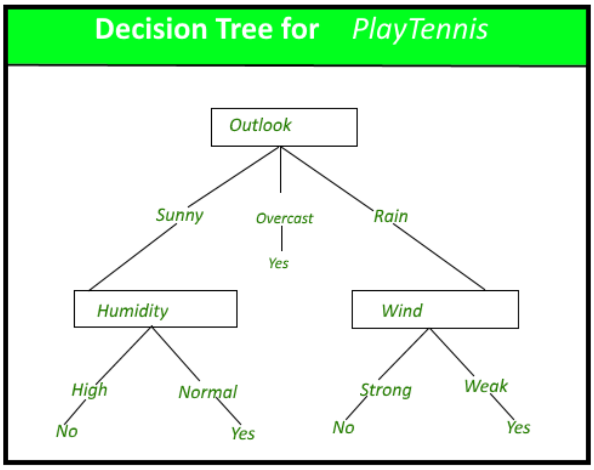
Experiment No. 3
Apply Decision Tree Algorithm on Adult Census Income Dataset and analyze the performance of the model
Date of Performance:
Date of Submission:

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

Objective: Able to perform various feature engineering tasks, apply Decision Tree Algorithm on the given dataset and maximize the accuracy, Precision, Recall, F1 score.

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 & Output:

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
# Adult dataset path
adult_dataset_path = "/content/adult_dataset.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	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.week	native.country	income
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-family	White	Female	0	4356	40	United-States	<=50K
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0	4356	18	United-States	<=50K
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	Black	Female	0	4356	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
```

```
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)
```

```
df.describe()
```

	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

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

df_check_missing_workclass
```

1836

```
f_check_missing_occupation = (df['occupation']=='?').sum()

df_check_missing_occupation

1843
df_missing = (df=='?').sum()

df_missing
```

	0
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
```

	0
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

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

	0
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

```
# 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()
```

	0
workclass	1836
education	0
marital.status	0
occupation	1843
relationship	0
race	0
sex	0
native.country	583
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'>
Index: 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	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

```
# 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	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

```
# 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	workclass	education	marital.status	occupation	relationship	race	sex	native.country	income
1	82	132870	9	0	4356	18	2	11	6	3	1	4	0	38	0
3	54	140359	4	0	3900	40	2	5	0	6	4	4	0	38	0
4	41	264663	10	0	3900	40	2	15	5	9	3	4	0	38	0
5	34	216864	9	0	3770	45	2	11	0	7	4	4	0	38	0
6	38	150601	6	0	3770	40	2	0	5	0	4	4	1	38	0

```
# convert target variable income to categorical

df['income'] = df['income'].astype('category')

# 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	workclass	education	marital.status	occupation	relationship	race	sex	native.country
1	82	132870	9	0	4356	18	2	11	6	3	1	4	0	38
3	54	140359	4	0	3900	40	2	5	0	6	4	4	0	38
4	41	264663	10	0	3900	40	2	15	5	9	3	4	0	38

```
# 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	workclass	education	marital.status	occupation	relationship	race	sex	native.country
24351	42	289636	9	0	0	46	2	11	2	13	0	4	1	38
15626	37	52465	9	0	0	40	1	11	4	7	1	4	1	38
4347	38	125933	14	0	0	40	0	12	2	9	0	4	1	19
23972	44	183829	13	0	0	38	5	9	4	0	1	4	0	38
26843	35	198841	11	0	0	35	2	8	0	12	3	4	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)

# 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
```

Conclusion:

The model performs well overall with an accuracy of 85%. However, the performance on class 1 (minority class) is not as strong, with lower recall and F1-score. This indicates that the model is better at identifying the majority class (class 0) but struggles with the minority class (class 1).

confusion matrix:

- True Positives (TP) for class 1 = Recall × Support = 0.52 × 2182 ≈ 1135
- False Negatives (FN) for class 1 = Support - TP = 2182 - 1135 ≈ 1047
- True Negatives (TN) for class 0 = Recall × Support = 0.95 × 6867 ≈ 6524
- False Positives (FP) for class 0 = Support - TN = 6867 - 6524 ≈ 343