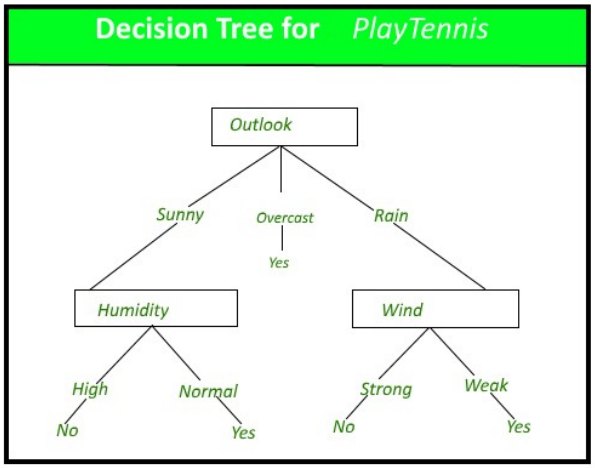
Experiment No. 3
Apply Decision Tree Algorithm on Adult Census Income Dataset and analyze the performance of the model
Date of Performance:
Date of Submission:

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:

```
# 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')
```

```
[3] # 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)
```

```
[4] # 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	0.0	4356.0	40.0	United-States	<=50K
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0.0	4356.0	18.0	United-States	<=50K
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	Black	Female	0.0	4356.0	40.0	United-States	<=50K

```
[5] 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     : 40205
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 :  15
```

```
[6] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40205 entries, 0 to 40204
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             40205 non-null  object
 1   workclass       40205 non-null  object
 2   fnlwgt          40205 non-null  object
 3   education       40205 non-null  object
 4   education.num   40205 non-null  object
 5   marital.status  40204 non-null  object
 6   occupation      40204 non-null  object
 7   relationship    40204 non-null  object
 8   race            40204 non-null  object
 9   sex             40204 non-null  object
 10  capital.gain    40203 non-null  float64
 11  capital.loss    40203 non-null  float64
 12  hours.per.week  40203 non-null  float64
 13  native.country  40203 non-null  object
 14  income          40203 non-null  object
dtypes: float64(3), object(12)
memory usage: 4.6+ MB
```

```
[7] df_missing = (df=='?').sum()
    df_missing
```

```
age                0
workclass       2297
fnlwgt             0
education          0
education.num      0
marital.status     0
occupation      2304
relationship       0
race               0
sex                0
capital.gain       0
capital.loss       0
hours.per.week     0
native.country   717
income             0
dtype: int64
```

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

```
age             0.000000
workclass       5.713220
fnlwgt          0.000000
education       0.000000
education.num   0.000000
marital.status  0.000000
occupation      5.730631
relationship    0.000000
race            0.000000
```

0s completed a

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

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.week	native.country	income
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0.0	4356.0	18.0	United-States	<=50K
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	0.0	3900.0	40.0	United-States	<=50K
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	0.0	3900.0	40.0	United-States	<=50K
5	34	Private	216864	HS-grad	9	Divorced	Other-service	Unmarried	White	Female	0.0	3770.0	45.0	United-States	<=50K
6	38	Private	150601	10th	6	Separated	Adm-clerical	Unmarried	White	Male	0.0	3770.0	40.0	United-States	<=50K

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

```
from sklearn import preprocessing

# encode categorical variables using label Encoder

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

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	native.country	income
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	United-States	<=50K
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	United-States	<=50K
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	United-States	<=50K
5	34	Private	216864	HS-grad	9	Divorced	Other-service	Unmarried	White	Female	United-States	<=50K
6	38	Private	150601	10th	6	Separated	Adm-clerical	Unmarried	White	Male	United-States	<=50K

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

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	native.country	income
1	65	2	2501	11	15	6	3	1	4	0	38	0
3	37	2	2971	5	10	0	6	4	4	0	38	0
4	24	2	12347	15	1	5	9	3	4	0	38	0

```
# 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
DecisionTreeClassifier(max_depth=5)
```

```
[20]
# 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       0.85      0.95      0.90      8491
           1       0.77      0.48      0.59      2674

    accuracy                           0.84     11165
   macro avg       0.81      0.72      0.75     11165
weighted avg       0.83      0.84      0.83     11165
```

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

```
[[8099  392]
 [1387 1287]]
0.8406627854903717
```

```
[ ] !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)
```

Conclusion:

1. Discuss about the how categorical attributes have been dealt with during data pre-processing.

=>Missing values with '?' are removed from specific columns (e.g., 'workclass,' 'occupation,' 'native.country').

Categorical attributes are label-encoded, converting them into numerical values.

The target variable ('income') is converted to a categorical format.

The data is split into training and testing sets.

A decision tree classifier is trained on the data

Model performance is evaluated using classification metrics like precision, recall, F1-score, and accuracy

2. Discuss the hyper-parameter tunning done based on the decision tree obtained.

Hyperparameter tuning is the process of finding the optimal values for the hyperparameters of a machine learning model to improve its performance.

In the provided code, only one hyperparameter, max_depth, is manually set to 5. While this can be useful for creating a more interpretable tree, it may not necessarily result in the best predictive performance.

3. Comment on the accuracy, confusion matrix, precision, recall and F1 score obtained.

ACCURACY :

Accuracy measures the proportion of correctly classified instances out of all instances. In this case, the model has an accuracy of approximately 84.07%, indicating that it correctly predicts the income category for about 84.07% of the test data.

CONFUSION MATRIX:

The confusion matrix provides a more detailed view of the model's performance:

True Positives (TP): 1287 - The number of instances correctly classified as positive (income > 50K)

True Negatives (TN): 8099 - The number of instances correctly classified as negative (income <= 50K).

False Positives (FP): 392 - The number of instances incorrectly classified as positive (income > 50K).

False Negatives (FN): 1387 - The number of instances incorrectly classified as negative (income <= 50K).

PRECISION

In this case, precision for the positive class (income > 50K) can be calculated as 1287 / (1287 + 392), which is the ratio of correctly predicted high-income individuals to all predicted high-income individuals.

RECALL:

In this case, recall for the positive class is 1287 / (1287 + 1387), which is the ratio of

correctly predicted high-income individuals to all actual high-income individuals.

F1 SCORE :

The F1 score, which is approximately 0.5901 in this case, is a measure of a model's performance that balances both precision and recall