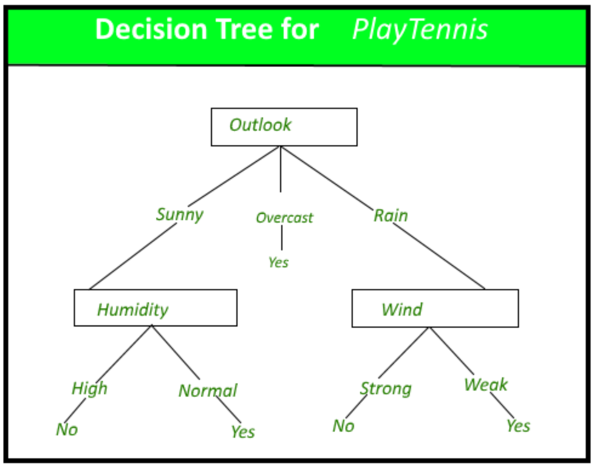
Experiment No. 3
Apply Decision Tree Algorithm on Adult Census Income Dataset and analyze the performance of the model
Date of Performance:
Date of Submission:

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 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 = "/content/adult_dataset.csv"

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

```
df = load_adult_data()
df.head()
```

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-family	White	Female	0	4356
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0	4356
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	Black	Female	0	4356
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	0	3900
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	0	3900

```
df.info()
percent_missing = (df=='?').sum() * 100/len(df)
percent_missing
```

```
<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
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 = df[df['workclass'] !='?']
df.head()
```

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0	4356
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	0	3900
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	0	3900
5	34	Private	216864	HS-grad	9	Divorced	Other-service	Unmarried	White	Female	0	3770
6	38	Private	150601	10th	6	Separated	Adm-clerical	Unmarried	White	Male	0	3770

```
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
```

```
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

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

```
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

```
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	relationshi
1	82	132870	9	0	4356	18	2	11	6	3	[illegible]
3	54	140359	4	0	3900	40	2	5	0	6	[illegible]
4	41	264663	10	0	3900	40	2	15	5	9	[illegible]
5	34	216864	9	0	3770	45	2	11	0	7	[illegible]
6	38	150601	6	0	3770	40	2	0	5	0	[illegible]

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

```
from sklearn.model_selection import train_test_split
X = df.drop('income',axis=1)
y = df['income']
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	relatior

```
from sklearn.tree import DecisionTreeClassifier

dt_default = DecisionTreeClassifier(max_depth=5)
dt_default.fit(X_train,y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=5)
```

```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score

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
```

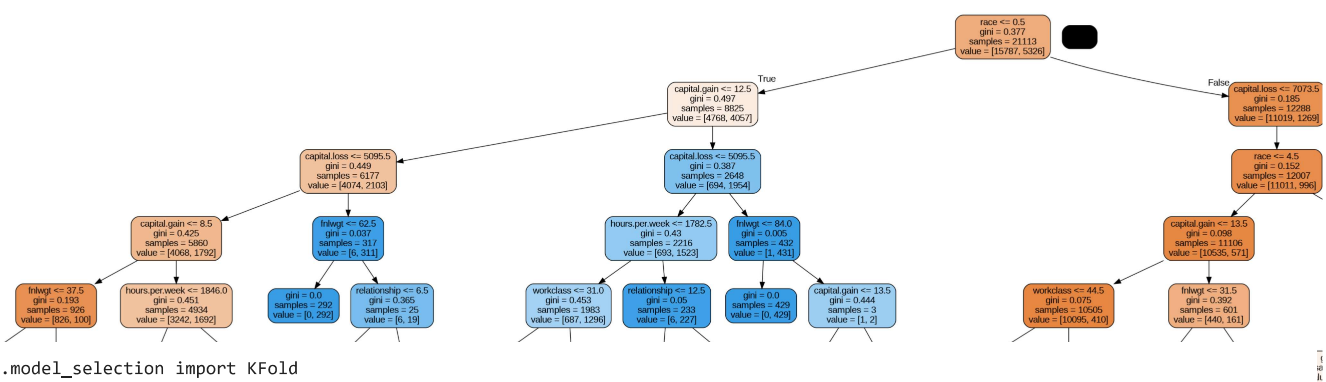
```
print(confusion_matrix(y_test,y_pred_default))
print(accuracy_score(y_test,y_pred_default))
```

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

```
from IPython.display import Image
from six import StringIO
from sklearn.tree import export_graphviz
import pydotplus,graphviz

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




```
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)
scores = tree.cv_results_
pd.DataFrame(scores).head()
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	params	split0_test_score	split1_test_score	spl:
0	0.017888	0.003653	0.004094	0.000567	1	{'max_depth': 1}	0.747810	0.747810	
1	0.019536	0.002511	0.003296	0.000128	2	{'max_depth': 2}	0.812219	0.818612	
2	0.025570	0.001021	0.003385	0.000179	3	{'max_depth': 3}	0.828558	0.834241	
3	0.031259	0.002697	0.003501	0.000278	4	{'max_depth': 4}	0.832583	0.840871	
4	0.034728	0.000259	0.003315	0.000084	5	{'max_depth': 5}	0.834241	0.844897	

```
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)
scores = tree.cv_results_
pd.DataFrame(scores).head()
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_leaf	params	split0_test_score	split1_tes
0	0.083604	0.002404	0.003690	0.000076	5	{'min_samples_leaf': 5}	0.825716	[illegible]
1	0.068328	0.001686	0.003611	0.000128	25	{'min_samples_leaf': 25}	0.841819	[illegible]
2	0.064112	0.005004	0.003481	0.000097	45	{'min_samples_leaf': 45}	0.843003	[illegible]
3	0.060507	0.001580	0.003446	0.000062	65	{'min_samples_leaf': 65}	0.841108	
4	0.057527	0.002340	0.003587	0.000302	85	{'min_samples_leaf': 85}	0.838030	[illegible]

```
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)
scores = tree.cv_results_
pd.DataFrame(scores).head()
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_split	params	split0_test_score	split1_t
0	0.131679	0.005760	0.005303	0.000086	5	{'min_samples_split': 5}	0.811982	
1	0.111237	0.014334	0.004763	0.001007	25	{'min_samples_split': 25}	0.825006	
2	0.085567	0.006586	0.004064	0.000822	45	{'min_samples_split': 45}	0.835188	
3	0.078796	0.001440	0.004016	0.000491	65	{'min_samples_split': 65}	0.839451	

```
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)
cv_results = pd.DataFrame(grid_search.cv_results_)
cv_results
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param_max_depth	param_min_samples_leaf	param_min_sa
0	0.040272	0.004975	0.003267	0.000087	entropy	5	50	
1	0.037706	0.000605	0.003830	0.001051	entropy	5	50	
2	0.037048	0.000511	0.003274	0.000134	entropy	5	100	
3	0.037072	0.000566	0.003257	0.000132	entropy	5	100	
4	0.061445	0.002255	0.003530	0.000087	entropy	10	50	
5	0.061074	0.001897	0.003580	0.000254	entropy	10	50	
6	0.056859	0.001767	0.003367	0.000067	entropy	10	100	

```
print("best accuracy", grid_search.best_score_)
print(grid_search.best_estimator_)
```

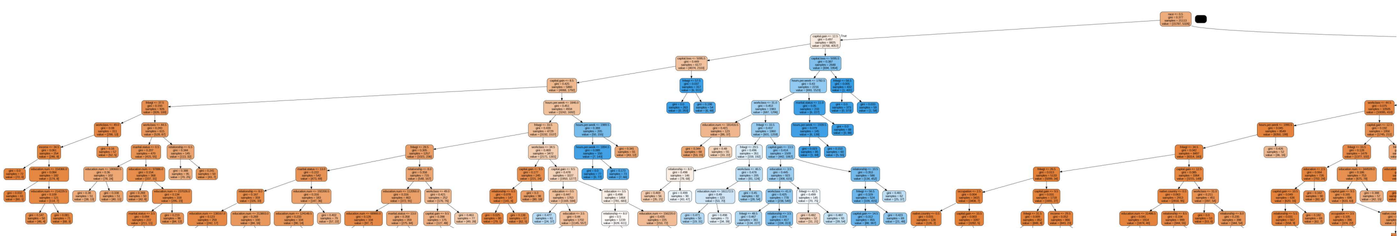
```
best accuracy 0.8510400232064759
DecisionTreeClassifier(max_depth=10, min_samples_leaf=50, min_samples_split=50)
```

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

```
0.850922753895458
```

```
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())
```



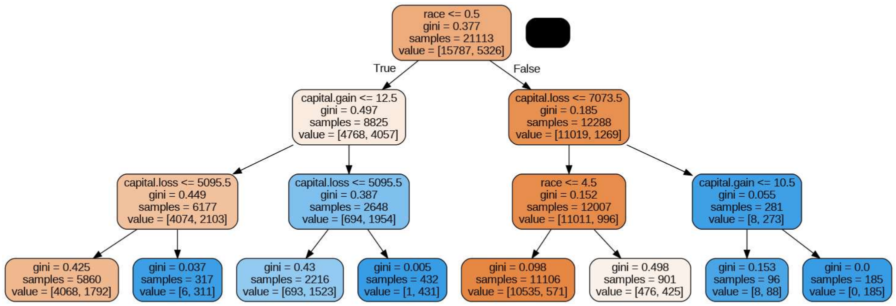
```
clf_gini = DecisionTreeClassifier(criterion = "gini",
                                  random_state = 100,
                                  max_depth=3,
                                  min_samples_leaf=50,
                                  min_samples_split=50)
clf_gini.fit(X_train, y_train)

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

```
0.8393192617968837
```

```
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())
```




```
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
```

```
print(confusion_matrix(y_test,y_pred))
```

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

Conclusion:

1. The categorical values can be handled majorly in two ways. One way is with one hot encoding and another is with label encoding. In this experiment we have used label encoder to handle the categorical values.
2. The default tree is quite complex, and we need to simplify it by tuning the hyperparameters. In this dataset we have tuned the following parameters.

   A. max_depth: The max_depth parameter denotes maximum depth of the tree. It can take any integer value or None.

   B. min_samples_leaf: The hyperparameter min_samples_leaf indicates the minimum number of samples required to be at a leaf.

   C. min_samples_split: The hyperparameter min_samples_split is the minimum no. of samples required to split an internal node. Its default value is 2, which means that even if a node has 2 samples it can be further divided into leaf nodes.

3. Accuracy: We have achieved an accuracy of 85% on our testing dataset.

   Confusion Matrix: True positive = 6564, False positive = 303, False negative = 1151, True negative = 1031.

   Precision: We have achieved precision of 84% for positive class and 77% for negative class.

   Recall: Recall of 0.95 indicates that the model captured the instances for positive class and 0.47 model captured the instances for negative class.

   F1 Score: The F1 score of 0.90 is the mean of precision and recall for positive class and 0.59 is the mean of precision and recall for negative class in the model's performance.