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

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.

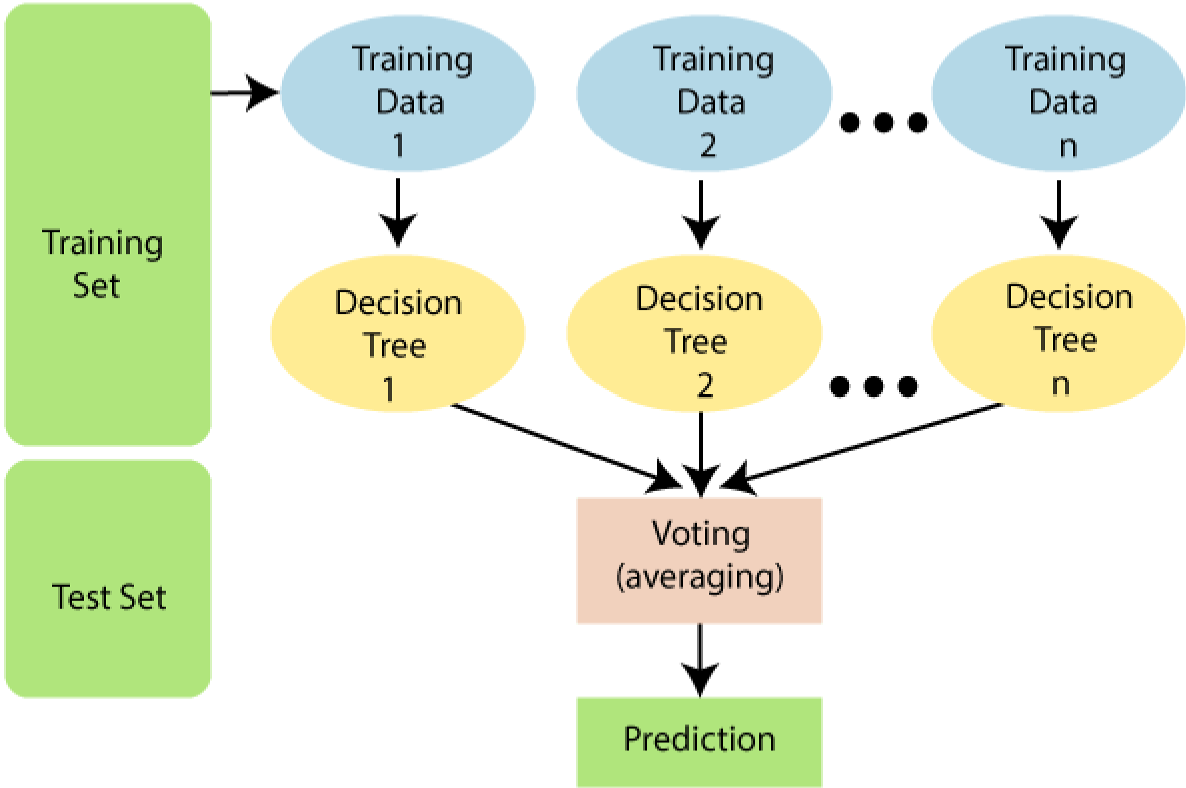
Theory:

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:

1. State the observations about the data set from the correlation heat map.

Almost all the attributes show relatively weak correlation with each other. Sex and relationship have a comparatively higher negative correlation. A negative correlation, often referred to as an inverse or inverse correlation, describes a statistical relationship between two variables where they move in opposite directions. In other words, when one variable increases, the other tends to decrease, and vice versa.

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

The model's accuracy of 85.78% suggests that it's performing reasonably well in overall classification.

Looking at the confusion matrix, it's evident that there were 5358 true negatives and 1232 true positives, while 406 instances were false negatives, and 686 were false positives. This information helps us understand the model's performance in more detail. The model's precision for class 0 (negative class) is 89%, indicating a high percentage of true negatives correctly classified, while for class 1 (positive class), it's 75%, suggesting some false positives.

The recall for class 0 is 93%, which means that the model effectively captures most of the actual negative instances. However, the recall for class 1 is 64%, indicating that it misses a significant portion of actual positives.

The F1-score, which balances precision and recall, is 0.91 for class 0 and 0.69 for class 1. This shows a trade-off between precision and recall, with class 0 having a better balance than class 1.

3. Compare the results obtained by applying random forest and decision tree algorithm on the Adult Census Income Dataset

When comparing the results of the Random Forest and Decision Tree classifiers:

Accuracy: Random Forest achieves a higher accuracy of 85.78% compared to the Decision Tree's 81.18%. This indicates that Random Forest is better at making overall correct predictions.

Precision: Random Forest has a higher precision for both classes (0 and 1) compared to the Decision Tree. This suggests that Random Forest is better at minimizing false positives.

Recall: Random Forest exhibits a better recall for both classes as well, indicating that it is better at capturing actual positive instances (class 1) and actual negative instances (class 0) compared to the Decision Tree.

F1 Score: Random Forest also outperforms the Decision Tree in terms of the F1 score for both classes, signifying a better balance between precision and recall for both classes.

```
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
```

```
import warnings
warnings.filterwarnings('ignore')
adult_dataset_path = "/content/adult.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	rela
0	90	?	77053	HS-grad	9	Widowed	?	No
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	No
2	66	?	186061	Some-college	10	Widowed	?	
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	

```
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', 'capit

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)
memory usage: 3.7+ MB
```

```
df.describe()
```

	age	fnlwgt	education.num	capital.gain	capital.loss	hours.p
count	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.000000	32561
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	12
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1
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	45
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99

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

```
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
```

```
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()
```

```
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
```

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

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	rela
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	No
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	
5	34	Private	216864	HS-grad	9	Divorced	Other-service	
6	38	Private	150601	10th	6	Separated	Adm-clerical	

```
df_categorical = df.select_dtypes(include=['object'])
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
```

```
from sklearn import preprocessing
df_categorical = df.select_dtypes(include=['object'])
df_categorical.head()
```

	workclass	education	marital.status	occupation	relationship	race	sex	nat
1	Private	HS-grad	Widowed	Exec-managerial	Not-in-family	White	Female	
3	Private	7th-8th	Divorced	Machine-op-inspct	Unmarried	White	Female	
4	Private	Some-college	Separated	Prof-specialty	Own-child	White	Female	

```
# 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.
1	3	11	6	4	1	4	0	
3	3	5	0	7	4	4	0	
4	3	15	5	10	3	4	0	
5	3	11	0	8	4	4	0	
6	3	0	5	1	4	4	1	

```
# 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	education.num	capital.gain	capital.loss	hours.per.week	workclass
1	82	132870	9	0	4356	18	3
3	54	140359	4	0	3900	40	3
4	41	264663	10	0	3900	40	3
5	34	216864	9	0	3770	45	3
6	38	150601	6	0	3770	40	3

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30725 entries, 1 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             30725 non-null  int64
 1   fnlwgt          30725 non-null  int64
 2   education.num   30725 non-null  int64
 3   capital.gain    30725 non-null  int64
 4   capital.loss    30725 non-null  int64
 5   hours.per.week  30725 non-null  int64
 6   workclass       30725 non-null  int64
 7   education       30725 non-null  int64
 8   marital.status  30725 non-null  int64
 9   occupation      30725 non-null  int64
 10  relationship    30725 non-null  int64
 11  race            30725 non-null  int64
 12  sex             30725 non-null  int64
 13  native.country  30725 non-null  int64
 14  income          30725 non-null  int64
dtypes: int64(15)
memory usage: 3.8 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: 30725 entries, 1 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             30725 non-null  int64
 1   fnlwgt          30725 non-null  int64
 2   education.num   30725 non-null  int64
 3   capital.gain    30725 non-null  int64
 4   capital.loss    30725 non-null  int64
 5   hours.per.week  30725 non-null  int64
 6   workclass       30725 non-null  int64
 7   education       30725 non-null  int64
 8   marital.status  30725 non-null  int64
 9   occupation      30725 non-null  int64
 10  relationship    30725 non-null  int64
 11  race            30725 non-null  int64
 12  sex             30725 non-null  int64
 13  native.country  30725 non-null  int64
 14  income          30725 non-null  category
dtypes: category(1), int64(14)
memory usage: 3.5 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	education.num	capital.gain	capital.loss	hours.per.week	workclass
1	82	132870	9	0	4356	18	3
3	54	140359	4	0	3900	40	3
4	41	264663	10	0	3900	40	3

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

```
1    0
3    0
4    0
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	education.num	capital.gain	capital.loss	hours.per.week	workc
23791	67	286372	13	0	0	40	
17414	34	318886	9	0	0	40	
13106	38	200153	12	0	0	40	
12421	62	211035	13	0	0	30	
13576	42	31621	12	0	0	40	

```
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: 85.78495183545951%
[[5358  406]
 [ 686 1232]]
              precision    recall  f1-score   support

           0       0.89      0.93      0.91      5764
           1       0.75      0.64      0.69      1918

    accuracy                           0.86      7682
   macro avg       0.82      0.79      0.80      7682
weighted avg       0.85      0.86      0.85      7682
```