# DSBDAL PRACTICAL :- 5

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd

In [7]: dataset = pd.read_csv('C:/Users/HP/Downloads/Social_Network_Ads.csv')

In [8]: dataset.head()
```

Out[8]:

| | User ID | Gender | Age | EstimatedSalary | Purchased |
|---|---|---|---|---|---|
| 0 | 15624510 | Male | 19 | 19000 | 0 |
| 1 | 15810944 | Male | 35 | 20000 | 0 |
| 2 | 15668575 | Female | 26 | 43000 | 0 |
| 3 | 15603246 | Female | 27 | 57000 | 0 |
| 4 | 15804002 | Male | 19 | 76000 | 0 |

```
In [34]: x = dataset.iloc[:,2:4]
         y = dataset.iloc[:,4]

In [35]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, ranc

In [36]: scale = StandardScaler()
         x_train = scale.fit_transform(x_train)
         x_test = scale.transform(x_test)

In [37]: lr = LogisticRegression(random_state = 0,solver = 'lbfgs')
         lr.fit(x_train,y_train)
         pred = lr.predict(x_test)

         print(x_test[:10])
         print('-'*15)
         print(pred[:10])
```

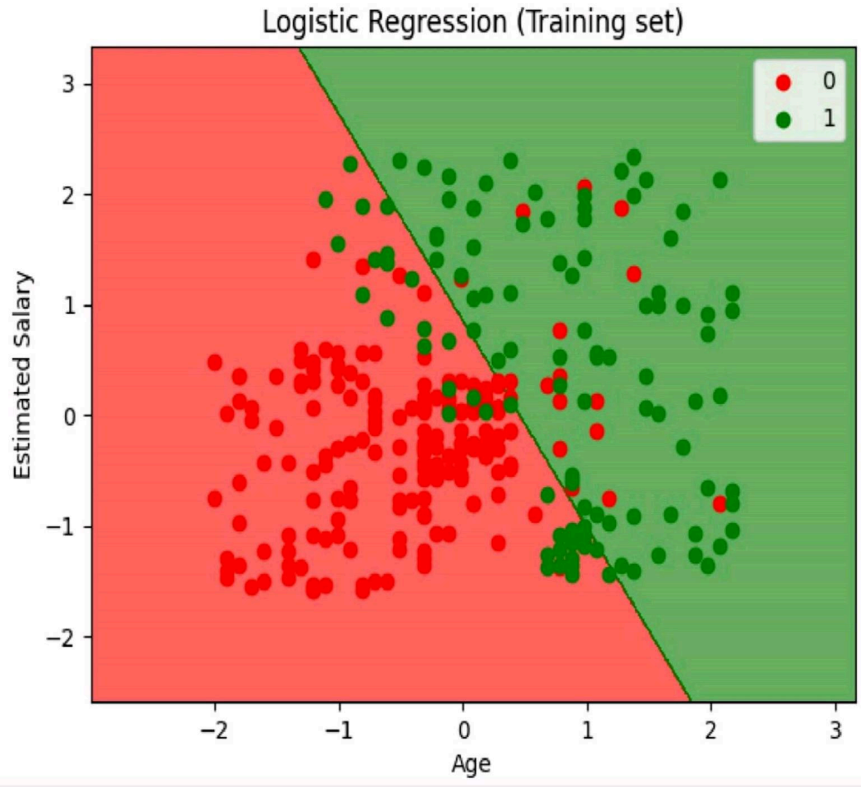
```
[[ 0.812419   -1.39920777]
 [ 2.0889839   0.52871943]
 [-0.95513241 -0.75656537]
 [ 1.0088136   0.76240757]
 [-0.85693511 -1.22394166]
 [-0.75873781 -0.23076704]
 [ 0.9106163   1.08372877]
 [-0.85693511  0.38266434]
 [ 0.2232352   0.14897619]
 [ 0.4196298  -0.14313399]]
---------------
[0 1 0 1 0 0 1 0 0 0]
```

```
In [38]: print('Expected Output:',pred[:10])
         print('-'*15)
         print('Predicted Output:\n',y_test[:10])
```

```
Expected Output: [0 1 0 1 0 0 1 0 0 0]
---------------
Predicted Output:
 209    0
280    1
33     0
210    1
93     0
84     0
329    1
94     0
266    0
126    0
Name: Purchased, dtype: int64
```

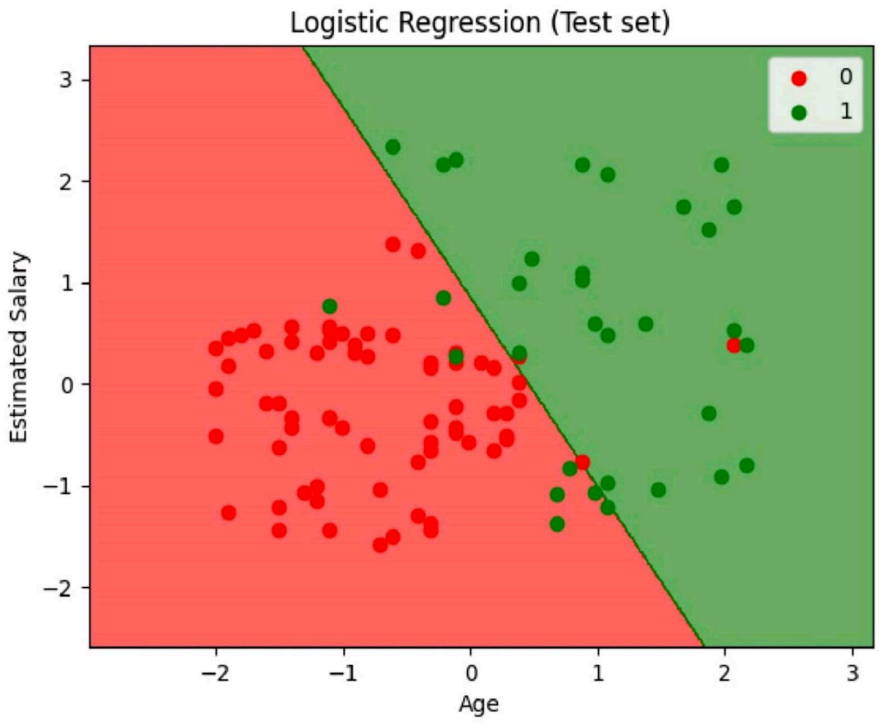
In [17]:
```
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:,
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:,
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).
             alpha = 0.6, 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('Logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_2784\2759427564.py:10: UserWarning:
*c* argument looks like a single numeric RGB or RGBA sequence, which should b
e avoided as value-mapping will have precedence in case its length matches wi
th *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.
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```



In [18]:
```
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:,
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:,
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).
             alpha = 0.6, 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('Logistic Regression (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_2784\238990405.py:10: UserWarning: *
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 wit
h a single row if you intend to specify the same RGB or RGBA value for all po
ints.
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```



In [42]:
```
print('\nAccuracy: {:.2f}'.format(accuracy_score(y_test,pred)))
print('Error Rate: ',(fp+fn)/(tp+tn+fn+fp))
print('Sensitivity (Recall or True positive rate) :',tp/(tp+fn))
print('Specificity (True negative rate) :',tn/(fp+tn))
print('Precision (Positive predictive value) :',tp/(tp+fp))
print('False Positive Rate :',fp/(tn+fp))
```

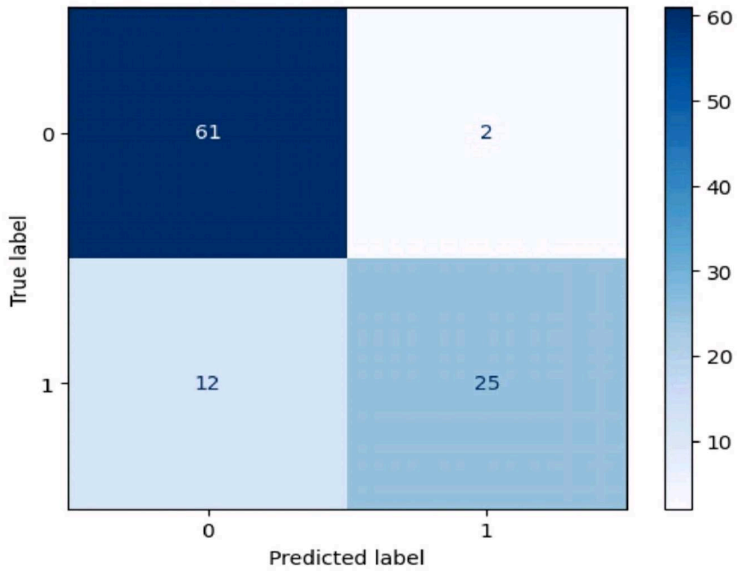
```
Accuracy: 0.86
Error Rate:  0.14
Sensitivity (Recall or True positive rate) : 0.6756756756756757
Specificity (True negative rate) : 0.9682539682539683
Precision (Positive predictive value) : 0.9259259259259259
False Positive Rate : 0.031746031746031744
```

In [39]:
```
matrix = confusion_matrix(y_test,pred,labels = lr.classes_)
print(matrix)

tp, fn, fp, tn = confusion_matrix(y_test,pred,labels=[1,0]).reshape(-1)
```

```
[[61  2]
 [12 25]]
```

In [40]:
```
conf_matrix = ConfusionMatrixDisplay(confusion_matrix=matrix,display_labels=lr
conf_matrix.plot(cmap=plt.cm.Blues)
plt.show()
```




In [41]:
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.97      0.90        63
           1       0.93      0.68      0.78        37

    accuracy                           0.86       100
   macro avg       0.88      0.82      0.84       100
weighted avg       0.87      0.86      0.85       100
```