Experiment No. 4
Apply Random Forest Algorithm on Adult Census Income Dataset and analyze the performance of the model
Date of Performance: 14-08-2023
Date of Submission: 22-08-2023

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.

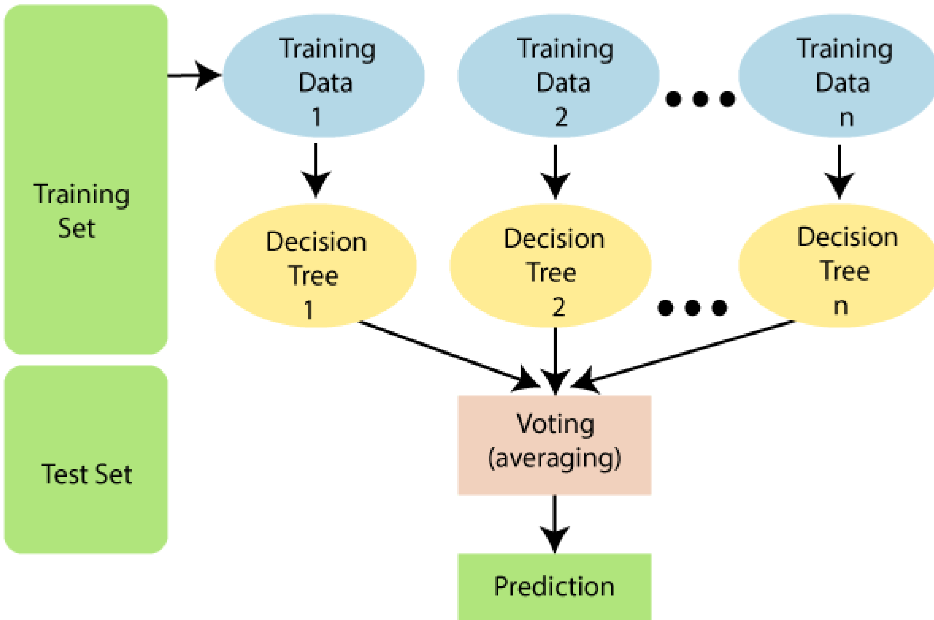
Theory:

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.

```
import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from collections import Counter

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, ExtraTreesClassifier, VotingClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB
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

sns.set(style='white', context='notebook', palette='deep')
# 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	education.num	marital.status	occupation	relatio
0	90	?	77053	HS-grad	9	Widowed	?	Not-in·
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in·
2	66	?	186061	Some-college	10	Widowed	?	Unm

```
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())
```

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

```
# Let's understand the type of values present in each column of our adult dataframe 'df'.
df.info()
```

	age	workclass	fnlwgt	education	education.num	marital.status	occup
0	90	?	77053	HS-grad	9	Widowed	
1	82	Private	132870	HS-grad	9	Widowed	mana
2	66	?	186061	Some-college	10	Widowed	
							Ma

```
# Numerical feature of summary/description
df.describe()
```

	age	fnlwgt	education.num	capital.gain	capital.los
count	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.00000
mean	38.581647	1.897784e+05	10.080679	1077.648844	87.3038
std	13.640433	1.055500e+05	2.572720	7385.292085	402.9602
min	17.000000	1.228500e+04	1.000000	0.000000	0.0000
25%	28.000000	1.178270e+05	9.000000	0.000000	0.0000
50%	37.000000	1.783560e+05	10.000000	0.000000	0.0000
75%	48.000000	2.370510e+05	12.000000	0.000000	0.0000

```
# Reformat Column We Are Predicting
dataset['income']=dataset['income'].map({'<=50K': 0, '>50K': 1, '<=50K.': 0, '>50K.': 1})
dataset.head(4)
```

	age	workclass	fnlwgt	education	education.num	marital.status	occup
0	90	?	77053	HS-grad	9	Widowed	
1	82	Private	132870	HS-grad	9	Widowed	mana
2	66	?	186061	Some-college	10	Widowed	
							Ma

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

```
1836
```

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

```
1843
```

```
# checking "?" values, how many are there in the whole dataset
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
```

```
# Let's find total number of rows which doesn't contain any missing value as '?'
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	occup
1	82	Private	132870	HS-grad	9	Widowed	mana
3	54	Private	140359	7th-8th	4	Divorced	Ma op
4	41	Private	264663	Some-college	10	Separated	sp
5	34	Private	216864	HS-grad	9	Divorced	s

```
# 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        7
relationship      0
race              0
```

```
sex                 0
native.country    556
income              0
dtype: int64
```

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

```
# check the dataset whether cleaned or not?
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

# 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
1	Private	HS-grad	Widowed	Exec-managerial	Not-in-family	White
3	Private	7th-8th	Divorced	Machine-op-inspct	Unmarried	White
4	Private	Some-college	Separated	Prof-specialty	Own-child	White

```
# 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
1	2	11	6	3	1	4
3	2	5	0	6	4	4
4	2	15	5	9	3	4
5	2	11	0	7	4	4

```
# 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
1	82	132870	9	0	4356	18
3	54	140359	4	0	3900	40
4	41	264663	10	0	3900	40
5	34	216864	9	0	3770	45

```
# look at column type
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  int64
dtypes: int64(15)
memory usage: 3.7 MB
```

```
# 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: 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
```

```
# 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
1	82	132870	9	0	4356	18
3	54	140359	4	0	3900	40

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

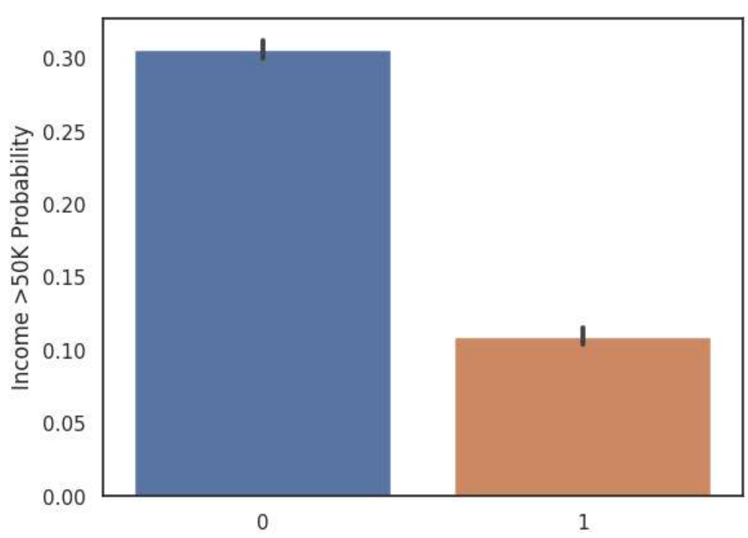
```
1    0
3    0
4    0
Name: income, dtype: category
Categories (2, int64): [0, 1]
```

```
# Splitting the data into train and test
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.30,random_state=99)

X_train.head()
```

	age	fnlwgt	education.num	capital.gain	capital.loss	hours.per.w
24351	42	289636	9	0	0	
15626	37	52465	9	0	0	
4347	38	125933	14	0	0	
23972	44	183829	13	0	0	

```
# Explore Sex vs Income
g = sns.barplot(x="sex",y="income",data=dataset)
g = g.set_ylabel("Income >50K Probability")
plt.show()
```




```
# Split-out Validation Dataset and Create Test Variables
array = dataset.values
X = array[:,0:8]
Y = array[:,8]
print('Split Data: X')
print(X)
```

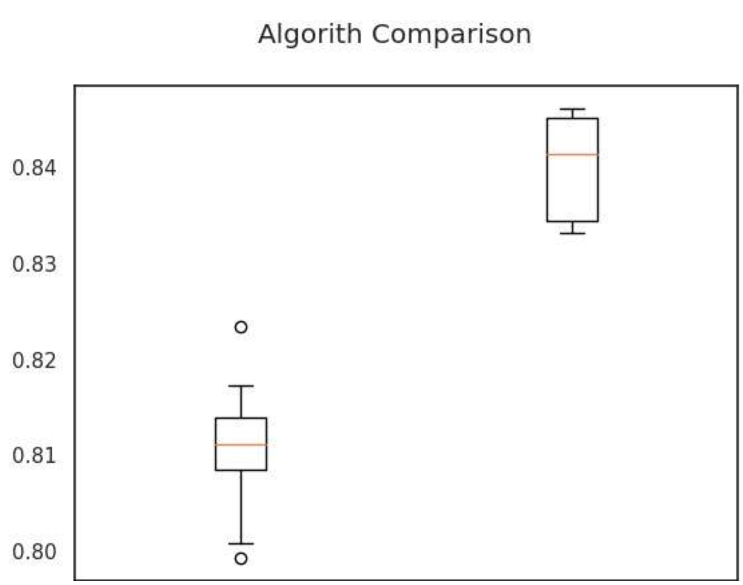
```
print('Split Data: Y')
print(Y)
validation_size = 0.20
seed = 7
num_folds = 10
scoring = 'accuracy'
X_train, X_validation, Y_train, Y_validation = train_test_split(X,Y,
    test_size=validation_size,random_state=seed)

# Params for Random Forest
num_trees = 100
max_features = 3

#Spot Check 5 Algorithms (LR, LDA, KNN, CART, GNB, SVM)
models = []
models.append(('CART', DecisionTreeClassifier()))
models.append(('RF', RandomForestClassifier(n_estimators=num_trees, max_features=max_features)))
#models.append(('SVM', SVC()))
# evalutate each model in turn
results = []
names = []
for name, model in models:
    kfold = KFold(n_splits=10,shuffle=True, random_state=seed)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
Split Data: X
[[    90  77053      9 ...      0   4356     40]
 [    82 132870      9 ...      0   4356     18]
 [    66 186061     10 ...      0   4356     40]
 ...
 [    40 154374      9 ...      0      0     40]
 [    58 151910      9 ...      0      0     40]
 [    22 201490      9 ...      0      0     20]]
Split Data: Y
[0 0 0 ... 1 0 0]
CART: 0.810658 (0.006761)
RF: 0.839988 (0.005096)
```

```
fig = plt.figure()
fig.suptitle('Algorith Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```




```
'''
Commented Out to Reduce Script Time - Took 20 Minutes to run.
```

```
best n_estimator = 250
best max_feature = 5
# Tune Random Forest
n_estimators = np.array([50,100,150,200,250])
max_features = np.array([1,2,3,4,5])
param_grid = dict(n_estimators=n_estimators,max_features=max_features)
model = RandomForestClassifier()
kfold = KFold(n_splits=num_folds, random_state=seed)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring, cv=kfold)
grid_result = grid.fit(X_train, Y_train)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
'''
```

```
'\nCommented Out to Reduce Script Time - Took 20 Minutes to run.\nbest n_
estimator = 250\nbest max_feature = 5\n# Tune Random Forest\nn_estimators
= np.array([50,100,150,200,250])\nmax_features = np.array([1,2,3,4,5])\np
aram_grid = dict(n_estimators=n_estimators,max_features=max_features)\nmo
del = RandomForestClassifier()\nkfold = KFold(n_splits=num_folds, random_
```

```
# 5. Finalize Model
# a) Predictions on validation dataset - KNN
random_forest = RandomForestClassifier(n_estimators=250,max_features=5)
random_forest.fit(X_train, Y_train)
predictions = random_forest.predict(X_validation)
print("Accuracy: %s%%" % (100*accuracy_score(Y_validation, predictions)))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

```
Accuracy: 84.4311377245509%
[[4566  398]
 [ 616  933]]
              precision    recall  f1-score   support

           0       0.88      0.92      0.90      4964
           1       0.70      0.60      0.65      1549

    accuracy                           0.84      6513
   macro avg       0.79      0.76      0.77      6513
weighted avg       0.84      0.84      0.84      6513
```

0s completed at 7:00 AM

Conclusion:

1)The correlation heatmap of the dataset reveals the following observations:

1. Some attributes, such as "age" and "education-num," show a moderate positive correlation, suggesting that higher education levels might be associated with older age.
2. "Capital-gain" and "capital-loss" have a weak correlation with other features, implying that these financial attributes might not be directly influenced by other factors in the dataset.
3. Most features appear to have low correlations with each other, indicating that they provide unique information for prediction.

2)The accuracy, confusion matrix, precision, recall and F1 score obtained.

Accuracy: The model achieved an accuracy of approximately 84.43%, indicating that it accurately predicted the income class for around 84.43% of the samples in the test set.

Confusion Matrix:

1. True Negative (TN): 4566 instances correctly classified as "income = 0."
2. False Positive (FP): 398 instances wrongly classified as "income = 1" when actual was "income = 0."
3. False Negative (FN): 616 instances wrongly classified as "income = 0" when actual was "income = 1."
4. True Positive (TP): 933 instances correctly classified as "income = 1."

Precision and Recall:

1. Precision (income = 0): Around 88% of predictions for "income = 0" were accurate.
2. Precision (income = 1): About 70% of predictions for "income = 1" were accurate.
3. Recall (income = 0): Approximately 92% of actual "income = 0" cases were correctly identified.
4. Recall (income = 1): Roughly 60% of actual "income = 1" cases were correctly identified.

F1-Score:

1. F1-Score (income = 0): Achieved around 0.90, representing the balance between precision and recall for "income = 0."
2. F1-Score (income = 1): Reached approximately 0.65, signifying the balance between precision and recall for "income = 1."

3)In comparing Random Forest and Decision Tree algorithms on the Adult Census Income Dataset:

1. **Accuracy:** Random Forest achieved around 84.43% accuracy, while Decision Tree reached 84% which is lower as compared to random forest.
2. **Confusion Matrix:** Both algorithms showed variations in True Positive, True Negative, False Positive, and False Negative values across classes.
3. **Precision, Recall, F1-Score:** Random Forest balanced precision and recall better, while Decision Tree might emphasize differently.
4. **Generalization:** Random Forest's ensemble approach aids generalization, while Decision Trees can overfit without depth control.
5. **Stability:** Random Forest is more stable due to ensembling, while Decision Trees are sensitive to data variations.
6. **Training Time:** Decision Trees train faster as they're single, whereas Random Forest trains multiple trees, taking longer.