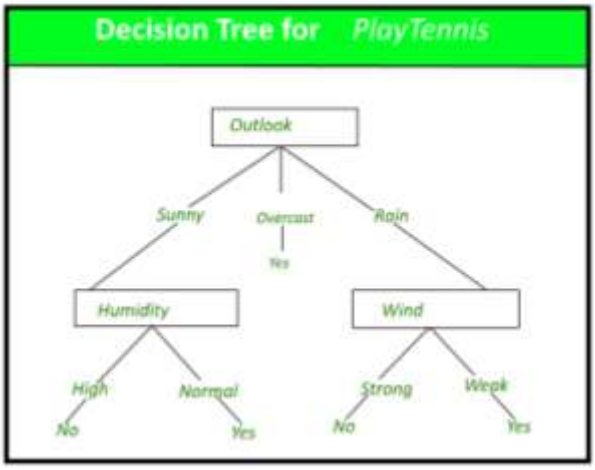
Experiment No. 3
Apply Decision Tree Algorithm on Adult Census Income Dataset and analyze the performance of the model
Date of Performance: 7-08-2023
Date of Submission: 20-08-2023

Aim: Apply Decision Tree Algorithm on Adult Census Income Dataset and analyze the performance of the model.

Objective: To perform various feature engineering tasks, apply Decision Tree Algorithm on the given dataset and maximize the accuracy, Precision, Recall, F1 score. Improve the performance by performing different data engineering and feature engineering tasks.

Theory: Decision Tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart-like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.



Dataset:

Predict whether income exceeds $50K/yr based on census data. Also known as "Adult" dataset.

Attribute Information:

Listing of attributes:

>50K, <=50K.

age: continuous.

workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.

fnlwgt: continuous.

education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.

education-num: continuous.

marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.

occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.

relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.

race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.

sex: Female, Male.

capital-gain: continuous.

capital-loss: continuous.

hours-per-week: continuous.

native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.

In [1]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

In [1]:
```
adult_dataset_path = "adult.csv"

def load_adult_data(adult_path=adult_dataset_path):
    csv_path = os.path.join(adult_path)
    return pd.read_csv(csv_path)
```

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

Out[3]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-family	W
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	W
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	B

In [4]:
```
print ("Rows     : " ,df.shape[0])
print ("Columns  : " ,df.shape[1])
print ("\nFeatures : \n" ,df.columns.tolist())
print ("\nMissing values :  ", df.isnull().sum().values.sum())
print ("\nUnique values :  \n",df.nunique())
```

```
Rows     :  32561
Columns  :  15

Features :
 ['age', 'workclass', 'fnlwgt', 'education', 'education.num', 'marital.statu
s', 'occupation', 'relationship', 'race', 'sex', 'capital.gain', 'capital.lo
ss', 'hours.per.week', 'native.country', 'income']

Missing values :   0

Unique values :
 age                73
workclass            9
fnlwgt           21648
education           16
education.num       16
marital.status       7
occupation          15
relationship         6
race                 5
sex                  2
capital.gain       119
capital.loss        92
hours.per.week      94
native.country      42
income               2
dtype: int64
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
age               32561 non-null int64
workclass         32561 non-null object
fnlwgt            32561 non-null int64
education         32561 non-null object
education.num     32561 non-null int64
marital.status    32561 non-null object
occupation        32561 non-null object
relationship      32561 non-null object
race              32561 non-null object
sex               32561 non-null object
capital.gain      32561 non-null int64
capital.loss      32561 non-null int64
hours.per.week    32561 non-null int64
native.country    32561 non-null object
income            32561 non-null object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

In [6]:
```
df.describe()
```

Out[6]:

	age	fnlwgt	education.num	capital.gain	capital.loss	hours.per.week
count	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.000000	32561.000000
mean	38.581647	1.897784e+05	10.080679	1077.648844	87.303830	40.437456
std	13.640433	1.055500e+05	2.572720	7385.292085	402.960219	12.347429
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.178270e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.783560e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.370510e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000

In [7]:
```
df.head()
```

Out[7]:

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

In [8]:
```
df_check_missing_workclass = (df['workclass']=='?').sum()
df_check_missing_workclass
```

Out[8]:
```
1836
```

In [9]:
```
df_check_missing_occupation = (df['occupation']=='?').sum()
df_check_missing_occupation
```

Out[9]:
```
1843
```

In [10]:
```
df_missing = (df=='?').sum()
df_missing
```

Out[10]:
```
age                0
workclass       1836
fnlwgt             0
education          0
education.num      0
marital.status     0
occupation      1843
relationship       0
race               0
sex                0
capital.gain       0
capital.loss       0
hours.per.week     0
native.country   583
income             0
dtype: int64
```

In [11]:
```
percent_missing = (df=='?').sum() * 100/len(df)
percent_missing
```

Out[11]:
```
age              0.000000
workclass        5.638647
fnlwgt           0.000000
education        0.000000
education.num    0.000000
marital.status   0.000000
occupation       5.660146
relationship     0.000000
race             0.000000
sex              0.000000
capital.gain     0.000000
capital.loss     0.000000
hours.per.week   0.000000
native.country   1.790486
income           0.000000
dtype: float64
```

In [12]:
```
df.apply(lambda x: x !='?',axis=1).sum()
```

Out[12]:
```
age               32561
workclass         30725
fnlwgt            32561
education         32561
education.num     32561
marital.status    32561
occupation        30718
relationship      32561
race              32561
sex               32561
capital.gain      32561
capital.loss      32561
hours.per.week    32561
native.country    31978
income            32561
dtype: int64
```

In [13]:
```
df = df[df['workclass'] !='?']
df.head()
```

Out[13]:

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

In [14]:
```
df_categorical = df.select_dtypes(include=['object'])
df_categorical.apply(lambda x: x=='?',axis=1).sum()
```

Out[14]:
```
workclass          0
education          0
marital.status     0
occupation         7
relationship       0
race               0
sex                0
native.country   556
income             0
dtype: int64
```

In [15]:
```
df = df[df['occupation'] !='?']
df = df[df['native.country'] !='?']
```

In [16]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30162 entries, 1 to 32560
Data columns (total 15 columns):
age               30162 non-null int64
workclass         30162 non-null object
fnlwgt            30162 non-null int64
education         30162 non-null object
education.num     30162 non-null int64
marital.status    30162 non-null object
occupation        30162 non-null object
relationship      30162 non-null object
race              30162 non-null object
sex               30162 non-null object
capital.gain      30162 non-null int64
capital.loss      30162 non-null int64
hours.per.week    30162 non-null int64
native.country    30162 non-null object
income            30162 non-null object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

In [17]:
```
from sklearn import preprocessing
df_categorical = df.select_dtypes(include=['object'])
df_categorical.head()
```

Out[17]:

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

In [18]:
```
le = preprocessing.LabelEncoder()
df_categorical = df_categorical.apply(le.fit_transform)
df_categorical.head()
```

Out[18]:

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

In [19]:
```
df = df.drop(df_categorical.columns,axis=1)
df = pd.concat([df,df_categorical],axis=1)
df.head()
```

Out[19]:

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

In [20]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30162 entries, 1 to 32560
Data columns (total 15 columns):
age               30162 non-null int64
fnlwgt            30162 non-null int64
education.num     30162 non-null int64
capital.gain      30162 non-null int64
capital.loss      30162 non-null int64
hours.per.week    30162 non-null int64
workclass         30162 non-null int64
education         30162 non-null int64
marital.status    30162 non-null int64
occupation        30162 non-null int64
relationship      30162 non-null int64
race              30162 non-null int64
sex               30162 non-null int64
native.country    30162 non-null int64
income            30162 non-null int64
dtypes: int64(15)
memory usage: 3.7 MB
```

In [21]:
```
df['income'] = df['income'].astype('category')
```

In [22]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30162 entries, 1 to 32560
Data columns (total 15 columns):
age               30162 non-null int64
fnlwgt            30162 non-null int64
education.num     30162 non-null int64
capital.gain      30162 non-null int64
capital.loss      30162 non-null int64
hours.per.week    30162 non-null int64
workclass         30162 non-null int64
education         30162 non-null int64
marital.status    30162 non-null int64
occupation        30162 non-null int64
relationship      30162 non-null int64
race              30162 non-null int64
sex               30162 non-null int64
native.country    30162 non-null int64
income            30162 non-null category
dtypes: category(1), int64(14)
memory usage: 3.5 MB
```

In [23]:
```
from sklearn.model_selection import train_test_split
```

In [24]:
```
X = df.drop('income',axis=1)
y = df['income']
```

In [25]:
```
X.head(3)
```

Out[25]:

	age	fnlwgt	education.num	capital.gain	capital.loss	hours.per.week	workclass	education
1	82	132870	9	0	4356	18	2	11
3	54	140359	4	0	3900	40	2	5
4	41	264663	10	0	3900	40	2	15

In [26]:
```
y.head(3)
```

Out[26]:
```
1    0
3    0
4    0
Name: income, dtype: category
Categories (2, int64): [0, 1]
```

In [27]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.30,random_sta
X_train.head()
```

Out[27]:

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

In [28]:
```
from sklearn.tree import DecisionTreeClassifier
dt_default = DecisionTreeClassifier(max_depth=5)
dt_default.fit(X_train,y_train)
```

Out[28]:
```
DecisionTreeClassifier(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=False,
                       random_state=None, splitter='best')
```

In [29]:
```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_sc

y_pred_default = dt_default.predict(X_test)
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 [30]:
```
print(confusion_matrix(y_test,y_pred_default))
print(accuracy_score(y_test,y_pred_default))
```

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

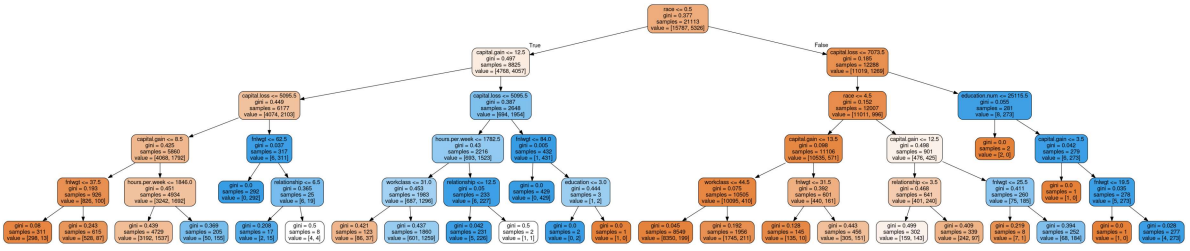
In [32]:
```
from IPython.display import Image
from sklearn.externals.six import StringIO
from sklearn.tree import export_graphviz
import pydotplus,graphviz

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

```
/opt/conda/lib/python3.6/site-packages/sklearn/externals/six.py:31: Deprecat
ionWarning: The module is deprecated in version 0.21 and will be removed in
version 0.23 since we've dropped support for Python 2.7. Please rely on the
official version of six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", DeprecationWarning)
```

Out[32]:
```
['fnlwgt',
 'education.num',
 'capital.gain',
 'capital.loss',
 'hours.per.week',
 'workclass',
 'education',
 'marital.status',
 'occupation',
 'relationship',
 'race',
 'sex',
 'native.country',
 'income']
```

In [34]:
```
# 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[34]:



In [35]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
n_folds = 5
parameters = {'max_depth': range(1, 40)}
dtree = DecisionTreeClassifier(criterion = "gini",
                               random_state = 100)
tree = GridSearchCV(dtree, parameters,
                    cv=n_folds,
                   scoring="accuracy")
tree.fit(X_train, y_train)
```

Out[35]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=DecisionTreeClassifier(class_weight=None,
                                              criterion='gini', max_depth=No
ne,
                                              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=10
0,
                                              splitter='best'),
             iid='warn', n_jobs=None, param_grid={'max_depth': range(1, 4
0)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

In [36]:
```
scores = tree.cv_results_
pd.DataFrame(scores).head()
```

Out[36]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	params
0	0.013396	0.000632	0.002289	0.000077	1	{'max_depth' 1
1	0.020650	0.000721	0.002585	0.000070	2	{'max_depth' 2
2	0.026334	0.000252	0.002494	0.000046	3	{'max_depth' 3
3	0.032663	0.000179	0.002555	0.000051	4	{'max_depth' 4
4	0.038361	0.000191	0.002559	0.000027	5	{'max_depth' 5

In [38]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV

n_folds = 5

parameters = {'min_samples_leaf': range(5, 200, 20)}

dtree = DecisionTreeClassifier(criterion = "gini",
                               random_state = 100)

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

Out[38]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=DecisionTreeClassifier(class_weight=None,
                                              criterion='gini', max_depth=No
ne,
                                              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=10
0,
                                              splitter='best'),
             iid='warn', 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 [39]:
```
scores = tree.cv_results_
pd.DataFrame(scores).head()
```

Out[39]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_leaf	
0	0.089932	0.001484	0.003079	0.000134	5	{'min_
1	0.075429	0.000998	0.002885	0.000082	25	{'min_
2	0.069292	0.002509	0.002822	0.000080	45	{'min_
3	0.064849	0.001231	0.002795	0.000029	65	{'min_
4	0.063291	0.001843	0.002785	0.000034	85	{'min_

In [41]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV

n_folds = 5

parameters = {'min_samples_split': range(5, 200, 20)}

dtree = DecisionTreeClassifier(criterion = "gini",
                               random_state = 100)

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

Out[41]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=DecisionTreeClassifier(class_weight=None,
                                              criterion='gini', max_depth=No
ne,
                                              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=10
0,
                                              splitter='best'),
             iid='warn', 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 [42]:
```
scores = tree.cv_results_
pd.DataFrame(scores).head()
```

Out[42]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_split	
0	0.097296	0.001128	0.003089	0.000046	5	{'min
1	0.092044	0.002086	0.003167	0.000325	25	{'min
2	0.088207	0.001850	0.002975	0.000045	45	{'min
3	0.085938	0.000758	0.003078	0.000290	65	{'min
4	0.084400	0.001154	0.002905	0.000031	85	{'min

In [44]:
```
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

dtree = DecisionTreeClassifier()
grid_search = GridSearchCV(estimator = dtree, param_grid = param_grid,
                           cv = n_folds, verbose = 1)

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 work
ers.
[Parallel(n_jobs=1)]: Done  80 out of  80 | elapsed:    4.6s finished
```

Out[44]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=DecisionTreeClassifier(class_weight=None,
                                              criterion='gini', max_depth=No
ne,
                                              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=No
ne,
                                              splitter='best'),
             iid='warn', 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 [45]:
```
cv_results = pd.DataFrame(grid_search.cv_results_)
cv_results
```

Out[45]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param_max_d
0	0.047459	0.002952	0.002560	0.000043	entropy	
1	0.046124	0.000434	0.002545	0.000037	entropy	
2	0.045717	0.000457	0.002580	0.000074	entropy	
3	0.045961	0.000535	0.002599	0.000055	entropy	
4	0.074791	0.000582	0.002760	0.000028	entropy	
5	0.074224	0.000680	0.002752	0.000041	entropy	
6	0.069409	0.000379	0.002742	0.000031	entropy	
7	0.070990	0.002794	0.002718	0.000024	entropy	
8	0.037624	0.000155	0.002568	0.000051	gini	
9	0.037756	0.000273	0.002660	0.000157	gini	
10	0.037441	0.000350	0.002502	0.000048	gini	
11	0.037432	0.000400	0.002667	0.000394	gini	

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param_max_d
12	0.061380	0.000678	0.002710	0.000048	gini	
13	0.061313	0.000828	0.002725	0.000045	gini	
14	0.058778	0.001751	0.002713	0.000035	gini	
15	0.058679	0.001109	0.002768	0.000033	gini	

In [46]:
```
print("best accuracy", grid_search.best_score_)
print(grid_search.best_estimator_)
```

```
best accuracy 0.8514659214701843
DecisionTreeClassifier(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=False,
                       random_state=None, splitter='best')
```

In [47]:
```
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)
```

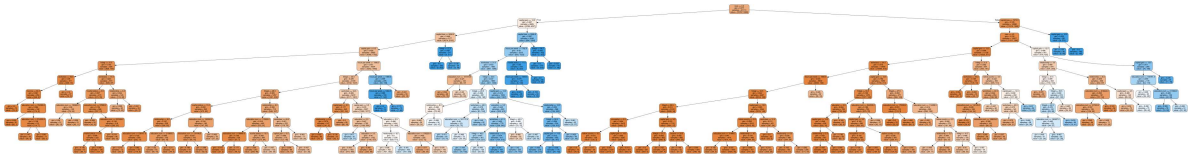
Out[47]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=10,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=50, min_samples_split=50,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=100, splitter='best')
```

In [48]:
```
clf_gini.score(X_test,y_test)
```

Out[48]:
```
0.850922753895458
```

In [49]:
```
dot_data = StringIO()
export_graphviz(clf_gini, out_file=dot_data,feature_names=features,filled=True

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

Out[49]:



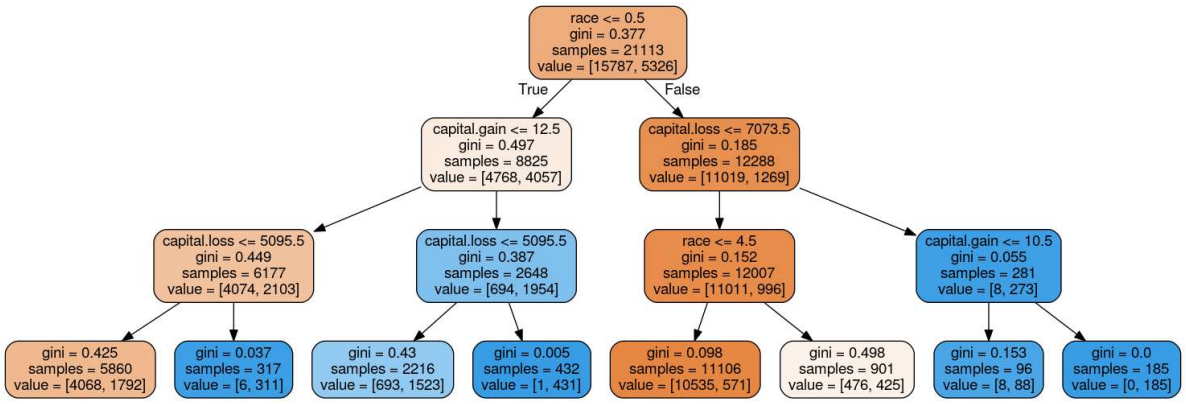
- You can see that this tree is too complex to understand. Let's try reducing the max_depth and see how the tree looks.

In [50]:
```
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)

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

```
0.8393192617968837
```

In [51]:
```
dot_data = StringIO()
export_graphviz(clf_gini, out_file=dot_data,feature_names=features,filled=True

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

Out[51]:



In [52]:
```
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 [53]:
```
print(confusion_matrix(y_test,y_pred))
```

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

Conclusion:

1. The given dataset contained categorical attributes such as workclass, education, marital status, relationship, race, sex, native country and income these categorical attributes haven been dealt using data preprocessing techniques such as label encoding, label encoding is a technique used in machine learning and data analysis to convert categorical variables into numerical format
2. The default tree was quite complex and was simplified by tunning the hyperparameters The following parameters were tuned: -
    a. Tuning max_depth: - The max_depth parameter denotes maximum depth of the tree. It can take any integer value or None. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. By default, it takes "None" value.
    b. Tuning min samples leaf: - The minimum number of samples required to be at a leaf node. If an integer value is taken then consider - -min_samples_leaf as the minimum no. If float, then it shows percentage. By default, it takes "1" value.
    c. Tuning min_samples_spilt: - This tells above the minimum no. of samples reqd. to split an internal node. If an integer value is taken then consider min_samples_split as the minimum no. If float, then it shows percentage. By default, it takes "2" value.
3. The Accuracy score obtained by our decision tree model on the testing data is 0.83 which means our model is 83% accurate on the testing data.
4. Confusion matrix is used to assess the performance of a classification model, in our case the no. of TP is 1031, no. of TN is 6564, no. of FP is 303 and no. of FN are 1151 which means our model is better in predicting negative cases than the positive cases
5. Precision measures the accuracy of the positive predictions and the precision score obtained by our model is 0.85
6. Recall measures the ability of the model to correctly identify all relevant instances and the Recall score obtained by our model is 0.96
7. F1-score is the harmonic mean of precision and recall and provides a balance between the 2 metrics and the F1-score obtained by our model is 0.90