Decision Tree: Income Prediction

Understanding and Cleaning the Data

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [3]:

```
df = pd.read_csv('E:/301/tree models/adult_dataset.csv')
df.head()
```

Out[3]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-family	White	Female	0	4356
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0	4356
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	Black	Female	0	4356
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	0	3900
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	0	3900

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             32561 non-null  int64 
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64 
 3   education       32561 non-null  object
 4   education.num   32561 non-null  int64 
 5   marital.status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital.gain    32561 non-null  int64 
 11  capital.loss    32561 non-null  int64 
 12  hours.per.week  32561 non-null  int64 
 13  native.country  32561 non-null  object
 14  income          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

In [5]:

```
# rows with missing values represented as'?'.
df_1 = df[df.workclass == '?']
df_1
```

Out[5]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capit
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-family	White	Female	0	
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	Black	Female	0	
14	51	?	172175	Doctorate	16	Never-married	?	Not-in-family	White	Male	0	
24	61	?	135285	HS-grad	9	Married-civ-spouse	?	Husband	White	Male	0	
44	71	?	100820	HS-grad	9	Married-civ-spouse	?	Husband	White	Male	0	
...	...	...	...	...	...	...	...	...	...	...	...	
32533	35	?	320084	Bachelors	13	Married-civ-spouse	?	Wife	White	Female	0	
32534	30	?	33811	Bachelors	13	Never-married	?	Not-in-family	Asian-Pac-Islander	Female	0	
32541	71	?	287372	Doctorate	16	Married-civ-spouse	?	Husband	White	Male	0	
32543	41	?	202822	HS-grad	9	Separated	?	Not-in-family	Black	Female	0	
32544	72	?	129912	HS-grad	9	Married-civ-spouse	?	Husband	White	Male	0	

1836 rows × 15 columns

In [6]:

```
df_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1836 entries, 0 to 32544
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             1836 non-null   int64 
 1   workclass       1836 non-null   object
 2   fnlwgt          1836 non-null   int64 
 3   education       1836 non-null   object
 4   education.num   1836 non-null   int64 
 5   marital.status  1836 non-null   object
 6   occupation      1836 non-null   object
 7   relationship    1836 non-null   object
 8   race            1836 non-null   object
 9   sex             1836 non-null   object
 10  capital.gain    1836 non-null   int64 
 11  capital.loss    1836 non-null   int64 
 12  hours.per.week  1836 non-null   int64 
 13  native.country  1836 non-null   object
 14  income          1836 non-null   object
dtypes: int64(6), object(9)
memory usage: 229.5+ KB
```

In [7]:

```
# dropping the rows having missing values in workclass
df = df[df['workclass'] != '?']
df.head()
```

Out[7]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0	4356
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	0	3900
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	0	3900
5	34	Private	216864	HS-grad	9	Divorced	Other-service	Unmarried	White	Female	0	3770
6	38	Private	150601	10th	6	Separated	Adm-clerical	Unmarried	White	Male	0	3770

In [8]:

```
# select all categorical variables
df_categorical = df.select_dtypes(include=['object'])

# checking whether any other columns contain a "?"
df_categorical.apply(lambda x: x=="?", axis=0).sum()
```

Out[8]:

```
workclass           0
education           0
marital.status      0
occupation          7
relationship        0
race                0
sex                 0
native.country    556
income              0
dtype: int64
```

In [9]:

```
# dropping the "?"s
df = df[df['occupation'] != '?']
df = df[df['native.country'] != '?']
```

In [10]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30162 entries, 1 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             30162 non-null  int64 
 1   workclass       30162 non-null  object
 2   fnlwgt          30162 non-null  int64 
 3   education       30162 non-null  object
 4   education.num   30162 non-null  int64 
 5   marital.status  30162 non-null  object
 6   occupation      30162 non-null  object
 7   relationship    30162 non-null  object
 8   race            30162 non-null  object
 9   sex             30162 non-null  object
 10  capital.gain    30162 non-null  int64 
 11  capital.loss    30162 non-null  int64 
 12  hours.per.week  30162 non-null  int64 
 13  native.country  30162 non-null  object
 14  income          30162 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

Data Preparation

In [11]:

```python
from sklearn import preprocessing


# encode categorical variables using Label Encoder

# select all categorical variables
df_categorical = df.select_dtypes(include=['object'])
df_categorical.head()
```

Out[11]:

	workclass	education	marital.status	occupation	relationship	race	sex	native.country	income
1	Private	HS-grad	Widowed	Exec-managerial	Not-in-family	White	Female	United-States	<=50K
3	Private	7th-8th	Divorced	Machine-op-inspct	Unmarried	White	Female	United-States	<=50K
4	Private	Some-college	Separated	Prof-specialty	Own-child	White	Female	United-States	<=50K
5	Private	HS-grad	Divorced	Other-service	Unmarried	White	Female	United-States	<=50K
6	Private	10th	Separated	Adm-clerical	Unmarried	White	Male	United-States	<=50K

In [12]:

```python
# apply Label encoder to df_categorical

le = preprocessing.LabelEncoder()
df_categorical = df_categorical.apply(le.fit_transform)
df_categorical.head()
```

Out[12]:

	workclass	education	marital.status	occupation	relationship	race	sex	native.country	income
1	2	11	6	3	1	4	0	38	0
3	2	5	0	6	4	4	0	38	0
4	2	15	5	9	3	4	0	38	0
5	2	11	0	7	4	4	0	38	0
6	2	0	5	0	4	4	1	38	0

In [13]:

```python
# concat df_categorical with original df
df = df.drop(df_categorical.columns, axis=1)
df = pd.concat([df, df_categorical], axis=1)
df.head()
```

Out[13]:

	age	fnlwgt	education.num	capital.gain	capital.loss	hours.per.week	workclass	education	marital.status	occupation	relationship
1	82	132870	9	0	4356	18	2	11	6	3	1
3	54	140359	4	0	3900	40	2	5	0	6	4
4	41	264663	10	0	3900	40	2	15	5	9	3
5	34	216864	9	0	3770	45	2	11	0	7	4
6	38	150601	6	0	3770	40	2	0	5	0	4

In [14]:

```python
# look at column types
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30162 entries, 1 to 32560
Data columns (total 15 columns):
```

```
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             30162 non-null  int64
 1   fnlwgt          30162 non-null  int64
 2   education.num   30162 non-null  int64
 3   capital.gain    30162 non-null  int64
 4   capital.loss    30162 non-null  int64
 5   hours.per.week  30162 non-null  int64
 6   workclass       30162 non-null  int32
 7   education       30162 non-null  int32
 8   marital.status  30162 non-null  int32
 9   occupation      30162 non-null  int32
 10  relationship    30162 non-null  int32
 11  race            30162 non-null  int32
 12  sex             30162 non-null  int32
 13  native.country  30162 non-null  int32
 14  income          30162 non-null  int32
dtypes: int32(9), int64(6)
memory usage: 2.6 MB
```

In [15]:

```
# convert target variable income to categorical
df['income'] = df['income'].astype('category')
```

Model Bulding and Evaluation

In [16]:

```
# Importing train-test-split
from sklearn.model_selection import train_test_split
```

In [17]:

```
# Putting feature variable to X
X = df.drop('income',axis=1)

# Putting response variable to y
y = df['income']
```

In [18]:

```
# Splitting the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.30,
                                                    random_state = 99)
X_train.head()
```

Out[18]:

	age	fnlwgt	education.num	capital.gain	capital.loss	hours.per.week	workclass	education	marital.status	occupation	relatior
24351	42	289636	9	0	0	46	2	11	2	13	
15626	37	52465	9	0	0	40	1	11	4	7	
4347	38	125933	14	0	0	40	0	12	2	9	
23972	44	183829	13	0	0	38	5	9	4	0	
26843	35	198841	11	0	0	35	2	8	0	12	

In [19]:

```
# Importing decision tree classifier from sklearn library
from sklearn.tree import DecisionTreeClassifier

# Fitting the decision tree with default hyperparameters, apart from
# max_depth which is 5 so that we can plot and read the tree.
dt_default = DecisionTreeClassifier(max_depth=5)
dt_default.fit(X_train, y_train)
```

Out[19]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=5, 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='deprecated',
                       random_state=None, splitter='best')
```

In [20]:

```
# Let's check the evaluation metrics of our default model

# Importing classification report and confusion matrix from sklearn metrics
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Making predictions
y_pred_default = dt_default.predict(X_test)

# Printing classification report
print(classification_report(y_test, y_pred_default))
```

```
              precision    recall  f1-score   support

           0       0.86      0.95      0.91      6867
           1       0.78      0.52      0.63      2182

    accuracy                           0.85      9049
   macro avg       0.82      0.74      0.77      9049
weighted avg       0.84      0.85      0.84      9049
```

In [21]:

```
# Printing confusion matrix and accuracy
print(confusion_matrix(y_test,y_pred_default))
print(accuracy_score(y_test,y_pred_default))
```

```
[[6553  314]
 [1038 1144]]
0.8505912255497845
```

Plotting the Decision Tree

In [23]:

```
# Importing required packages for visualization
from IPython.display import Image
from sklearn.externals.six import StringIO
from sklearn.tree import export_graphviz


# Putting features
features = list(df.columns[1:])
features
```

Out[23]:

```
['fnlwgt',
 'education.num',
 'capital.gain',
 'capital.loss',
 'hours.per.week',
 'workclass',
 'education',
 'marital.status',
 'occupation',
 'relationship',
 'race',
 'sex',
```

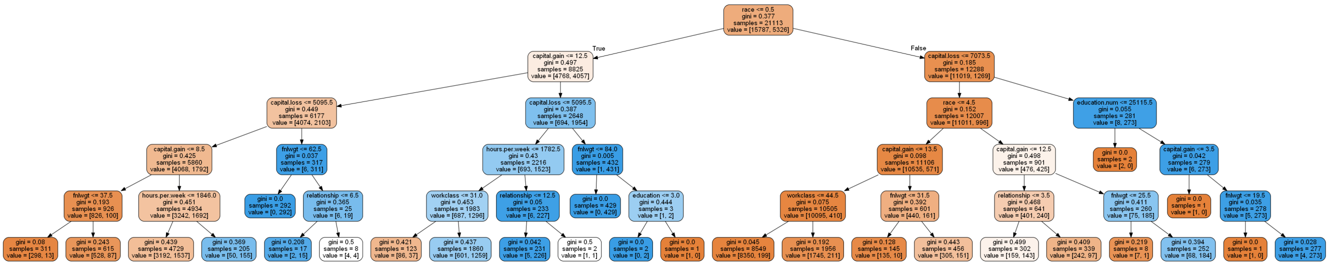
```
 'native.country',
 'income']
```

In [24]:

```
# plotting tree with max_depth=3
dot_data = StringIO()
export_graphviz(dt_default, out_file=dot_data,
                feature_names=features, filled=True,rounded=True)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[24]:



Hyperparameter Tuning

In [25]:

```
# GridSearchCV to find optimal max_depth
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV


# specify number of folds for k-fold CV
n_folds = 5

# parameters to build the model on
parameters = {'max_depth': range(1, 40)}

# instantiate the model
dtree = DecisionTreeClassifier(criterion = "gini",
                               random_state = 100)

# fit tree on training data
tree = GridSearchCV(dtree, parameters,
                    cv=n_folds,
                   scoring="accuracy")
tree.fit(X_train, y_train)
```

Out[25]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              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='deprecated',
                                              random_state=100,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': range(1, 40)}, pre_dispatch='2*n_jobs',
             refit=True, return_train_score=False, scoring='accuracy',
             verbose=0)
```

In [26]:

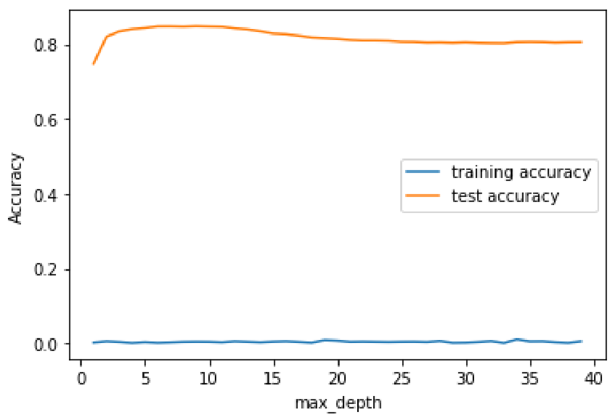
```
# scores of GridSearch CV
scores = tree.cv_results_
pd.DataFrame(scores).head()
```

Out[26]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	params	split0_test_score	split1_test_score
0	0.028370	0.001187	0.003069	0.002677	1	{'max_depth': 1}	0.747810	0.747810
1	0.036536	0.006332	0.006541	0.008023	2	{'max_depth': 2}	0.812219	0.818612
2	0.050151	0.008720	0.004684	0.006249	3	{'max_depth': 3}	0.828558	0.834241
3	0.062308	0.003408	0.002005	0.002325	4	{'max_depth': 4}	0.832583	0.840871
4	0.076496	0.003978	0.004183	0.002502	5	{'max_depth': 5}	0.834241	0.844897

In [28]:

```
# plotting accuracies with max_depth
plt.figure()
plt.plot(scores["param_max_depth"],
         scores["mean_score_time"],
         label="training accuracy")
plt.plot(scores["param_max_depth"],
         scores["mean_test_score"],
         label="test accuracy")
plt.xlabel("max_depth")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



In [29]:

```
# GridSearchCV to find optimal max_depth
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV


# specify number of folds for k-fold CV
n_folds = 5

# parameters to build the model on
parameters = {'min_samples_leaf': range(5, 200, 20)}

# instantiate the model
dtree = DecisionTreeClassifier(criterion = "gini",
                               random_state = 100)

# fit tree on training data
tree = GridSearchCV(dtree, parameters,
                    cv=n_folds,
                    scoring="accuracy")
```

```
tree.fit(X_train, y_train)
```

Out[29]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              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='deprecated',
                                              random_state=100,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'min_samples_leaf': range(5, 200, 20)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

In [30]:

```
# scores of GridSearch CV
scores = tree.cv_results_
pd.DataFrame(scores).head()
```

Out[30]:

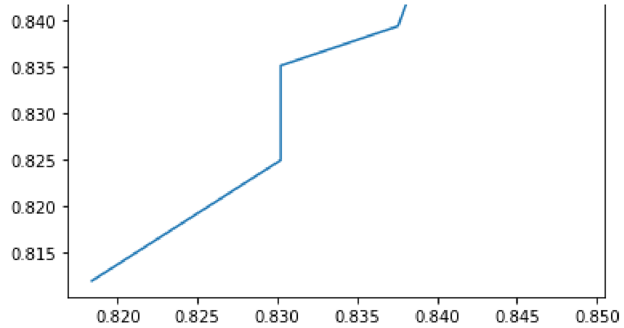
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_leaf	params	split0_test_score	split1_
0	0.158142	0.005706	0.003644	0.006283	5	{'min_samples_leaf': 5}	0.825716	
1	0.146483	0.014285	0.007195	0.003729	25	{'min_samples_leaf': 25}	0.841819	
2	0.148788	0.032665	0.003109	0.002285	45	{'min_samples_leaf': 45}	0.843003	
3	0.126007	0.005495	0.004272	0.002220	65	{'min_samples_leaf': 65}	0.841108	
4	0.121249	0.005447	0.004260	0.001736	85	{'min_samples_leaf': 85}	0.838030	

In [39]:

```
# plotting accuracies with min_samples_leaf
plt.figure()
plt.plot(scores["split2_test_score"],
         scores["split0_test_score"],
         label="training accuracy")
plt.plot(scores["param_min_samples_leaf"],
         scores["mean_test_score"],
         label="test accuracy")
plt.xlabel("min_samples_leaf")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-39-3fba0aa9fd42> in <module>
      4          scores["split0_test_score"],
      5          label="training accuracy")
----> 6 plt.plot(scores["param_min_samples_leaf"],
      7          scores["mean_test_score"],
      8          label="test accuracy")

KeyError: 'param_min_samples_leaf'
```



In [35]:

```
# GridSearchCV to find optimal min_samples_split
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV


# specify number of folds for k-fold CV
n_folds = 5

# parameters to build the model on
parameters = {'min_samples_split': range(5, 200, 20)}

# instantiate the model
dtree = DecisionTreeClassifier(criterion = "gini",
                               random_state = 100)

# fit tree on training data
tree = GridSearchCV(dtree, parameters,
                    cv=n_folds,
                   scoring="accuracy")
tree.fit(X_train, y_train)
```

Out[35]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              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='deprecated',
                                              random_state=100,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'min_samples_split': range(5, 200, 20)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

In [36]:

```
# scores of GridSearch CV
scores = tree.cv_results_
pd.DataFrame(scores).head()
```

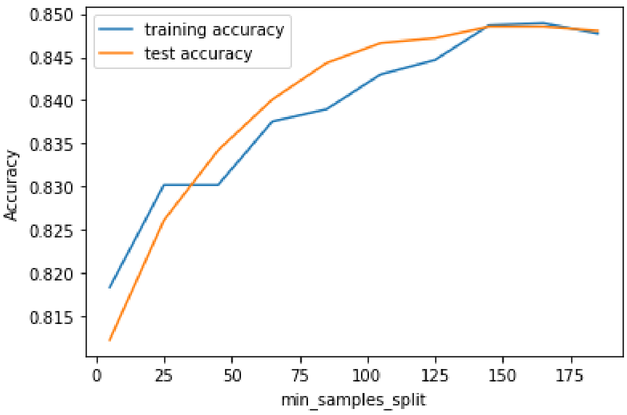
Out[36]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_split	params	split0_test_score	split1
0	0.185636	0.016668	0.004584	0.006362	5	{'min_samples_split': 5}	0.811982	
1	0.160407	0.010761	0.005313	0.006492	25	{'min_samples_split': 25}	0.825006	
2	0.183473	0.027249	0.002950	0.002643	45	{'min_samples_split': 45}	0.835188	
3	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	{'min_samples_split':	[illegible]	

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_split	params	split0_test_score	split1
3	0.173695	0.009600	0.004880	0.002943	65	65}	0.839451	
4	0.191491	0.051685	0.004084	0.001937	85	{'min_samples_split': 85}	0.846081	

In [40]:

```
# plotting accuracies with min_samples_leaf
plt.figure()
plt.plot(scores["param_min_samples_split"],
         scores["split2_test_score"],
         label="training accuracy")
plt.plot(scores["param_min_samples_split"],
         scores["mean_test_score"],
         label="test accuracy")
plt.xlabel("min_samples_split")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Grid Search to Find Optimal Hyperparameters

In [41]:

```
# Create the parameter grid
param_grid = {
    'max_depth': range(5, 15, 5),
    'min_samples_leaf': range(50, 150, 50),
    'min_samples_split': range(50, 150, 50),
    'criterion': ["entropy", "gini"]
}

n_folds = 5

# Instantiate the grid search model
dtree = DecisionTreeClassifier()
grid_search = GridSearchCV(estimator = dtree, param_grid = param_grid,
                          cv = n_folds, verbose = 1)

# Fit the grid search to the data
grid_search.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  80 out of  80 | elapsed:    8.9s finished
```

Out[41]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              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='deprecated',
                                                    random_state=None,
                                                    splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'criterion': ['entropy', 'gini'],
                         'max_depth': range(5, 15, 5),
                         'min_samples_leaf': range(50, 150, 50),
                         'min_samples_split': range(50, 150, 50)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=1)
```

In [42]:

```
# cv results
cv_results = pd.DataFrame(grid_search.cv_results_)
cv_results
```

Out[42]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param_max_depth	param_min_samples_leaf	param_
0	0.084381	0.008839	0.003411	0.002675	entropy	5	50	
1	0.079006	0.009346	0.000000	0.000000	entropy	5	50	
2	0.086810	0.005144	0.001530	0.001354	entropy	5	100	
3	0.092239	0.005534	0.004738	0.002507	entropy	5	100	
4	0.151544	0.028606	0.002993	0.002158	entropy	10	50	
5	0.141180	0.006196	0.004137	0.002705	entropy	10	50	
6	0.144369	0.009476	0.005005	0.002937	entropy	10	100	
7	0.137399	0.011951	0.004676	0.001611	entropy	10	100	
8	0.073815	0.007633	0.006890	0.003323	gini	5	50	
9	0.072070	0.006772	0.012776	0.006500	gini	5	50	
10	0.080588	0.004658	0.000693	0.001386	gini	5	100	
11	0.082917	0.009548	0.001523	0.001721	gini	5	100	

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param_max_depth	param_min_samples_leaf	param_
12	0.129256	0.001851	0.001307	0.002614	gini	10	50	
13	0.127499	0.005397	0.005323	0.005784	gini	10	50	
14	0.123655	0.014355	0.005215	0.006601	gini	10	100	
15	0.116437	0.008007	0.003818	0.007635	gini	10	100	

In [43]:

```
# printing the optimal accuracy score and hyperparameters
print("best accuracy", grid_search.best_score_)
print(grid_search.best_estimator_)
```

```
best accuracy 0.8510400232064759
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=10, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=50, min_samples_split=50,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

Running the model with best parameters obtained from grid search

In [44]:

```
# model with optimal hyperparameters
clf_gini = DecisionTreeClassifier(criterion = "gini",
                                  random_state = 100,
                                  max_depth=10,
                                  min_samples_leaf=50,
                                  min_samples_split=50)
clf_gini.fit(X_train, y_train)
```

Out[44]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=10, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=50, min_samples_split=50,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=100, splitter='best')
```

In [45]:

```
# accuracy score
clf_gini.score(X_test,y_test)
```

Out[45]:

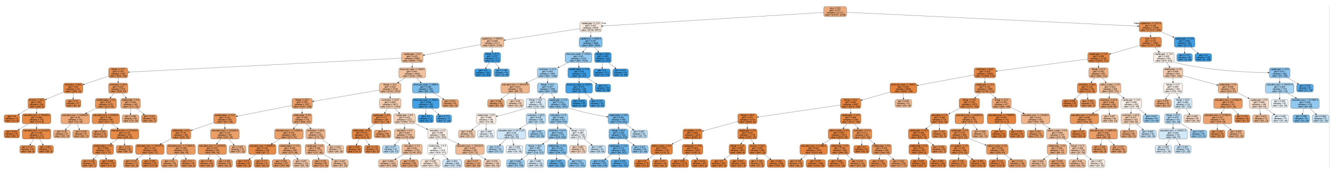
```
0.850922753895458
```

In [46]:

```
# plotting the tree
dot_data = StringIO()
export_graphviz(clf_gini, out_file=dot_data,feature_names=features,filled=True,rounded=True)
```

```
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[46]:



In [47]:

```
# tree with max_depth = 3
clf_gini = DecisionTreeClassifier(criterion = "gini",
                                  random_state = 100,
                                  max_depth=3,
                                  min_samples_leaf=50,
                                  min_samples_split=50)
clf_gini.fit(X_train, y_train)

# score
print(clf_gini.score(X_test,y_test))
```

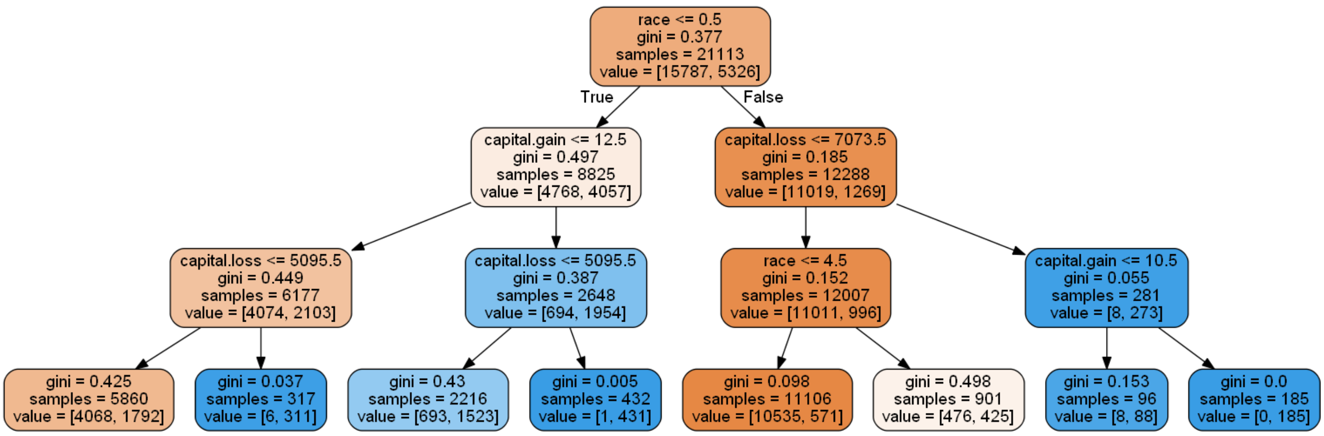
```
0.8393192617968837
```

In [48]:

```
# plotting tree with max_depth=3
dot_data = StringIO()
export_graphviz(clf_gini, out_file=dot_data,feature_names=features,filled=True,rounded=True)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[48]:



In [49]:

```
# classification metrics
from sklearn.metrics import classification_report,confusion_matrix
y_pred = clf_gini.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.96      0.90      6867
           1       0.77      0.47      0.59      2182

    accuracy                           0.84      9049
   macro avg       0.81      0.71      0.74      9049
weighted avg       0.83      0.84      0.82      9049
```

In [50]:

```
# confusion matrix
print(confusion_matrix(y_test,y_pred))
```

```
[[6564  303]
 [1151 1031]]
```

In [ ]:

```
```