UCS 2612 Machine Learning Laborotory
Assignment 8

Name : Mega V
Reg No : 3122 21 5001 051

AIM :

To develop a python program to diagnose breast cancer using Ensemble Models and to Visualize the features from the dataset and interpret the results obtained by the model using Matplotlib library.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from scipy import stats
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_curve, auc
```

LOADING THE DATASET :

```
df = pd.read_csv("data.csv")
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	t
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	

5 rows × 33 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
```

```
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

```
df.describe()
```

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetr
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.

8 rows × 32 columns

□ PREPROCESSING :

```
# Null Values :

print(df.isnull().sum())
```

```
id                           0
diagnosis                    0
radius_mean                  0
texture_mean                 0
perimeter_mean               0
area_mean                    0
smoothness_mean              0
compactness_mean             0
concavity_mean               0
concave points_mean          0
symmetry_mean                0
fractal_dimension_mean       0
radius_se                    0
texture_se                   0
perimeter_se                 0
area_se                      0
smoothness_se                0
compactness_se               0
concavity_se                 0
concave points_se            0
symmetry_se                  0
fractal_dimension_se         0
radius_worst                 0
texture_worst                0
perimeter_worst              0
area_worst                   0
smoothness_worst             0
compactness_worst            0
concavity_worst              0
concave points_worst         0
symmetry_worst               0
fractal_dimension_worst      0
Unnamed: 32                569
dtype: int64
```

```
# Removing Unamed: 32 column as its completely NaN :

df.drop(df.columns[-1], axis=1, inplace=True)
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	r
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	

5 rows × 32 columns

```
# Converting categorical to numeric values :

df["diagnosis"] = df["diagnosis"].map({"M" : 0, "B": 1})
```

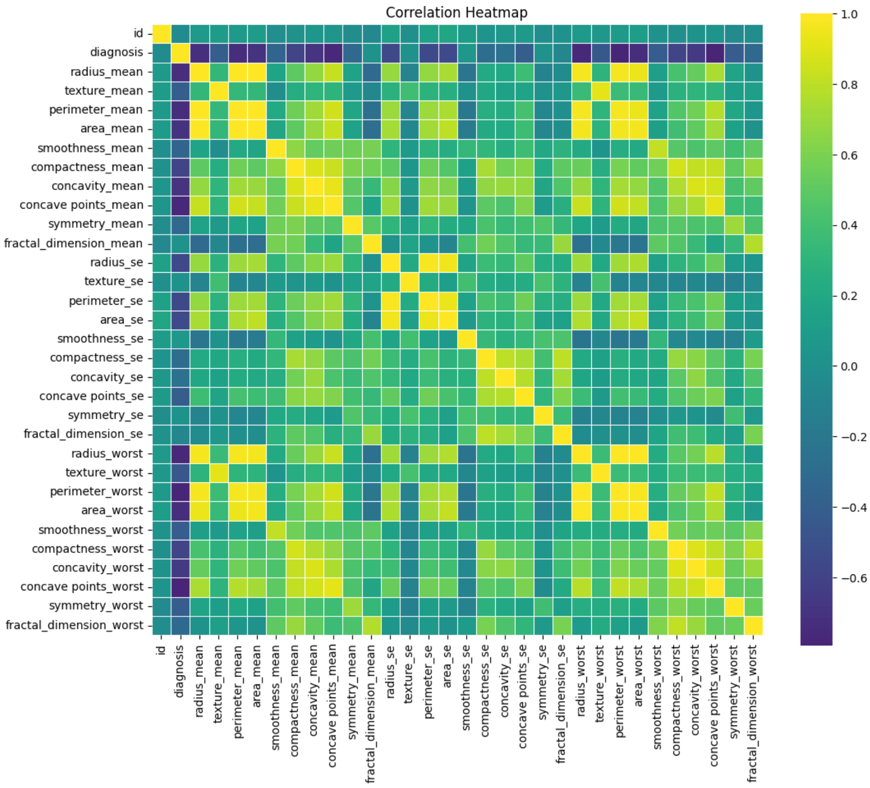
```
# Standardisation :

numeric_cols = df.columns[2:]

standardiser = StandardScaler()
df[numeric_cols] = standardiser.fit_transform(df[numeric_cols])
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	r
0	842302	0	1.097064	-2.073335	1.269934	0.984375	1.568466	3.283515	2.652874	2.532475	...	
1	842517	0	1.829821	-0.353632	1.685955	1.908708	-0.826962	-0.487072	-0.023846	0.548144	...	
2	84300903	0	1.579888	0.456187	1.566503	1.558884	0.942210	1.052926	1.363478	2.037231	...	
3	84348301	0	-0.768909	0.253732	-0.592687	-0.764464	3.283553	3.402909	1.915897	1.451707	...	
4	84358402	0	1.750297	-1.151816	1.776573	1.826229	0.280372	0.539340	1.371011	1.428493	...	

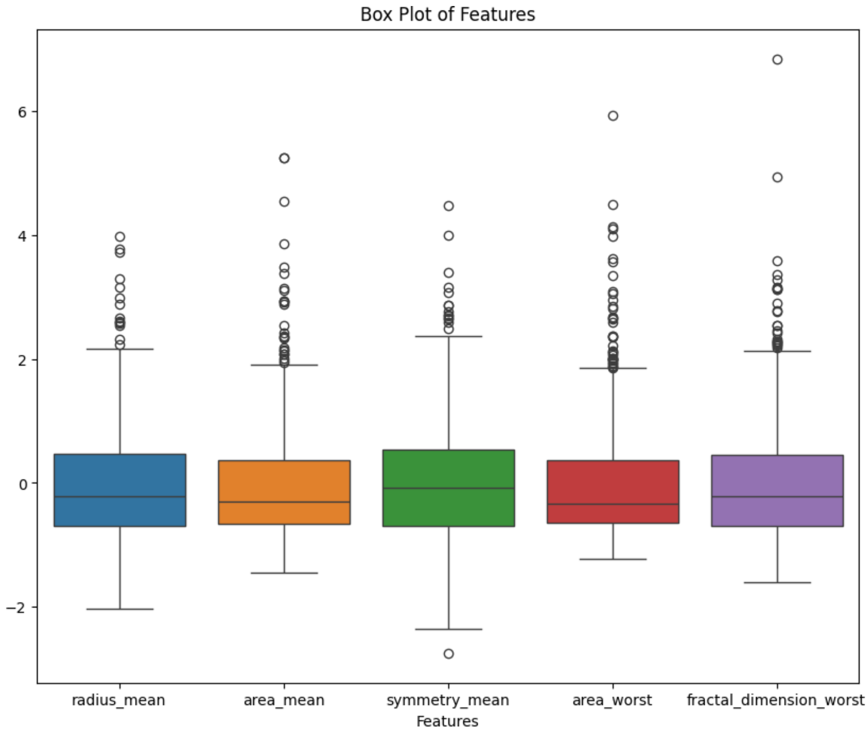
5 rows × 32 columns

▯ EXPLORATORY DATA ANALYSIS :

```
# Heat Map :

corr_matrix = df.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, cmap='viridis', center=0, square=True, linewidths=.5)
plt.title('Correlation Heatmap')
plt.show()
```



```
# Box Plot :

plt.figure(figsize=(10, 8))
sns.boxplot(data=df[["radius_mean", "area_mean", "symmetry_mean", "area_worst", "fractal_dimension_worst"]])
plt.title('Box Plot of Features')
plt.xlabel('Features')
plt.show()
```




```
print(df.shape)
```

```
(569, 32)
```

```
# Outlier Detection :

z_scores = stats.zscore(df)
threshold = 4
outlier_mask = (z_scores > threshold) | (z_scores < -threshold)
df = df[~outlier_mask.any(axis=1)]
```

```
print(df.shape)
```

```
(526, 32)
```

□ TRAINING AND TESTING SPLIT :

```
X = df.drop("diagnosis", axis = 1)
Y = df["diagnosis"]

X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state = 42)
```

□ RANDOM FOREST MODEL :

```
rf_classifier = RandomForestClassifier(n_estimators=20, random_state=42)
rf_classifier.fit(X_train, y_train)
y_pred = rf_classifier.predict(X_test)

train_accuracy = accuracy_score(y_train, rf_classifier.predict(X_train))
test_accuracy = accuracy_score(y_test, y_pred)

print("************************************ Random Forest ********************************")
print("\nTraining Accuracy:", round(train_accuracy, 4))
print("Test Accuracy:", round(test_accuracy, 4))
```

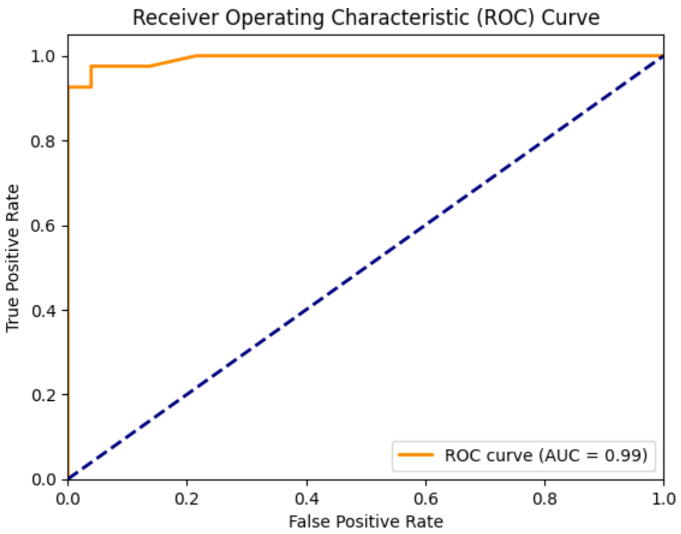
```
************************************ Random Forest ********************************

Training Accuracy: 0.9924
```

```
Test Accuracy: 0.9697
```

```
# Cross Validation :

cv_scores = cross_val_score(rf_classifier, X, Y, cv=5)
print("Cross-Validation Score:", round(cv_scores.mean(), 4))
```

```
Cross-Validation Score: 0.9468
```

```
# ROC Curve :

y_pred_proba = rf_classifier.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```



As the cross validation score is 94%, the Random Forest ensemble model is not overfitting and hence performing better with an training accuracy of 99% and testing accuracy of 97%

ADABOOST ENSEMBLE MODEL :

```
adaboost_classifier = AdaBoostClassifier(n_estimators=20, random_state=42)
adaboost_classifier.fit(X_train, y_train)
y_pred = adaboost_classifier.predict(X_test)

train_accuracy = adaboost_classifier.score(X_train, y_train)
test_accuracy = accuracy_score(y_test, y_pred)

print("******************************************* ADABOOST **************************************")
print("\nTraining Accuracy:", round(train_accuracy, 4))
print("Testing Accuracy:", round(test_accuracy, 4))
```

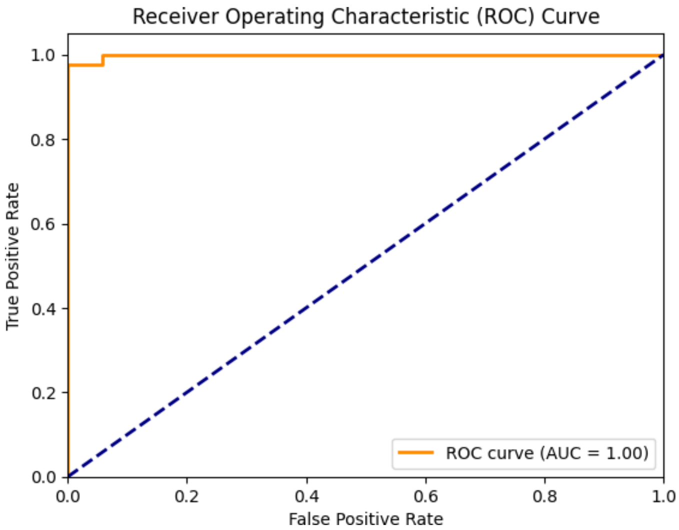
```
******************************************* ADABOOST **************************************

Training Accuracy: 1.0
Testing Accuracy: 0.9697
c:\Python312\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:519: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will
  warnings.warn(
```

```
# Cross Validation :

cv_scores = cross_val_score(adaboost_classifier, X, Y, cv=5)
print("Cross-Validation Score:", round(cv_scores.mean(), 4))
```

```
c:\Python312\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:519: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will
  warnings.warn(
c:\Python312\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:519: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will
  warnings.warn(
c:\Python312\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:519: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will
```

```
    warnings.warn(
c:\Python312\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:519: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will
    warnings.warn(
c:\Python312\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:519: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will
    warnings.warn(
Cross-Validation Score: 0.9639
```

```
# ROC Curve :

y_pred_proba = adaboost_classifier.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```



As the cross validation score is 96%, the Adaboost ensemble model is not overfitting and hence performing better with an training accuracy of 100% and testing accuracy of 96%

```
'''
                          Training Accuracy          Testing Accuracy            Cross Validation Score            AUC

Random Forest Model :          99%                        97%                          95%                         99%

Adaboost            :          100%                       96%                          96%                         100%

'''
```

```
'\n                          Training Accuracy          Testing Accuracy            Cross Validation Score            AUC\n\nRandom
Forest Model :          99%                        97%                          95%                         99%\n\nAdaboost
:              100%                       96%                          96%                         100%\n\n'
```

Both models are performing good without overfitting, But Adaboost is slightly better in training whereas Random forest is slightly better in testing phase. Overall, Adaboost has a slight edge over Random Forest in terms of training accuracy and cross validation score making it perform better for unseen data.

Learning Outcome :

1. Interpret the results and evaluate the performance of random forest models.
2. Implement and tune random forest algorithms for predictive modeling tasks. Apply executable techniques to improve the performance of machine learning models.