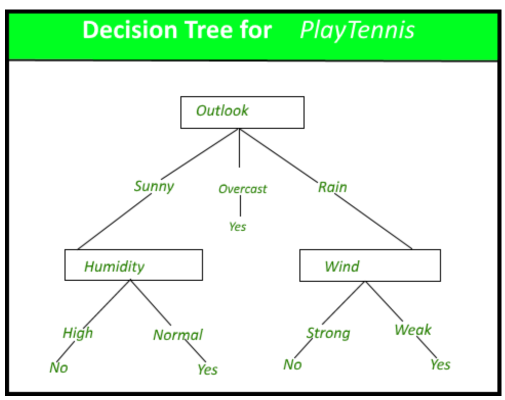
Experiment No. 3
Apply Decision Tree Algorithm on Adult Census Income Dataset and analyze the performance of the model
Date of Performance: 07-08-2023
Date of Submission: 08-10-2023

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.

ml-experiment-3

October 8, 2023

[ ]:
```
# Import libraries
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# To ignore warning messages
import warnings
warnings.filterwarnings('ignore')
```

[ ]:
```
# Adult dataset path
adult_dataset_path = "adult.csv"

# Function for loading adult dataset
def load_adult_data(adult_path=adult_dataset_path):
    csv_path = os.path.join(adult_path)
    return pd.read_csv(csv_path)
```

[ ]:
```
# Calling load adult function and assigning to a new variable df
df = load_adult_data()
# load top 3 rows values from adult dataset
df.head(3)
```

[ ]:
```
   age workclass  fnlwgt     education  education.num marital.status  \
0   90         ?   77053       HS-grad              9        Widowed
1   82   Private  132870       HS-grad              9        Widowed
2   66         ?  186061  Some-college             10        Widowed

        occupation   relationship   race     sex  capital.gain  capital.loss  \
0                ?  Not-in-family  White  Female             0          4356
1  Exec-managerial  Not-in-family  White  Female             0          4356
2                ?      Unmarried  Black  Female             0          4356

   hours.per.week native.country income
0              40  United-States  <=50K
```

```
1               18  United-States   <=50K
2               40  United-States   <=50K
```

[ ]:
```
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.status',
'occupation', 'relationship', 'race', 'sex', 'capital.gain', 'capital.loss',
'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
```

[ ]:
```
# Let's understand the type of values present in each column of our adult
 ↪dataframe 'df'.
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
```

[ ]:
```
# Numerical feature of summary/description
df.describe()
```

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

       hours.per.week
count    32561.000000
mean        40.437456
std         12.347429
min          1.000000
25%         40.000000
50%         40.000000
75%         45.000000
max         99.000000
```

[ ]:
```
# pull top 5 row values to understand the data and how it's look like
df.head()
```

[ ]:
```
   age workclass  fnlwgt     education  education.num marital.status  \
0   90         ?   77053       HS-grad              9        Widowed
1   82   Private  132870       HS-grad              9        Widowed
2   66         ?  186061  Some-college             10        Widowed
```

```
3   54    Private  140359       7th-8th              4       Divorced
4   41    Private  264663  Some-college             10      Separated

           occupation   relationship   race     sex  capital.gain  \
0                   ?  Not-in-family  White  Female             0
1     Exec-managerial  Not-in-family  White  Female             0
2                   ?      Unmarried  Black  Female             0
3   Machine-op-inspct      Unmarried  White  Female             0
4      Prof-specialty      Own-child  White  Female             0

   capital.loss  hours.per.week native.country income
0          4356              40  United-States  <=50K
1          4356              18  United-States  <=50K
2          4356              40  United-States  <=50K
3          3900              40  United-States  <=50K
4          3900              40  United-States  <=50K
```

[ ]:
```python
# checking "?" total values present in particular 'workclass' feature
df_check_missing_workclass = (df['workclass']=='?').sum()
df_check_missing_workclass
```

[ ]:
```
1836
```

[ ]:
```python
# checking "?" total values present in particular 'occupation' feature
df_check_missing_occupation = (df['occupation']=='?').sum()
df_check_missing_occupation
```

[ ]:
```
1843
```

[ ]:
```python
# checking "?" values, how many are there in the whole dataset
df_missing = (df=='?').sum()
df_missing
```

[ ]:
```
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
```

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

[ ]:
```
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
```

[ ]:
```
#Let's find total number of rows which doesn't contain any missing value as '?'
df.apply(lambda x: x !='?',axis=1).sum()
```

[ ]:
```
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
```

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

[ ]:
```
   age workclass  fnlwgt     education  education.num marital.status  \
1   82   Private  132870       HS-grad              9        Widowed
3   54   Private  140359       7th-8th              4       Divorced
4   41   Private  264663  Some-college             10      Separated
5   34   Private  216864       HS-grad              9       Divorced
6   38   Private  150601          10th              6      Separated

         occupation   relationship   race     sex  capital.gain  \
1   Exec-managerial  Not-in-family  White  Female             0
3 Machine-op-inspct      Unmarried  White  Female             0
4    Prof-specialty      Own-child  White  Female             0
5     Other-service      Unmarried  White  Female             0
6      Adm-clerical      Unmarried  White    Male             0

   capital.loss  hours.per.week native.country income
1          4356              18  United-States  <=50K
3          3900              40  United-States  <=50K
4          3900              40  United-States  <=50K
5          3770              45  United-States  <=50K
6          3770              40  United-States  <=50K
```

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

# checking whether any other column contains '?' value
df_categorical.apply(lambda x: x=='?',axis=1).sum()
```

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

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

[ ]:
```
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
```

[ ]:
```
from sklearn import preprocessing

# encode categorical variables using label Encoder

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

[ ]:
```
  workclass     education marital.status         occupation   relationship  \
1   Private       HS-grad        Widowed    Exec-managerial  Not-in-family
3   Private       7th-8th       Divorced  Machine-op-inspct      Unmarried
4   Private  Some-college      Separated     Prof-specialty      Own-child
5   Private       HS-grad       Divorced      Other-service      Unmarried
6   Private          10th      Separated       Adm-clerical      Unmarried

    race     sex native.country income
1  White  Female  United-States  <=50K
3  White  Female  United-States  <=50K
4  White  Female  United-States  <=50K
5  White  Female  United-States  <=50K
6  White    Male  United-States  <=50K
```

[ ]:
```
# apply label encoder to df_categorical
le = preprocessing.LabelEncoder()
df_categorical = df_categorical.apply(le.fit_transform)
df_categorical.head()
```

```
[ ]:    workclass  education  marital.status  occupation  relationship  race  sex  \
     1          2         11               6           3             1     4    0
     3          2          5               0           6             4     4    0
     4          2         15               5           9             3     4    0
     5          2         11               0           7             4     4    0
     6          2          0               5           0             4     4    1

        native.country  income
     1              38       0
     3              38       0
     4              38       0
     5              38       0
     6              38       0
```

```
[ ]: # Next, Concatenate df_categorical dataframe with original df (dataframe)

     # first, Drop earlier duplicate columns which had categorical values
     df = df.drop(df_categorical.columns,axis=1)
     df = pd.concat([df,df_categorical],axis=1)
     df.head()
```

```
[ ]:    age  fnlwgt  education.num  capital.gain  capital.loss  hours.per.week  \
     1   82  132870              9             0          4356              18
     3   54  140359              4             0          3900              40
     4   41  264663             10             0          3900              40
     5   34  216864              9             0          3770              45
     6   38  150601              6             0          3770              40

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

        native.country  income
     1              38       0
     3              38       0
     4              38       0
     5              38       0
     6              38       0
```

```
[ ]: # look at column type
     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  int64
 7   education       30162 non-null  int64
 8   marital.status  30162 non-null  int64
 9   occupation      30162 non-null  int64
 10  relationship    30162 non-null  int64
 11  race            30162 non-null  int64
 12  sex             30162 non-null  int64
 13  native.country  30162 non-null  int64
 14  income          30162 non-null  int64
dtypes: int64(15)
memory usage: 3.7 MB
```

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

[ ]:
```python
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  int64
 7   education       30162 non-null  int64
 8   marital.status  30162 non-null  int64
 9   occupation      30162 non-null  int64
 10  relationship    30162 non-null  int64
 11  race            30162 non-null  int64
 12  sex             30162 non-null  int64
 13  native.country  30162 non-null  int64
 14  income          30162 non-null  category
dtypes: category(1), int64(14)
memory usage: 3.5 MB
```

[ ]:
```
# Importing train_test_split
from sklearn.model_selection import train_test_split
```

[ ]:
```
# Putting independent variables/features to X
X = df.drop('income',axis=1)

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

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

[ ]:
```
   age  fnlwgt  education.num  capital.gain  capital.loss  hours.per.week  \
1   82  132870              9             0          4356              18
3   54  140359              4             0          3900              40
4   41  264663             10             0          3900              40

   workclass  education  marital.status  occupation  relationship  race  sex  \
1          2         11               6           3             1     4    0
3          2          5               0           6             4     4    0
4          2         15               5           9             3     4    0

   native.country
1              38
3              38
4              38
```

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

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

[ ]:
```
# 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()
```

[ ]:
```
       age  fnlwgt  education.num  capital.gain  capital.loss  hours.per.week  \
24351   42  289636              9             0             0              46
15626   37   52465              9             0             0              40
4347    38  125933             14             0             0              40
23972   44  183829             13             0             0              38
26843   35  198841             11             0             0              35
```

```
       workclass  education  marital.status  occupation  relationship  race  \
24351          2         11               2          13             0     4
15626          1         11               4           7             1     4
4347           0         12               2           9             0     4
23972          5          9               4           0             1     4
26843          2          8               0          12             3     4

       sex  native.country
24351    1              38
15626    1              38
4347     1              19
23972    0              38
26843    1              38
```

[ ]:
```
# 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)
```

[ ]:
```
DecisionTreeClassifier(max_depth=5)
```

[ ]:
```
# 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 classifier report after prediction
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
```

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

```
[[6553  314]
 [1039 1143]]
0.8504807161012267
```

[ ]:
```
!pip install pydotplus
```

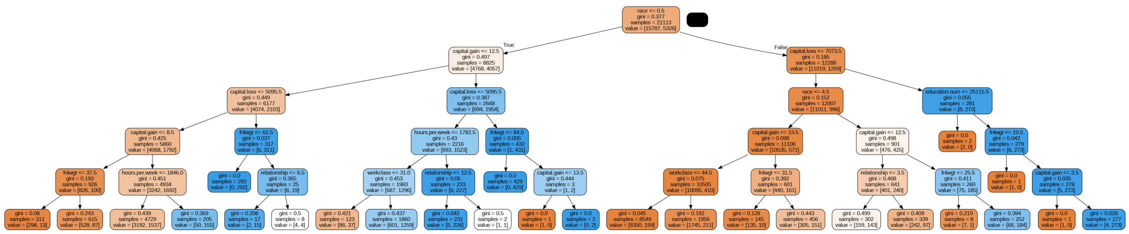
```
Requirement already satisfied: pydotplus in /usr/local/lib/python3.10/dist-
packages (2.0.2)
Requirement already satisfied: pyparsing>=2.0.1 in
/usr/local/lib/python3.10/dist-packages (from pydotplus) (3.1.1)
```

[ ]:
```
# Importing required packages for visualization
from IPython.display import Image
from six import StringIO
from sklearn.tree import export_graphviz
import pydotplus,graphviz

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

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

[ ]:
```
# 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())
```

[ ]:



[ ]:

```python
# 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)
```

[ ]:

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=100),
             param_grid={'max_depth': range(1, 40)}, scoring='accuracy')
```

[ ]:

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

[ ]:

```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0       0.035181      0.011561         0.010106        0.002992
1       0.061280      0.018088         0.013943        0.008235
2       0.074714      0.026539         0.014324        0.013822
3       0.074829      0.025231         0.011173        0.008233
4       0.055147      0.004359         0.007180        0.002890

  param_max_depth             params  split0_test_score  split1_test_score  \
0               1  {'max_depth': 1}           0.747810           0.747810
```

```
1              2  {'max_depth': 2}          0.812219           0.818612
2              3  {'max_depth': 3}          0.828558           0.834241
3              4  {'max_depth': 4}          0.832583           0.840871
4              5  {'max_depth': 5}          0.834241           0.844897

   split2_test_score  split3_test_score  split4_test_score  mean_test_score  \
0           0.747573           0.747750           0.747750         0.747738
1           0.820507           0.825675           0.822833         0.819969
2           0.834478           0.836570           0.837518         0.834273
3           0.842529           0.842729           0.842255         0.840193
4           0.847265           0.842729           0.847466         0.843319

   std_test_score  rank_test_score
0        0.000087               39
1        0.004538               16
2        0.003115               12
3        0.003860                9
4        0.004858                7
```

[ ]:
```
"""
# plotting accuracies with max_depth
plt.figure()
plt.plot(scores["param_max_depth"],
         scores["mean_train_score"],
         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()
"""
```

[ ]:
```
'\n# plotting accuracies with
max_depth\nplt.figure()\nplt.plot(scores["param_max_depth"], \n
scores["mean_train_score"], \n         label="training
accuracy")\nplt.plot(scores["param_max_depth"], \n
scores["mean_test_score"], \n         label="test accuracy")\nplt.xlabel("max_de
pth")\nplt.ylabel("Accuracy")\nplt.legend()\nplt.show()\n'
```

[ ]:
```
# 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
```

```python
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)
```

[ ]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=100),
             param_grid={'min_samples_leaf': range(5, 200, 20)},
             scoring='accuracy')
```

[ ]:
```python
# scores of GridSearch CV
scores = tree.cv_results_
pd.DataFrame(scores).head()
```

[ ]:
```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0       0.083994      0.003596         0.004026        0.000177
1       0.070918      0.003142         0.003753        0.000242
2       0.063995      0.001806         0.003521        0.000141
3       0.061234      0.004158         0.004317        0.000716
4       0.063998      0.009761         0.004074        0.000192

  param_min_samples_leaf                     params  split0_test_score  \
0                      5   {'min_samples_leaf': 5}           0.825716
1                     25  {'min_samples_leaf': 25}           0.841819
2                     45  {'min_samples_leaf': 45}           0.843003
3                     65  {'min_samples_leaf': 65}           0.841108
4                     85  {'min_samples_leaf': 85}           0.838030

   split1_test_score  split2_test_score  split3_test_score  split4_test_score  \
0           0.827848           0.819560           0.826149           0.818806
1           0.851291           0.839451           0.842018           0.849360
2           0.849159           0.846555           0.851018           0.851729
3           0.852711           0.845371           0.851492           0.838465
4           0.849159           0.845371           0.851492           0.842018

   mean_test_score  std_test_score  rank_test_score
0         0.823616        0.003696               10
1         0.844788        0.004651                6
```

```
2          0.848293         0.003194                 1
3          0.845830         0.005589                 2
4          0.845214         0.004834                 3
```

[ ]:
```
"""
# plotting accuracies with min_samples_leaf
plt.figure()
plt.plot(scores["param_min_samples_leaf"],
         scores["mean_train_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()
"""
```

[ ]:
```
'\n# plotting accuracies with
min_samples_leaf\nplt.figure()\nplt.plot(scores["param_min_samples_leaf"], \n
scores["mean_train_score"], \n         label="training
accuracy")\nplt.plot(scores["param_min_samples_leaf"], \n
scores["mean_test_score"], \n         label="test accuracy")\nplt.xlabel("min_sa
mples_leaf")\nplt.ylabel("Accuracy")\nplt.legend()\nplt.show()\n'
```

[ ]:
```
# 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)
```

[ ]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=100),
             param_grid={'min_samples_split': range(5, 200, 20)},
             scoring='accuracy')
```

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

[ ]:
```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0       0.190965      0.053279         0.007489        0.002150
1       0.140156      0.022450         0.006022        0.000463
2       0.129533      0.010133         0.006719        0.000084
3       0.090867      0.022550         0.004043        0.000819
4       0.079408      0.004440         0.003670        0.000101

  param_min_samples_split                       params  split0_test_score  \
0                       5   {'min_samples_split': 5}           0.811982
1                      25  {'min_samples_split': 25}           0.825006
2                      45  {'min_samples_split': 45}           0.835188
3                      65  {'min_samples_split': 65}           0.839451
4                      85  {'min_samples_split': 85}           0.846081

   split1_test_score  split2_test_score  split3_test_score  split4_test_score  \
0           0.811035           0.818376           0.811701           0.808385
1           0.825243           0.830215           0.822596           0.827570
2           0.839687           0.830215           0.827333           0.838702
3           0.845844           0.837556           0.833728           0.843913
4           0.853895           0.838977           0.837281           0.845334

   mean_test_score  std_test_score  rank_test_score
0         0.812296        0.003296               10
1         0.826126        0.002581                9
2         0.834225        0.004783                8
3         0.840098        0.004360                7
4         0.844314        0.005898                6
```

[ ]:
```
"""
# plotting accuracies with min_samples_leaf
plt.figure()
plt.plot(scores["param_min_samples_split"],
         scores["mean_train_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()
"""
```

[ ]: '\n# plotting accuracies with
min_samples_leaf\nplt.figure()\nplt.plot(scores["param_min_samples_split"], \n
scores["mean_train_score"], \n   label="training
accuracy")\nplt.plot(scores["param_min_samples_split"], \n
scores["mean_test_score"], \n   label="test accuracy")\nplt.xlabel("min_sa
mples_split")\nplt.ylabel("Accuracy")\nplt.legend()\nplt.show()\n'

[ ]:
```
# 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
```

[ ]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             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)},
             verbose=1)
```

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

[ ]:
```
    mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0        0.038696      0.001708         0.003295        0.000058
1        0.038543      0.001311         0.003356        0.000141
2        0.037763      0.001160         0.003247        0.000045
3        0.038969      0.002066         0.003636        0.000621
```

```
4        0.060942      0.002272         0.003482        0.000060
5        0.061919      0.001492         0.003888        0.000460
6        0.058421      0.003067         0.003427        0.000110
7        0.057515      0.001752         0.003373        0.000031
8        0.034163      0.000422         0.003496        0.000234
9        0.034110      0.000630         0.003269        0.000118
10       0.034014      0.000566         0.003180        0.000066
11       0.035852      0.002114         0.003935        0.000863
12       0.056857      0.002691         0.004146        0.000793
13       0.055483      0.000874         0.003322        0.000032
14       0.054844      0.005373         0.003306        0.000030
15       0.054681      0.003145         0.003366        0.000040

   param_criterion param_max_depth param_min_samples_leaf  \
0          entropy               5                     50
1          entropy               5                     50
2          entropy               5                    100
3          entropy               5                    100
4          entropy              10                     50
5          entropy              10                     50
6          entropy              10                    100
7          entropy              10                    100
8             gini               5                     50
9             gini               5                     50
10            gini               5                    100
11            gini               5                    100
12            gini              10                     50
13            gini              10                     50
14            gini              10                    100
15            gini              10                    100

   param_min_samples_split                                             params  \
0                       50  {'criterion': 'entropy', 'max_depth': 5, 'min_…
1                      100  {'criterion': 'entropy', 'max_depth': 5, 'min_…
2                       50  {'criterion': 'entropy', 'max_depth': 5, 'min_…
3                      100  {'criterion': 'entropy', 'max_depth': 5, 'min_…
4                       50  {'criterion': 'entropy', 'max_depth': 10, 'min…
5                      100  {'criterion': 'entropy', 'max_depth': 10, 'min…
6                       50  {'criterion': 'entropy', 'max_depth': 10, 'min…
7                      100  {'criterion': 'entropy', 'max_depth': 10, 'min…
8                       50  {'criterion': 'gini', 'max_depth': 5, 'min_sam…
9                      100  {'criterion': 'gini', 'max_depth': 5, 'min_sam…
10                      50  {'criterion': 'gini', 'max_depth': 5, 'min_sam…
11                     100  {'criterion': 'gini', 'max_depth': 5, 'min_sam…
12                      50  {'criterion': 'gini', 'max_depth': 10, 'min_sa…
13                     100  {'criterion': 'gini', 'max_depth': 10, 'min_sa…
14                      50  {'criterion': 'gini', 'max_depth': 10, 'min_sa…
```

```
15                  100  {'criterion': 'gini', 'max_depth': 10, 'min_sa…

   split0_test_score  split1_test_score  split2_test_score  \
0           0.834241           0.843950           0.840398
1           0.834241           0.843950           0.840398
2           0.834241           0.842529           0.840398
3           0.834241           0.842529           0.840398
4           0.842529           0.851527           0.847265
5           0.842529           0.851527           0.847265
6           0.845134           0.852475           0.847502
7           0.845134           0.852475           0.847502
8           0.834241           0.844897           0.847502
9           0.834241           0.844897           0.847502
10          0.834241           0.843476           0.844897
11          0.834241           0.843476           0.844897
12          0.843950           0.851291           0.849870
13          0.843950           0.851291           0.849870
14          0.836372           0.848449           0.843239
15          0.836372           0.848449           0.843239

   split3_test_score  split4_test_score  mean_test_score  std_test_score  \
0           0.845097           0.845334         0.841804        0.004173
1           0.845097           0.845334         0.841804        0.004173
2           0.845097           0.845808         0.841615        0.004157
3           0.845097           0.845808         0.841615        0.004157
4           0.854334           0.853861         0.849903        0.004456
5           0.854334           0.853861         0.849903        0.004456
6           0.854098           0.845571         0.848956        0.003661
7           0.854098           0.845571         0.848956        0.003661
8           0.845097           0.847466         0.843841        0.004927
9           0.845097           0.847466         0.843841        0.004927
10          0.845097           0.845334         0.842609        0.004234
11          0.845097           0.845334         0.842609        0.004234
12          0.855045           0.855045         0.851040        0.004093
13          0.855045           0.855045         0.851040        0.004093
14          0.854098           0.846518         0.845735        0.005862
15          0.854098           0.846518         0.845735        0.005862

   rank_test_score
0               13
1               13
2               15
3               15
4                3
5                3
6                5
7                5
```

```
8                 9
9                 9
10               11
11               11
12                1
13                1
14                7
15                7
```

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

```
best accuracy 0.8510400232064759
DecisionTreeClassifier(max_depth=10, min_samples_leaf=50, min_samples_split=50)
```

[ ]:
```
# 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)
```

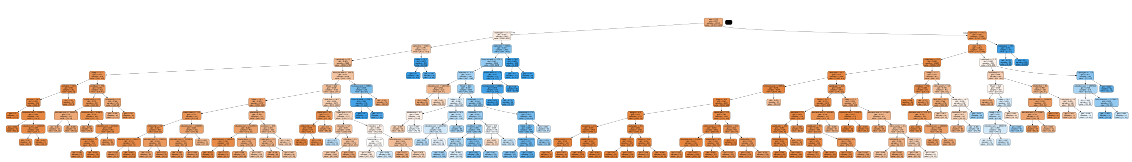
[ ]:
```
DecisionTreeClassifier(max_depth=10, min_samples_leaf=50, min_samples_split=50,
                       random_state=100)
```

[ ]:
```
# accuracy score
clf_gini.score(X_test,y_test)
```

[ ]:
```
0.850922753895458
```

[ ]:
```
# 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())
```

[ ]:



[ ]:

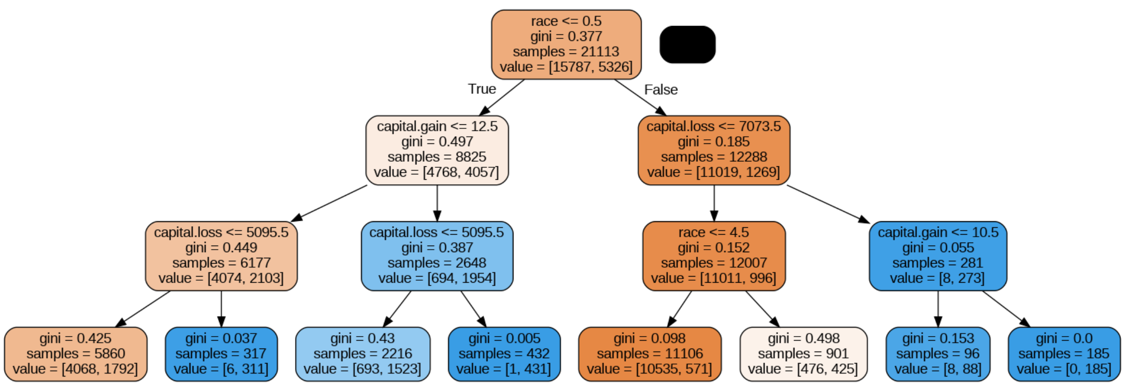
```python
# tree with max_depth = 3
clf_gini = DecisionTreeClassifier(criterion = "gini",
                                  random_state = 100,
                                  max_depth=3,
                                  min_samples_leaf=50,
                                  min_samples_split=50)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

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

```
0.8393192617968837
```

[ ]:

```python
# 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())
```

[ ]:



[ ]:

```python
# 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
```

[ ]:
```python
# confusion matrix
print(confusion_matrix(y_test,y_pred))
```

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

Conclusion:

1. Through the utilization of the Label Encoder technique, categorical attributes have been transformed from their original textual forms into numerical representations, assigning a unique integer to each distinct categorical value within each respective column. This transformation facilitates the integration of these categorical variables into machine learning algorithms, which typically require numerical inputs.

2. Hyperparameter tuning was conducted based on the decision tree model obtained, with a focus on key hyperparameters:

- Max Depth: This parameter constrains the depth of the decision tree, guarding against excessive complexity and overfitting to the training data.
- Min Samples Split: Setting a minimum number of samples required in a node before further splitting helps prevent overly specific decisions based on a small number of instances.
- Min Samples Leaf: This parameter determines the minimum number of samples to be present in a leaf node, serving to discourage nodes with very few instances.
- Criterion: The choice of criterion (e.g., "Gini impurity" or "entropy") defines the measure of split quality.

3. The model achieved an accuracy of approximately 84%, implying that it correctly predicted the class labels for 84% of instances in the test dataset.

4. Examining the confusion matrix, we find the following results:
    - True Positive (TP): 1031
    - True Negative (TN): 6564
    - False Positive (FP): 303
    - False Negative (FN): 1151

    The confusion matrix reveals that the model performs reasonably well in predicting class 0, evident by the high count of true negatives and true positives. However, it faces challenges when predicting class 1.

5. Precision: Approximately 0.77, indicating that when the model predicts class 1, it is often correct. Roughly 77% of instances predicted as class 1 are genuinely class 1.

6. Recall: Approximately 0.47, implying that the model misses a notable number of actual class 1 instances. It correctly identifies only about 47% of all real instances belonging to class 1.

7. The F1 score for class 1 strikes a balance between precision and recall, offering a comprehensive view of the model's performance.