Report

Step 1: Importing the Libraries

First step is to include importing the libraries which are NumPy, Pandas and Matplotlib.

- **import numpy as np**
- **import matplotlib.pyplot as plt**
- **import pandas as pd**
- **from sklearn.metrics import confusion_matrix, f1_score,accuracy_score,precision_recall_curve,precision_score,plot_precision_recall_curve,recall_score**

Step 2: Importing the dataset

In this step, we are Procuring data from the data set using panda's method read.csv() and storing it in the variable called dataset. Then we are assigning the corresponding variables to X and y.

dataset = pd.read_csv('Social_Network_Ads.csv')

X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

```
dataset = pd.read_csv('Social_Network_Ads.csv')
```

```
dataset.head()
```

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0
3	27	57000	0
4	19	76000	0

```
# Splitting the dataset features into X and y
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

Step 3: Splitting the dataset into the Training set and Test set

Here we are splitting the data into training and the test set. In this, we have the test_size =0.25, which denotes we are taking 25% of data to keep as a Test set and the remaining 75% of data will be used for Training as the training set.

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

Step 4: Feature Scaling

As we reduce the range of X's values to a lower value, this extra step will speed up the program. Here, we reduce the X train and X test to a narrow range of -2 to +2. As an illustration, a salary of 75000 is scaled down to 0.16418997.

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Step 5: Training the Random Forest Classification model on the Training Set

We may import the RandomForestClassifier Class and fit the training set to our model once the training test is prepared. The class SVC is assigned to the variable classifier. Entropy is the criterion employed in this case. The model is subsequently trained using the classifier.fit() function.

```
# Training the Random Forest Classification model on the Training set
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
```

```
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=0)
```

Step 6: Predicting the Test set results

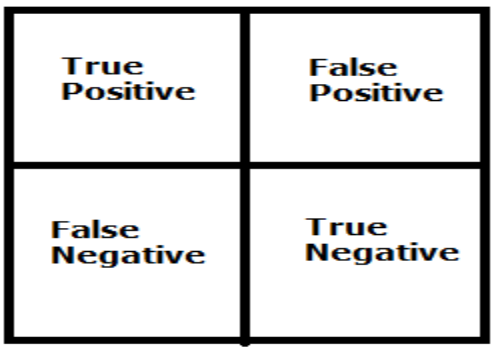
The values for the Test set are predicted in this phase using the classifier.predict() method, and they are then saved in the variable y pred.

```
# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(type(X_test))
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

Step 7: Confusion Matrix and Accuracy

Most of the classification techniques involve this stage. This shows the trained model's accuracy and plots the confusion matrix.

When the true values of the Test Set are known, a table called the confusion matrix is used to display the proportion of correct and incorrect predictions on a classification task. It follows that format.



The True values are the number of correct predictions made. We got an accuracy of 91%. In the test set.

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_test=accuracy_score(y_test, y_pred)
accuracy_test
```

```
[[63  5]
 [ 4 28]]
```

```
]: 0.91
```

```
y_pred_train = classifier.predict(X_train)
accuracy_train=accuracy_score(y_train, y_pred_train)
accuracy_train
```

```
]: 0.98
```

Step 8: Comparing the Real Values with Predicted Values

To compare the categorized values of the original Test set (y test) and the predicted results (y pred), a Pandas DataFrame is constructed in this stage.

This is for checking the accuracy of the predicted value and trying to spot the value which was incorrectly predicted.

```
df = pd.DataFrame({'Real Values':y_test, 'Predicted Values':y_pred})
df
```

]:

	Real Values	Predicted Values
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
...	...	...
95	1	0
96	0	0
97	1	1
98	1	1
99	1	1

100 rows × 2 columns

Step 9: Precision result for Train and Test data set.

Precision:

Precision is the ratio of the number of accurately identified members of a class to the total number of times the class was predicted by the model.

```
#calculating precision
precision=precision_score(y_train, y_pred_train)
print(precision)
```

```
0.981651376146789
```

```
precision1=precision_score(y_test, y_pred)
print(precision1)
```

```
0.8484848484848485
```

Step 10: F1 Score result for a Train set

F1 score combines precision and recalls into one statistic, it is a little less obvious. F1 will be high if precision and recall are both high. F1 will be low if they are both low. F1 will be low if one is high and the other is low. F1 is a fast technique to determine whether the classifier is genuinely effective at recognizing class members.

```
f1_score=f1_score(y_train, y_pred_train)
f1_score
```

```
0.9727272727272728
```

```
f1_score1=f1_score(y_test, y_pred)
f1_score1
```

```
0.8615384615384615
```

Step 9: Visualizing the Results.

Here we visualize the results of the Random Forest classification model on a graph that is plotted along with the two regions.

a) Visualizing Train set

b) Visualizing Test set

```
# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
    plt.title('Random Forest Classification (Test set)')
    plt.xlabel('Age')
    plt.ylabel('Estimated Salary')
    plt.legend()
    plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence
in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a single row
if you intend to specify the same RGB or RGBA value for all points.
```