Experiment No. 4
Apply Random Forest Algorithm on Adult Census Income Dataset and analyze the performance of the model
Date of Performance:14–08–23
Date of Submission:22–08–23

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

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

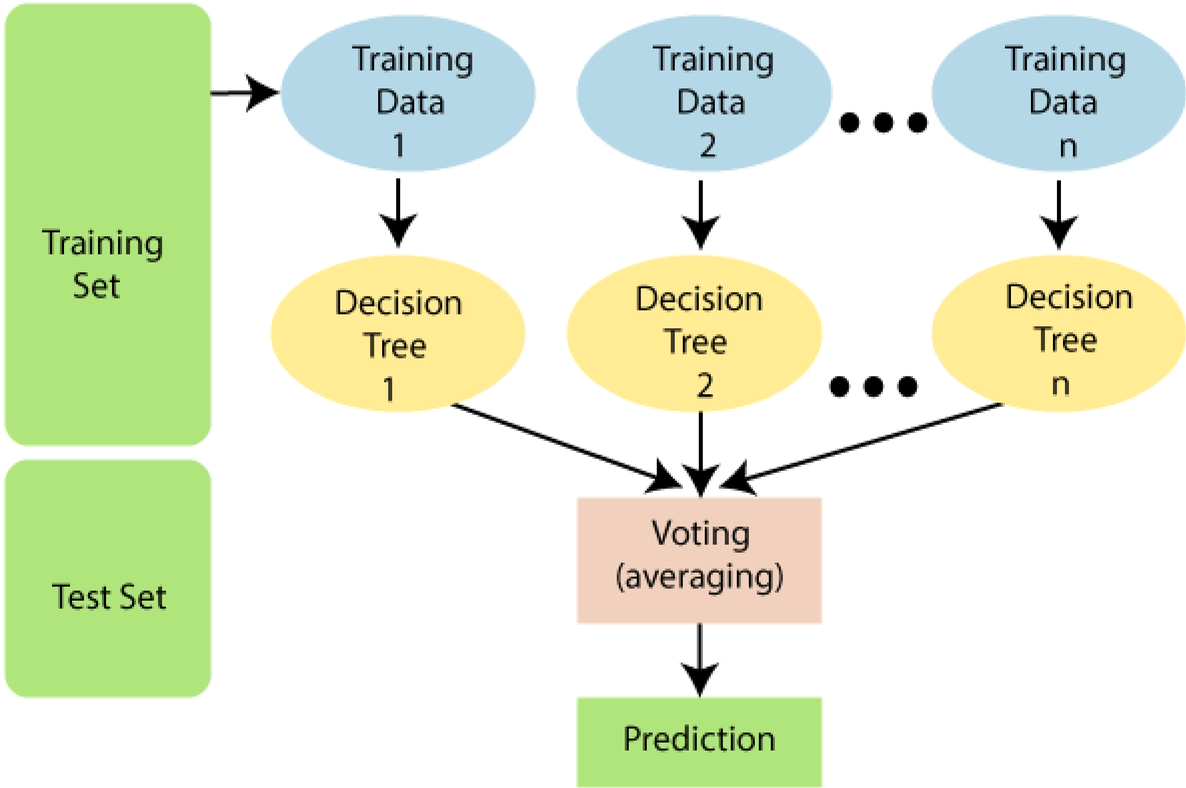
Theory:

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

The below diagram explains the working of the Random Forest algorithm:



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:

The correlations among these variables are generally weak in terms of strength, lacking strong linear associations with one another.

Specifically, age shows a mild positive correlation with both education number and hours worked per week. Education numbers exhibit a slight positive correlation with capital gains, and there is a weak negative correlation between capital gains and capital losses.

In terms of model evaluation:

Accuracy: The model achieves an accuracy of 85.44%, indicating that it correctly predicts income levels for the majority of instances.

Confusion Matrix: The confusion matrix reveals 8015 true positive predictions, 628 false positive predictions, 1047 false negatives, and 1819 true negative predictions.

Precision: For income level 0, precision is 0.08, and for income level 1, precision is 0.74. This means that for income level 0, only a small fraction of predictions are accurate, while for income level 1, a significant portion of predictions are accurate.

Recall: The recall for income level 0 is 0.93, indicating that the model effectively captures a high proportion of instances belonging to income level 0. However, the recall for income level 1 is 0.63, suggesting that the model captures a moderate proportion of instances from income level 1.

F1-score: The F1-score is a balanced measure combining precision and recall. It is 0.91 for income level 0 and 0.68 for income level 1, indicating the overall effectiveness of the model.

In comparison to a Decision Tree, a Random Forest tends to yield superior results. The Random Forest model aggregates predictions from multiple Decision Trees, which can enhance accuracy and generalization in the model's performance.

```
# Import libraries
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set(style='white', context='notebook', palette='deep')
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, cross_val_score, StratifiedKFold, learning_curve, train_test_split, KFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score


# To ignore warning messages
import warnings
warnings.filterwarnings('ignore')
# Adult dataset path
adult_dataset_path = "/content/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	educational-num	marital-status	occupation	relationship
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband

```
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 :  48842
Columns :  15

Features :
 ['age', 'workclass', 'fnlwgt', 'education', 'educational-num', 'marital-status', 'occupation', 'relationship', 'race', 'gender', 'capit

Missing values :  0

Unique values :
 age                  74
workclass             9
fnlwgt            28523
education            16
educational-num      16
marital-status        7
occupation           15
relationship          6
race                  5
gender                2
capital-gain        123
capital-loss         99
hours-per-week       96
native-country       42
income                2
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   age              48842 non-null  int64 
 1   workclass        48842 non-null  object
 2   fnlwgt           48842 non-null  int64 
 3   education        48842 non-null  object
 4   educational-num  48842 non-null  int64 
 5   marital-status   48842 non-null  object
 6   occupation       48842 non-null  object
 7   relationship     48842 non-null  object
 8   race             48842 non-null  object
 9   gender           48842 non-null  object
 10  capital-gain     48842 non-null  int64 
 11  capital-loss     48842 non-null  int64 
 12  hours-per-week   48842 non-null  int64 
 13  native-country   48842 non-null  object
 14  income           48842 non-null  object
dtypes: int64(6), object(9)
memory usage: 5.6+ MB
```

```
df.describe()
```

	age	fnlwgt	educational-num	capital-gain	capital-loss	hours-per-week
count	48842.000000	4.884200e+04	48842.000000	48842.000000	48842.000000	48842.000000
mean	38.643585	1.896641e+05	10.078089	1079.067626	87.502314	40.422382
std	13.710510	1.056040e+05	2.570973	7452.019058	403.004552	12.391444
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.175505e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.781445e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.376420e+05	12.000000	0.000000	0.000000	45.000000

```
df.head()
```

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relation
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husl
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husl
3	44	Private	160323	Some-	10	Married-civ-	Machine-	Husl

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

```
2799
```

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

```
2809
```

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

```
age                  0
workclass         2799
fnlwgt               0
education            0
educational-num      0
marital-status       0
occupation        2809
relationship         0
race                 0
gender               0
capital-gain         0
capital-loss         0
hours-per-week       0
native-country     857
income               0
dtype: int64
```

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

```
age                0.000000
workclass          5.730724
fnlwgt             0.000000
education          0.000000
educational-num    0.000000
marital-status     0.000000
occupation         5.751198
relationship       0.000000
race               0.000000
gender             0.000000
capital-gain       0.000000
capital-loss       0.000000
hours-per-week     0.000000
native-country     1.754637
income             0.000000
dtype: float64
```

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

```
age                48842
workclass          46043
fnlwgt             48842
education          48842
educational-num    48842
marital-status     48842
occupation         46033
relationship       48842
race               48842
gender             48842
capital-gain       48842
capital-loss       48842
hours-per-week     48842
native-country     47985
income             48842
dtype: int64
```

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

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relation
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husl

```
# 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         10
relationship        0
race                0
gender              0
native-country    811
income              0
dtype: int64
```

```
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	gender	native-country
0	Private	11th	Never-married	Machine-op-inspct	Own-child	Black	Male	United-States
1	Private	HS-grad	Married-civ-spouse	Farming-fishing	Husband	White	Male	United-States
			Married-					

```
# 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	gender	native-country
0	2	1	4	6	3	2	1	39
1	2	11	2	4	0	4	1	39
2	1	7	2	10	0	4	1	39
3	2	15	2	6	0	2	1	39

```
# 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	educational-num	capital-gain	capital-loss	hours-per-week	workclass	education	ma

```
# look at column type
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 46033 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   age              46033 non-null  int64
 1   fnlwgt           46033 non-null  int64
 2   educational-num  46033 non-null  int64
 3   capital-gain     46033 non-null  int64
 4   capital-loss     46033 non-null  int64
 5   hours-per-week   46033 non-null  int64
 6   workclass        46033 non-null  int64
 7   education        46033 non-null  int64
 8   marital-status   46033 non-null  int64
 9   occupation       46033 non-null  int64
 10  relationship     46033 non-null  int64
 11  race             46033 non-null  int64
 12  gender           46033 non-null  int64
 13  native-country   46033 non-null  int64
 14  income           46033 non-null  int64
dtypes: int64(15)
memory usage: 5.6 MB
```

```
plt.figure(figsize=(14,10))
sns.heatmap(df.corr(),annot=True,fmt='.2f')
plt.show()
```

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

```
# check df info again whether everything is in right format or not
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 46033 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   age              46033 non-null  int64
 1   fnlwgt           46033 non-null  int64
 2   educational-num  46033 non-null  int64
 3   capital-gain     46033 non-null  int64
 4   capital-loss     46033 non-null  int64
 5   hours-per-week   46033 non-null  int64
 6   workclass        46033 non-null  int64
 7   education        46033 non-null  int64
 8   marital-status   46033 non-null  int64
 9   occupation       46033 non-null  int64
 10  relationship     46033 non-null  int64
 11  race             46033 non-null  int64
 12  gender           46033 non-null  int64
 13  native-country   46033 non-null  int64
 14  income           46033 non-null  category
dtypes: category(1), int64(14)
memory usage: 5.3 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	educational-num	capital-gain	capital-loss	hours-per-week	workclass	education	ma
0	25	226802	7	0	0	40	2	1	
1	38	89814	9	0	0	50	2	11	

```
y.head(3)
```

```
0    0
1    0
2    1
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)
X_train.head()
```

	age	fnlwgt	educational-num	capital-gain	capital-loss	hours-per-week	workclass	education
13554	58	196502	10	0	0	60	2	15
46282	27	297457	9	0	0	40	2	11
25679	27	30244	9	0	0	80	4	11
8775	42	165309	9	0	0	50	2	11

```
test_size = 0.20
seed = 7
num_folds = 10
scoring = 'accuracy'
# Params for Random Forest
```

```
num_trees = 100
max_features = 3


random_forest = RandomForestClassifier(n_estimators=250,max_features=5)
random_forest.fit(X_train, y_train)
predictions = random_forest.predict(X_test)
print("Accuracy: %s%%" % (100*accuracy_score(y_test, predictions)))
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
```

```
Accuracy: 85.44617256060475%
[[8015  628]
 [1047 1819]]
              precision    recall  f1-score   support

           0       0.88      0.93      0.91      8643
           1       0.74      0.63      0.68      2866

    accuracy                           0.85     11509
   macro avg       0.81      0.78      0.80     11509
weighted avg       0.85      0.85      0.85     11509
```