AML Lab Assignment-4:

Name: Gurvinder Kaur Matharu
PRN: 1032230432
Roll No.: PA14

AIM:

Implementation of Ensemble, Random Forests. Analyze the performance

THEORY:

Ensemble methods combine multiple individual models (base learners) to create a stronger, more robust predictive model. By leveraging diverse algorithms or training data subsets, they aim to reduce overfitting, improve generalization, and boost overall predictive performance by aggregating the predictions or decisions of the individual models through techniques like averaging, voting, or weighting. Ensembles, like Random Forest, Gradient Boosting, or AdaBoost, often outperform single models by leveraging the collective wisdom of multiple models.

Random Forests are ensemble learning methods that create multiple decision trees during training. They operate by aggregating predictions from each tree and relying on the voting or averaging of these outputs to make final predictions. Random Forests reduce overfitting by introducing randomness in the feature selection and bootstrapping of data samples for each tree, resulting in robust and accurate predictions.

Here, decision trees, random forest and ensemble classifier have been implemented and their performance was compared.

CODE EXECUTION & OUTPUT:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_auc_score, roc_curve, auc
from sklearn.preprocessing import LabelEncoder
```

```
data = pd.read_csv('/content/Employee.csv')
```

```
data.head()
```

	Education	JoiningYear	City	PaymentTier	Age	Gender	EverBenched	ExperienceInCurrentDomain
0	Bachelors	2017	Bangalore	3	34	Male	No	0
1	Bachelors	2013	Pune	1	28	Female	No	3
2	Bachelors	2014	New Delhi	3	38	Female	No	2
3	Masters	2016	Bangalore	3	27	Male	No	5
4	Masters	2017	Pune	3	24	Male	Yes	2

```
data.columns
```

```
Index(['Education', 'JoiningYear', 'City', 'PaymentTier', 'Age', 'Gender',
       'EverBenched', 'ExperienceInCurrentDomain', 'LeaveOrNot'],
      dtype='object')
```

```
data.describe()
```

	JoiningYear	PaymentTier	Age	ExperienceInCurrentDomain	LeaveOrNot
count	4653.000000	4653.000000	4653.000000	4653.000000	4653.000000
mean	2015.062970	2.698259	29.393295	2.905652	0.343864
std	1.863377	0.561435	4.826087	1.558240	0.475047
min	2012.000000	1.000000	22.000000	0.000000	0.000000
25%	2013.000000	3.000000	26.000000	2.000000	0.000000
50%	2015.000000	3.000000	28.000000	3.000000	0.000000
75%	2017.000000	3.000000	32.000000	4.000000	1.000000
max	2018.000000	3.000000	41.000000	7.000000	1.000000

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4653 entries, 0 to 4652
Data columns (total 9 columns):
 #   Column                     Non-Null Count  Dtype 
---  ------                     --------------  ----- 
 0   Education                  4653 non-null   object
 1   JoiningYear                4653 non-null   int64 
 2   City                       4653 non-null   object
 3   PaymentTier                4653 non-null   int64 
 4   Age                        4653 non-null   int64 
 5   Gender                     4653 non-null   object
 6   EverBenched                4653 non-null   object
 7   ExperienceInCurrentDomain  4653 non-null   int64 
 8   LeaveOrNot                 4653 non-null   int64 
dtypes: int64(5), object(4)
memory usage: 327.3+ KB
```

```
data.shape
```

```
(4653, 9)
```

```
data.isnull().sum()
```

```
Education                    0
JoiningYear                  0
City                         0
PaymentTier                  0
Age                          0
Gender                       0
EverBenched                  0
ExperienceInCurrentDomain    0
LeaveOrNot                   0
dtype: int64
```

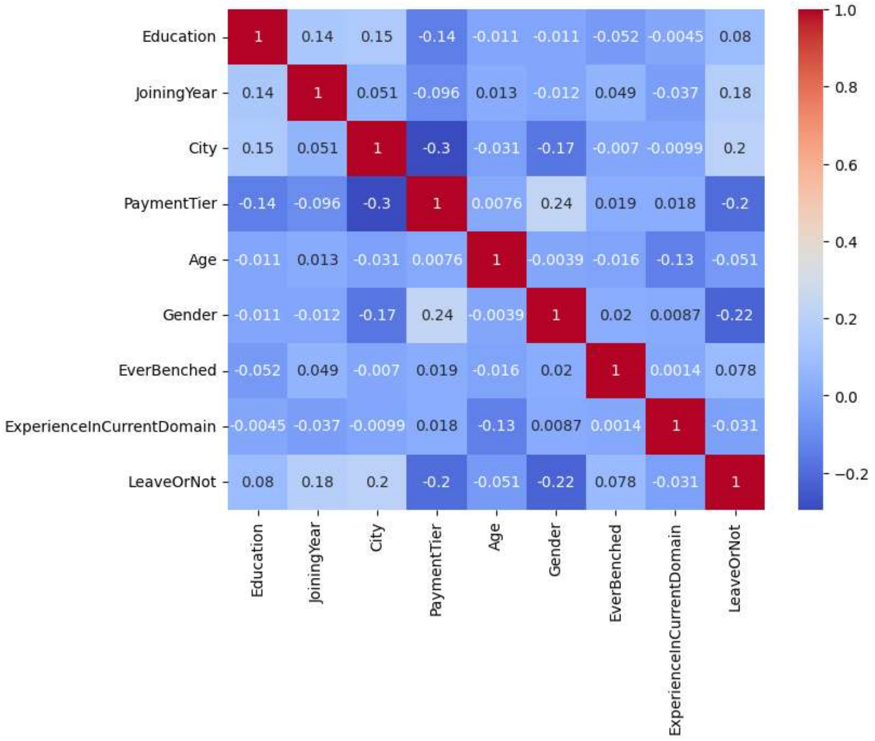
```
encoder = LabelEncoder()
```

```
data['City'] = encoder.fit_transform(data['City'])
data['Education'] = encoder.fit_transform(data['Education'])
data['Gender'] = encoder.fit_transform(data['Gender'])
data['EverBenched'] = encoder.fit_transform(data['EverBenched'])
```

```
data.head()
```

	Education	JoiningYear	City	PaymentTier	Age	Gender	EverBenched	ExperienceInCurrentDomain	Leav
0	0	2017	0	3	34	1	0	0	
1	0	2013	2	1	28	0	0	3	
2	0	2014	1	3	38	0	0	2	
3	1	2016	0	3	27	1	0	5	
4	1	2017	2	3	24	1	1	2	

```
corr = data.corr()
plt.figure(figsize=(8,6))
sns.heatmap(corr, cmap = 'coolwarm', annot=True)
plt.show()
```



```
X = data.drop("LeaveOrNot", axis=1)
y = data["LeaveOrNot"]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

```
tree_clf = DecisionTreeClassifier(random_state=42)
tree_clf.fit(X_train, y_train)
rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)
rf_clf.fit(X_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
ensemble_clf = VotingClassifier(estimators=[
    ('decision_tree', tree_clf),
    ('random_forest', rf_clf)
], voting='hard')

ensemble_clf.fit(X_train, y_train)
```

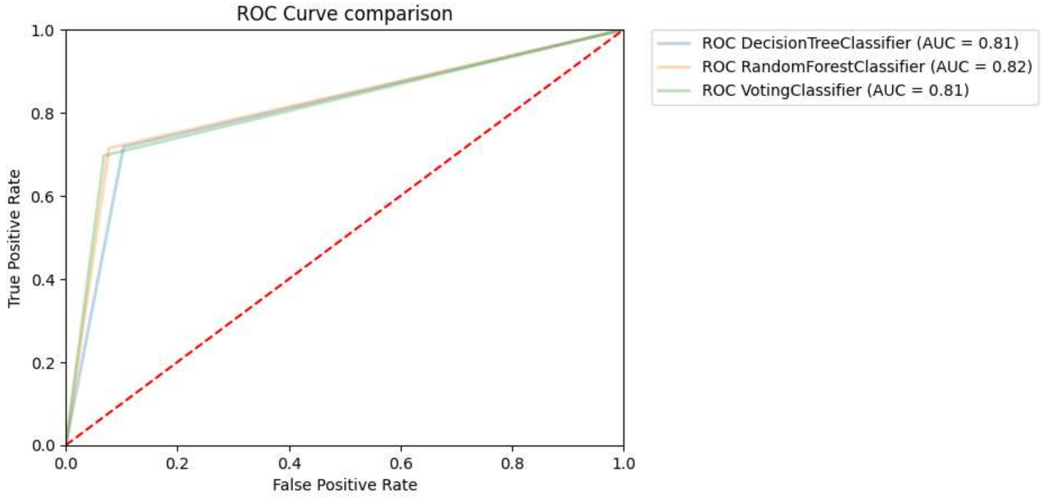
```
▸ VotingClassifier
   decision_tree               random_forest
▸ DecisionTreeClassifier    ▸ RandomForestClassifier
```

```
tree_pred = tree_clf.predict(X_test)
rf_pred = rf_clf.predict(X_test)
ensemble_pred = ensemble_clf.predict(X_test)
print(f"Decision Tree Accuracy: {accuracy_score(y_test, tree_pred)}")
print(f"Random Forest Accuracy: {accuracy_score(y_test, rf_pred)}")
print(f"Ensemble Accuracy: {accuracy_score(y_test, ensemble_pred)}")
```

```
Decision Tree Accuracy: 0.8367697594501718
Random Forest Accuracy: 0.8530927835051546
Ensemble Accuracy: 0.8530927835051546
```

```
MLA = [
    tree_clf,
    rf_clf,
    ensemble_clf
    ]
```

```
index = 1
for alg in MLA:
    predicted = alg.fit(X_train, y_train).predict(X_test)
    fp, tp, th = roc_curve(y_test, predicted)
    roc_auc_mla = auc(fp, tp)
    MLA_name = alg.__class__.__name__
    plt.plot(fp, tp, lw=2, alpha=0.3, label='ROC %s (AUC = %0.2f)'  % (MLA_name, roc_auc_mla))
    index+=1
plt.title('ROC Curve comparison')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.plot([0,1],[0,1],'r--')
plt.xlim([0,1])
plt.ylim([0,1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```


ROC Curve comparison
True Positive Rate
False Positive Rate
ROC DecisionTreeClassifier (AUC = 0.81)
ROC RandomForestClassifier (AUC = 0.82)
ROC VotingClassifier (AUC = 0.81)