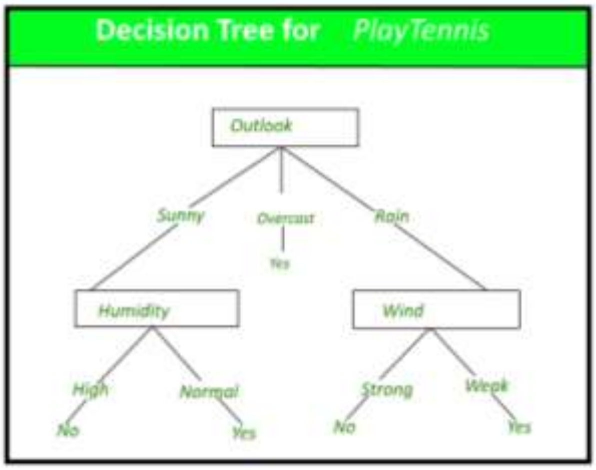
Experiment No. 3
Apply Decision Tree Algorithm on Adult Census Income Dataset and analyze the performance of the model
Date of Performance:
Date of Submission:

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.

Code:

Conclusion:

1. Discuss about the how categorical attributes have been dealt with during data pre-processing.
   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. Discuss the hyper-parameter tunning done based on the decision tree obtained.
   Hyperparameter tuning, also known as hyperparameter optimization, is the process of finding the best set of hyperparameters for a machine learning model. Hyperparameters are settings or configurations that are not learned from the data but are set prior to training a model.

3. Comment on the accuracy, confusion matrix, precision, recall and F1 score obtained.
   Accuracy: Model accuracy is about 81%. This means the model correctly predicted the class label.
   Precision: Precision measures the accuracy of the positive predictions and the precision score obtained by our model is 59%
   Recall: The recall rate is about 62%. This means that the model can only correctly identify about 62% of all real cases.
   F1 Score: The F1 score of 60.8% is between precision and recall, providing a balanced view of model performance.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import linear_model
from sklearn.tree import DecisionTreeClassifier
df= pd.read_csv('adult.csv')
```

```
df= pd.read_csv('adult.csv')
df.head()
```

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

```
df.shape
```

```
(32561, 15)
```

```
df.describe()
```

	age	fnlwgt	education.num	capital.gain	capital.loss	hours.p
count	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.000000	3256
mean	38.581647	1.897784e+05	10.080679	1077.648844	87.303830	4(
std	13.640433	1.055500e+05	2.572720	7385.292085	402.960219	1:
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	
25%	28.000000	1.178270e+05	9.000000	0.000000	0.000000	4(
50%	37.000000	1.783560e+05	10.000000	0.000000	0.000000	4(
75%	48.000000	2.370510e+05	12.000000	0.000000	0.000000	4!
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	9(

```
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
```

```
df['income'].value_counts()
```

```
<=50K    24720
>50K      7841
Name: income, dtype: int64
```

```
df['sex'].value_counts()
```

```
Male      21790
Female    10771
Name: sex, dtype: int64
```

```
df['native.country'].value_counts()
```

```
United-States                 29170
Mexico                          643
?                               583
Philippines                     198
Germany                         137
Canada                          121
Puerto-Rico                     114
El-Salvador                     106
India                           100
Cuba                             95
England                          90
Jamaica                          81
South                            80
China                            75
Italy                            73
Dominican-Republic               70
Vietnam                          67
Guatemala                        64
Japan                            62
Poland                           60
Columbia                         59
Taiwan                           51
Haiti                            44
Iran                             43
Portugal                         37
Nicaragua                        34
Peru                             31
Greece                           29
France                           29
Ecuador                          28
Ireland                          24
Hong                             20
Cambodia                         19
Trinadad&Tobago                  19
Laos                             18
Thailand                         18
Yugoslavia                       16
Outlying-US(Guam-USVI-etc)       14
Hungary                          13
Honduras                         13
Scotland                         12
Holand-Netherlands                1
Name: native.country, dtype: int64
```

```
df['workclass'].value_counts()
```

```
Private             22696
Self-emp-not-inc     2541
Local-gov            2093
?                    1836
State-gov            1298
Self-emp-inc         1116
Federal-gov           960
Without-pay            14
Never-worked            7
Name: workclass, dtype: int64
```

```
df['occupation'].value_counts()
```

```
Prof-specialty       4140
Craft-repair         4099
Exec-managerial      4066
Adm-clerical         3770
Sales                3650
Other-service        3295
Machine-op-inspct    2002
?                    1843
Transport-moving     1597
Handlers-cleaners    1370
Farming-fishing       994
Tech-support          928
Protective-serv       649
Priv-house-serv       149
Armed-Forces            9
Name: occupation, dtype: int64
```

```
df.replace('?', np.NaN,inplace = True)
df.head()
```

tal.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.week	native.cc
Widowed	NaN	Not-in-family	White	Female	0	4356	40	United-
Widowed	Exec-managerial	Not-in-family	White	Female	0	4356	18	United-
Widowed	NaN	Unmarried	Black	Female	0	4356	40	United-
Divorced	Machine-op-inspct	Unmarried	White	Female	0	3900	40	United-
Separated	Prof-specialty	Own-child	White	Female	0	3900	40	United-

```
#fills missing values (NaNs) with the most recent non-missing value
df.fillna(method = 'ffill', inplace = True)
```

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df['workclass'] = le.fit_transform(df['workclass'])
df['education'] = le.fit_transform(df['education'])
df['marital.status'] = le.fit_transform(df['marital.status'])
df['occupation'] = le.fit_transform(df['occupation'])
df['relationship'] = le.fit_transform(df['relationship'])
df['race'] = le.fit_transform(df['race'])
df['sex'] = le.fit_transform(df['sex'])
df['native.country'] = le.fit_transform(df['native.country'])
df['income'] = le.fit_transform(df['income'])
```

```
df.head()
```

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex
0	90	8	77053	11	9	6	14	1	4	C
1	82	3	132870	11	9	6	3	1	4	C
2	66	3	186061	15	10	6	3	4	2	C
3	54	3	140359	5	4	0	6	4	4	C
4	41	3	264663	15	10	5	9	3	4	C

```
df.describe()
```

onship	race	sex	capital.gain	capital.loss	hours.per.week	native.
000000	32561.000000	32561.000000	32561.000000	32561.000000	32561.000000	3256
446362	3.665858	0.669205	1077.648844	87.303830	40.437456	3
606771	0.848806	0.470506	7385.292085	402.960219	12.347429	
000000	0.000000	0.000000	0.000000	0.000000	1.000000	
000000	4.000000	0.000000	0.000000	0.000000	40.000000	3
000000	4.000000	1.000000	0.000000	0.000000	40.000000	3
000000	4.000000	1.000000	0.000000	0.000000	45.000000	3
000000	4.000000	1.000000	99999.000000	4356.000000	99.000000	4

```
#Splitting the data set into features and outcome
X = df.drop(['income'], axis=1)
Y = df['income']
```

```
df.isnull().sum()
```

```
age               0
workclass         0
fnlwgt            0
education         0
education.num     0
marital.status    0
occupation        0
relationship      0
race              0
sex               0
capital.gain      0
capital.loss      0
hours.per.week    0
native.country    0
income            0
dtype: int64
```

```
df.duplicated().sum()
```

```
24
```

```
df=df.dropna()
```

```
#Splitting the data into test data and training data

from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state = 42)
```

```
X_train.head()
```

	age	workclass	fnlwgt	education	education.num	marital.status	occupation
5514	26	3	256263	11	9	4	2
19777	24	3	170277	11	9	4	7
10781	36	3	75826	9	13	0	0
32240	22	6	24395	15	10	2	0
9876	31	1	356689	9	13	2	9

```
Y_train = Y_train.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)
```

```
Y_train.head()
```

```
0    0
1    0
2    0
3    0
4    0
Name: income, dtype: int64
```

```
Y_test = Y_test.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)
```

```
from sklearn.tree import DecisionTreeClassifier
dec_tree = DecisionTreeClassifier(random_state=42)
dec_tree.fit(X_train, Y_train)
Y_pred_dec_tree = dec_tree.predict(X_test)
```

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
```

```
print('Decision Tree Classifier:')
print('Accuracy score:',accuracy_score(Y_test, Y_pred_dec_tree) * 100)
print('Precision :',precision_score(Y_test,Y_pred_dec_tree) *100)
print('Recall: ',recall_score(Y_test,Y_pred_dec_tree)* 100)
print('F1 score: ',f1_score(Y_test,Y_pred_dec_tree) *100)
```

```
Decision Tree Classifier:
Accuracy score: 81.1761093198219
Precision : 59.749216300940446
Recall:  62.0039037085231
F1 score:  60.85568326947638
```

✓ 0s completed at 12:24 AM