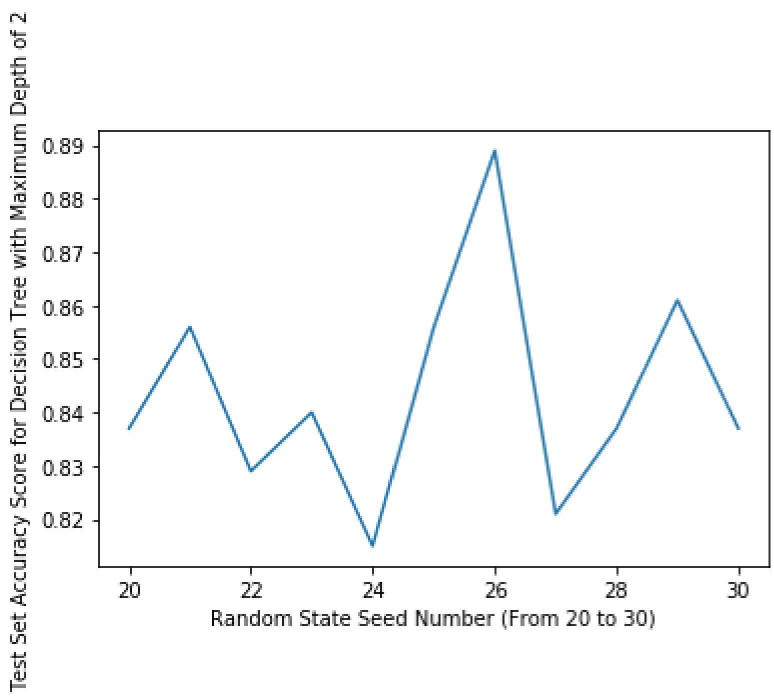
**Decide which random state seed number will produce the highest test set accuracy score for a decision tree with a maximum depth of 2.**

In [8]:
```
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

In [9]:
```
seed_range2 = range(20, 31)

acc_scores4 = []

for seed in seed_range2:
    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=seed)
    treereg = DecisionTreeClassifier(max_depth=2, random_state=1)
    treereg.fit(X_train, y_train)
    y_pred = treereg.predict(X_test)
    acc_scores4.append(metrics.accuracy_score(y_test, y_pred).round(3))
```

In [10]:
```
plt.plot(seed_range2, acc_scores4)
plt.xlabel('Random State Seed Number (From 20 to 30)')
plt.ylabel('Test Set Accuracy Score for Decision Tree with Maximum Depth of 2')
```

Out[10]:
```
Text(0,0.5,'Test Set Accuracy Score for Decision Tree with Maximum Depth of 2')
```




- 26 is the random state seed number that will produce the highest test set accuracy score for the decision tree with a maximum depth of 2.

**Split churn / attrition modeling data into training and test sets.**

In [11]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=26)
```

In [12]:
```
print(churn_model2.shape)
print(X_train.shape)
print(X_test.shape)
```

```
(1470, 11)
(1102, 10)
(368, 10)
```

**Fit a decision tree with maximum depth of 2 on training data set.**

In [13]:
```
treereg = DecisionTreeClassifier(max_depth=2, random_state=1)
treereg.fit(X_train, y_train)
```

Out[13]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=2,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=1,
            splitter='best')
```

**Make predictions on test data set and calculate test set accuracy.**

In [14]:
```
y_pred = treereg.predict(X_test)
print(metrics.accuracy_score(y_test, y_pred).round(3))
```

```
0.889
```

**Compute null accuracy manually.**

In [15]:
```
print(1 - y_test.mean())
```

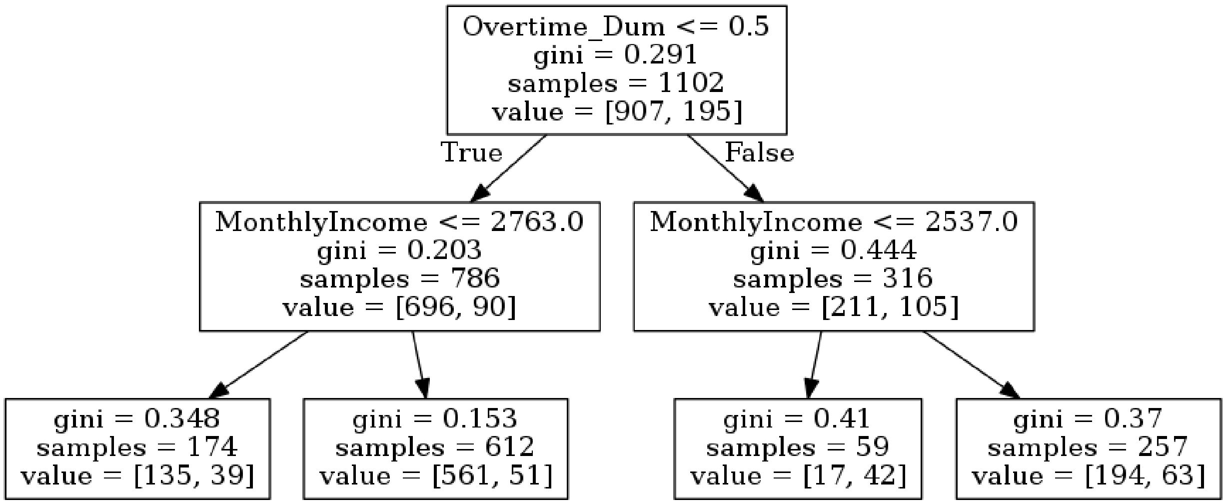
```
0.8858695652173914
```

**Create GraphViz file of decision tree and display it in the notebook.**

In [16]:
```
from sklearn.tree import export_graphviz
from IPython.display import Image
from IPython.display import display
```

In [17]:
```
feature_cols = ['Age', 'BusTravLevel', 'DistanceFromHome', 'EnvironmentSatisfacti
                'Overtime_Dum', 'Sales_Rep', 'Single', 'StockOptionLevel']
export_graphviz(treereg, out_file='Churn_Decision_Tree.dot', feature_names=featur
```

In [18]:
```
display(Image('Churn_Decision_Tree.png'))
```




**List out decision tree predictor features and their respective importances.**

In [19]:
```
pd.DataFrame({'feature':feature_cols, 'importance':treereg.feature_importances_})
```

Out[19]:

| | feature | importance |
|---|---|---|
| **5** | MonthlyIncome | 0.551401 |
| **6** | Overtime_Dum | 0.448599 |
| **0** | Age | 0.000000 |
| **1** | BusTravLevel | 0.000000 |
| **2** | DistanceFromHome | 0.000000 |
| **3** | EnvironmentSatisfaction | 0.000000 |
| **4** | JobInvolvement | 0.000000 |
| **7** | Sales_Rep | 0.000000 |
| **8** | Single | 0.000000 |
| **9** | StockOptionLevel | 0.000000 |

**Print confusion matrix to calculate test set accuracy and error rates plus precision and recall.**

In [20]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[320   6]
 [ 35   7]]
```

**Calculate test set accuracy rate.**

In [21]:
```
float(320 + 7) / float(320 + 6 + 35 + 7)
```

Out[21]:
```
0.8885869565217391
```

**Calculate test set misclassification / error rate.**

In [22]:
```
float(35 + 6) / float(320 + 6 + 35 + 7)
```

Out[22]:
```
0.11141304347826086
```

**Calculate precision to measure how confident the decision tree model is for capturing the positives in test set.**

In [23]:
```
float(7) / float(6 + 7)
```

Out[23]:
```
0.5384615384615384
```

**Calculate recall / sensitivity to measure how well the decision tree model is capturing the positives in test set.**

In [24]:
```
float(7) / float(35 + 7)
```

Out[24]:
```
0.16666666666666666
```

**Calculate specificity to measure how well the decision tree model is capturing the negatives in test set.**

In [25]:
```
float(320) / float(320 + 6)
```

Out[25]:
```
0.9815950920245399
```

**Print out test set classification report for logistic regression model.**

In [26]:
```
from sklearn.metrics import classification_report
```

In [27]:
```
print(classification_report(y_test, y_pred))
```

```
             precision    recall  f1-score   support

          0       0.90      0.98      0.94       326
          1       0.54      0.17      0.25        42

avg / total       0.86      0.89      0.86       368
```

**List out false positives in test data set.**

In [28]:
```
X_test[y_test < y_pred]
```

Out[28]:

| | Age | BusTravLevel | DistanceFromHome | EnvironmentSatisfaction | JobInvolvement | MonthlyInc |
|---|---|---|---|---|---|---|
| **1402** | 31 | 1 | 2 | 3 | 0 | 1 |
| **1028** | 41 | 1 | 5 | 1 | 3 | 2 |
| **925** | 42 | 1 | 7 | 1 | 3 | 2 |
| **889** | 27 | 1 | 14 | 0 | 2 | 2 |
| **1087** | 34 | 1 | 7 | 1 | 2 | 2 |
| **632** | 42 | 2 | 2 | 1 | 2 | 2 |

**List out false negatives in test data set.**

In [29]:

```
X_test[y_test > y_pred]
```

Out[29]:

| | Age | BusTravLevel | DistanceFromHome | EnvironmentSatisfaction | JobInvolvement | MonthlyInc |
|---|---|---|---|---|---|---|
| **1223** | 47 | 2 | 9 | 2 | 0 | 12 |
| **469** | 32 | 0 | 11 | 3 | 3 | 4 |
| **264** | 28 | 1 | 2 | 0 | 2 | 3 |
| **182** | 41 | 1 | 20 | 1 | 2 | 3 |
| **1438** | 23 | 2 | 9 | 3 | 2 | 1 |
| **849** | 43 | 1 | 9 | 0 | 0 | 5 |
| **50** | 48 | 1 | 1 | 0 | 1 | 5 |
| **1167** | 35 | 1 | 15 | 0 | 0 | 5 |
| **568** | 55 | 1 | 2 | 3 | 2 | 19 |
| **1379** | 27 | 2 | 22 | 0 | 1 | 2 |
| **562** | 33 | 1 | 1 | 3 | 2 | 2 |
| **1112** | 38 | 1 | 2 | 2 | 2 | 4 |
| **363** | 33 | 1 | 5 | 3 | 2 | 2 |
| **414** | 24 | 1 | 1 | 0 | 2 | 3 |
| **525** | 24 | 1 | 3 | 0 | 2 | 4 |
| **111** | 34 | 2 | 7 | 0 | 0 | 6 |
| **689** | 20 | 1 | 4 | 0 | 2 | 2 |
| **667** | 41 | 1 | 2 | 1 | 0 | 2 |
| **1365** | 29 | 2 | 24 | 2 | 3 | 1 |
| **980** | 31 | 2 | 2 | 2 | 1 | 2 |
| **1036** | 31 | 2 | 2 | 1 | 2 | 3 |
| **69** | 36 | 1 | 9 | 3 | 1 | 3 |
| **777** | 21 | 1 | 10 | 2 | 1 | 1 |
| **1162** | 35 | 1 | 10 | 3 | 1 | 10 |
| **761** | 36 | 1 | 15 | 0 | 2 | 4 |
| **1085** | 31 | 2 | 3 | 3 | 2 | 4 |
| **1395** | 31 | 2 | 26 | 0 | 2 | 5 |
| **1452** | 50 | 2 | 1 | 1 | 2 | 6 |
| **928** | 44 | 1 | 15 | 0 | 2 | 7 |
| **102** | 20 | 2 | 6 | 3 | 1 | 2 |
| **1031** | 46 | 1 | 9 | 0 | 2 | 10 |
| **693** | 36 | 1 | 3 | 2 | 1 | 10 |
| **662** | 20 | 1 | 2 | 2 | 1 | 2 |
| **656** | 32 | 1 | 25 | 0 | 2 | 2 |

| | Age | BusTravLevel | DistanceFromHome | EnvironmentSatisfaction | JobInvolvement | MonthlyInc |
|---|---|---|---|---|---|---|
| **939** | 32 | 1 | 7 | 3 | 2 | 4 |