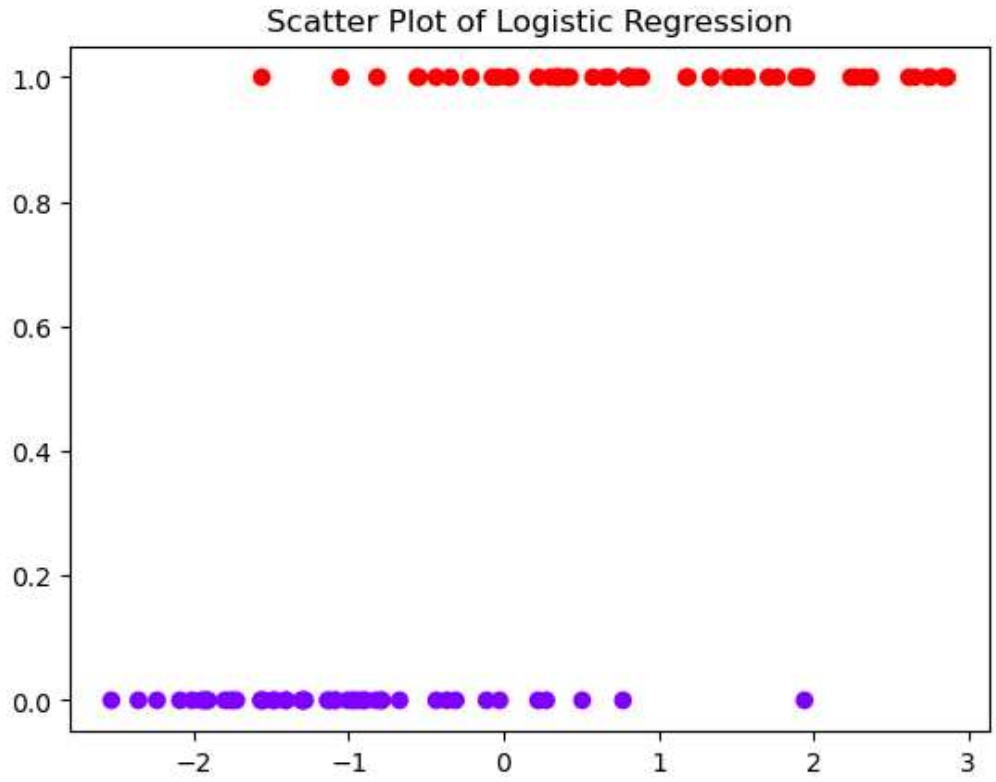
DSBDA Lab Assignment No. 5

Name: Akash Ganesh Padir
Roll No.: TEB04

In [1]:
```
#Step 1: Import the required modules
from sklearn.datasets import make_classification
from matplotlib import pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import pandas as pd
```

In [2]:
```
#Step 2: Generate the dataset
# Generate and dataset for Logistic Regression
x, y = make_classification(
n_samples=100,
n_features=1,
n_classes=2,
n_clusters_per_class=1,
flip_y=0.03,
n_informative=1,
n_redundant=0,
n_repeated=0
)
```

In [3]:
```
#Step 3: Visualize the Data
# Create a scatter plot
plt.scatter(x, y, c=y, cmap='rainbow')
plt.title('Scatter Plot of Logistic Regression')
plt.show()
```



In [4]:
```
#Step 4: Split the Dataset
# Split the dataset into training and test dataset
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=1)
```

In [5]:
```
#Step 5: Perform Logistic Regression
# Create a Logistic Regression Object, perform Logistic Regression
log_reg = LogisticRegression()
log_reg.fit(x_train, y_train)
```

Out[5]:
```
LogisticRegression()
```

In [6]:
```
# Show to Coeficient and Intercept
print(log_reg.coef_)
print(log_reg.intercept_)
```

```
[[1.77558569]]
[0.80572973]
```

In [7]:
```
#Step 6: Make prediction using the model
# Perform prediction using the test dataset
y_pred = log_reg.predict(x_test)
```

In [8]:
```
y_pred
```

Out[8]:
```
array([1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1,
       1, 1, 0])
```

In [9]:
```
#Step 7: Display the Confusion Matrix
#The confusion matrix helps you to see how the model performed. It tells you the number o
#false positives and false negatives. To see the confusion matrix, use:
# Show the Confusion Matrix
confusion_matrix(y_test, y_pred)
```

Out[9]:
```
array([[12,  6],
       [ 1,  6]], dtype=int64)
```

In [10]:
```
from sklearn.metrics import accuracy_score
print ("Accuracy : ", accuracy_score(y_test, y_pred))
```

```
Accuracy :  0.72
```

1 Implement logistic regression using Python/R to perform classification on Social_Network_Ads.csv dataset. Let's make the Logistic Regression model, predicting whether a user will purchase the product or not.

In [11]:
```
import pandas as pd
import math
import numpy as np
import seaborn as sns
from sklearn.datasets import make_classification
from matplotlib import pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
#from sklearn.metrics import preprocessing
from sklearn.metrics import classification_report
```

In [13]:
```
dataset =pd.read_csv('Social_Network_Ads.csv')
```

In [14]:
```
dataset
```

Out[14]:

	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
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

In [15]:
```
dataset.head()
```

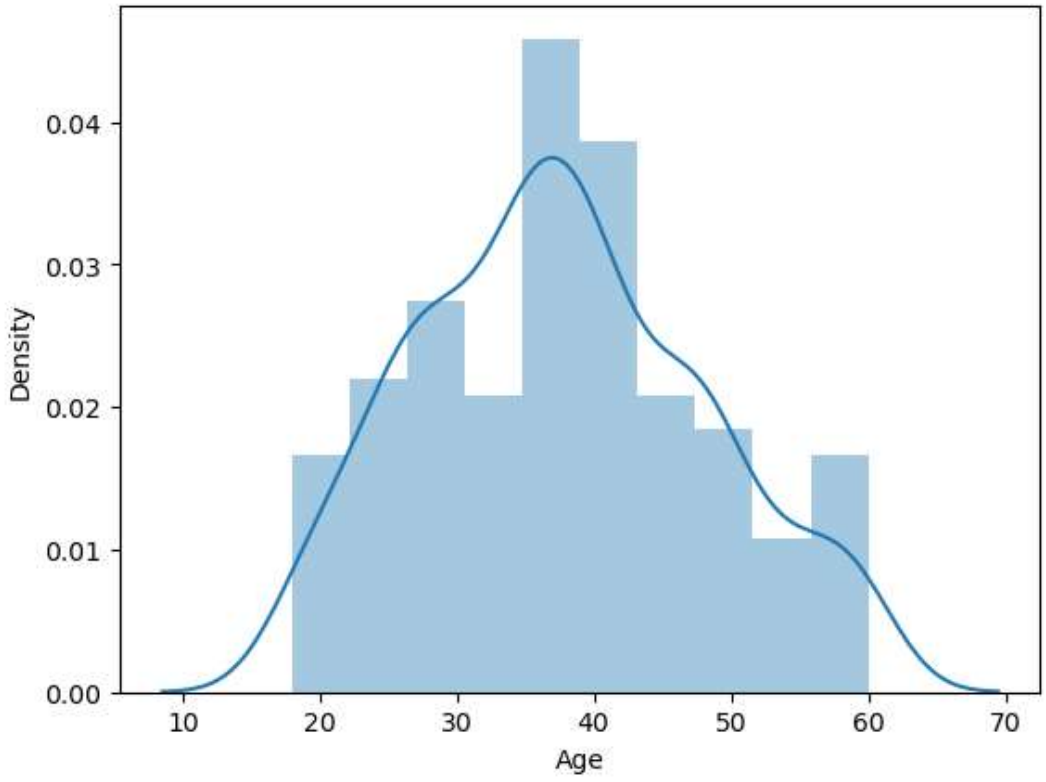
Out[15]:

	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 [16]:
```
sns.distplot(dataset['Age'])
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarnin
g: `distplot` is a deprecated function and will be removed in a future version. Please
adapt your code to use either `displot` (a figure-level function with similar flexibili
ty) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

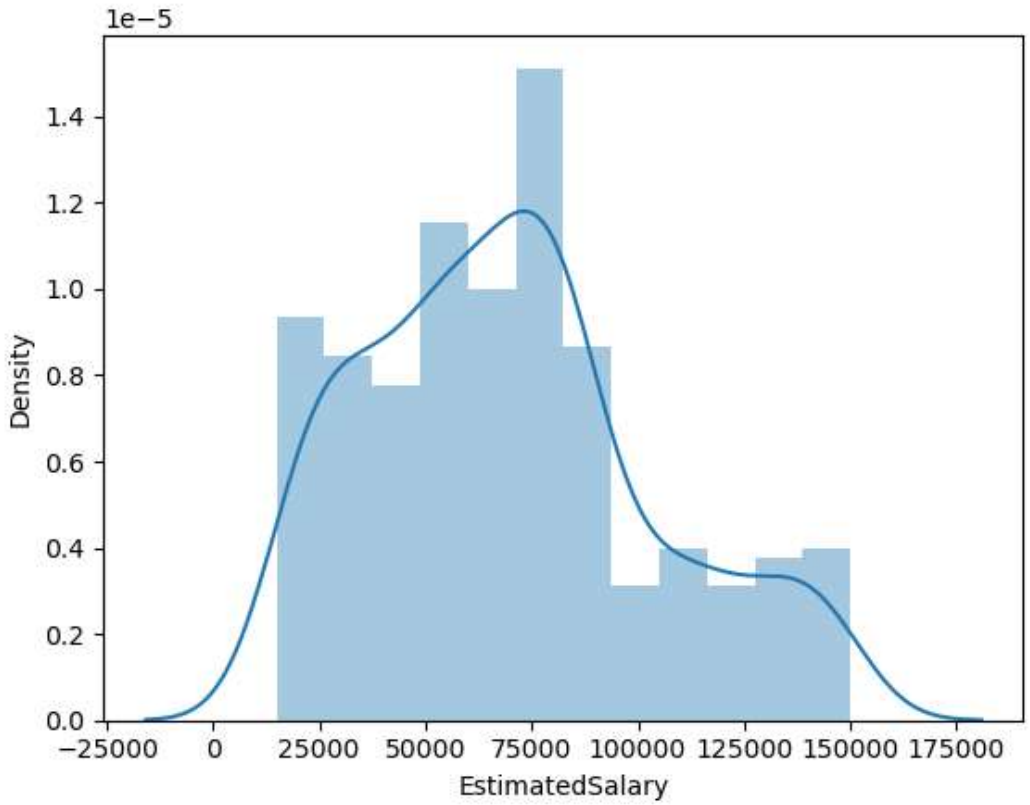
Out[16]: <AxesSubplot:xlabel='Age', ylabel='Density'>



In [17]:
```
sns.distplot(dataset['EstimatedSalary'])
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarnin
g: `distplot` is a deprecated function and will be removed in a future version. Please
adapt your code to use either `displot` (a figure-level function with similar flexibili
ty) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[17]: <AxesSubplot:xlabel='EstimatedSalary', ylabel='Density'>



In [18]:
```
# input
x = dataset.iloc[:, [2, 3]].values
# output
y = dataset.iloc[:, 4].values
```

In [19]:
```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.25, random_state = 0
```

In [20]:
```
from sklearn.preprocessing import StandardScaler
sc_x = StandardScaler()
xtrain = sc_x.fit_transform(xtrain)
xtest = sc_x.transform(xtest)
print (xtrain[0:10, :])
```

```
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761  1.89663484]
 [ 1.37390747 -1.40858358]
 [ 1.47293972  0.99784738]
 [ 0.08648817 -0.79972756]
 [-0.01254409 -0.24885782]
 [-0.21060859 -0.5677824 ]
 [-0.21060859 -0.19087153]]
```

In [21]:
```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(xtrain, ytrain)
```

Out[21]:
```
LogisticRegression(random_state=0)
```

In [22]:
```
y_pred = classifier.predict(xtest)
```

In [23]:
```
print(classification_report(ytest,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.96      0.92        68
           1       0.89      0.75      0.81        32

    accuracy                           0.89       100
   macro avg       0.89      0.85      0.87       100
weighted avg       0.89      0.89      0.89       100
```

In [24]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(ytest, y_pred)
print ("Confusion Matrix : \n", cm)
```

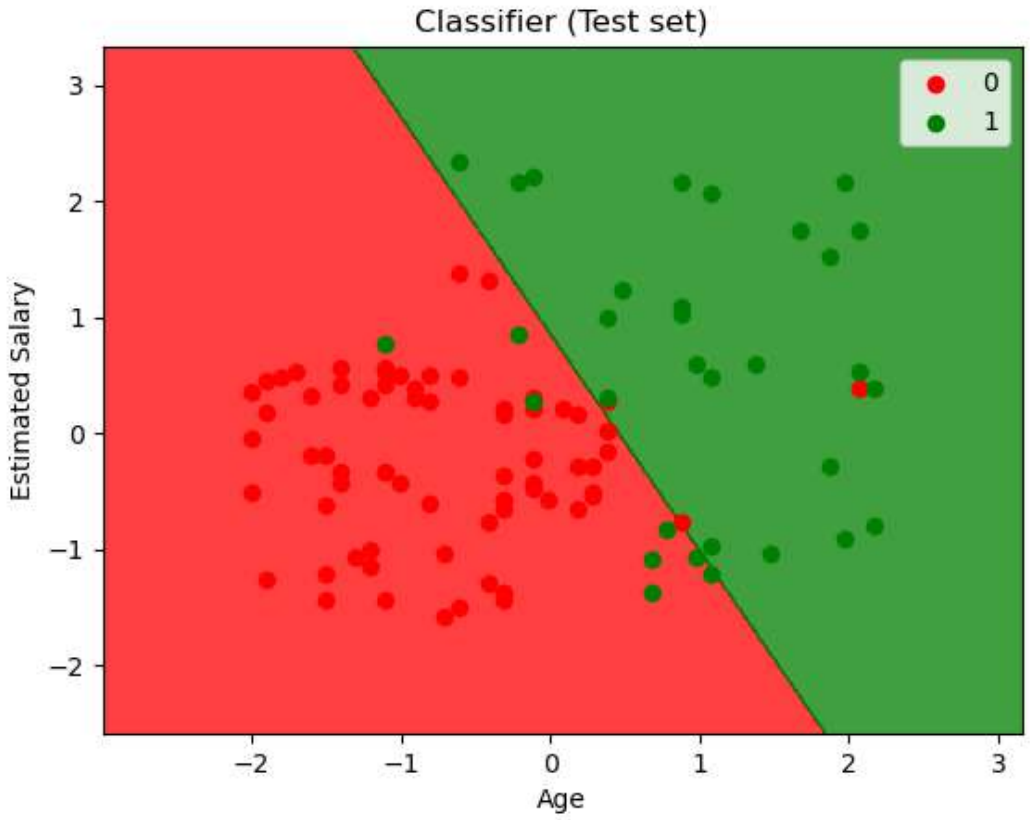
```
Confusion Matrix :
 [[65  3]
 [ 8 24]]
```

In [25]:
```
from sklearn.metrics import accuracy_score
print ("Accuracy : ", accuracy_score(ytest, y_pred))
```

```
Accuracy :  0.89
```

In [28]:
```
# Visualizing the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = xtest, ytest
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,
                               stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1,
                               stop = X_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(
    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('Classifier (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*.  Pleas
e use the *color* keyword-argument or provide a 2D array with a single row if you inten
d to specify the same RGB or RGBA value for all points.
*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*.  Pleas
e use the *color* keyword-argument or provide a 2D array with a single row if you inten
d to specify the same RGB or RGBA value for all points.
```



In [29]:
```
#Different Method
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset =pd.read_csv('Social_Network_Ads.csv')
dataset.head()
```

Out[29]:

	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 [31]:
```
#Different Method
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset =pd.read_csv('Social_Network_Ads.csv')
dataset.head()
```

Out[31]:

	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 [32]:
```
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values
print(X[:3, :])
print('-'*15)
print(y[:3])
```

```
[[   19 19000]
 [   35 20000]
 [   26 43000]]
---------------
[0 0 0]
```

In [33]:
```
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
print(X_train[:3])
print('-'*15)
print(y_train[:3])
print('-'*15)
print(X_test[:3])
print('-'*15)
print(y_test[:3])
```

```
[[    44  39000]
 [    32 120000]
 [    38  50000]]
---------------
[0 1 0]
---------------
[[   30 87000]
 [   38 50000]
 [   35 75000]]
---------------
[0 0 0]
```

In [34]:
```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

In [35]:
```
print(X_train[:3])
print('-'*15)
print(X_test[:3])
```

```
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]]
---------------
[[-0.80480212  0.50496393]
 [-0.01254409 -0.5677824 ]
 [-0.30964085  0.1570462 ]]
```

In [36]:
```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0, solver='lbfgs' )
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print(X_test[:10])
print('-'*15)
print(y_pred[:10])
```

```
[[-0.80480212  0.50496393]
 [-0.01254409 -0.5677824 ]
 [-0.30964085  0.1570462 ]
 [-0.80480212  0.27301877]
 [-0.30964085 -0.5677824 ]
 [-1.10189888 -1.43757673]
 [-0.70576986 -1.58254245]
 [-0.21060859  2.15757314]
 [-1.99318916 -0.04590581]
 [ 0.8787462  -0.77073441]]
---------------
[0 0 0 0 0 0 0 1 0 1]
```

In [37]:
```
print(y_pred[:20])
print(y_test[:20])
```

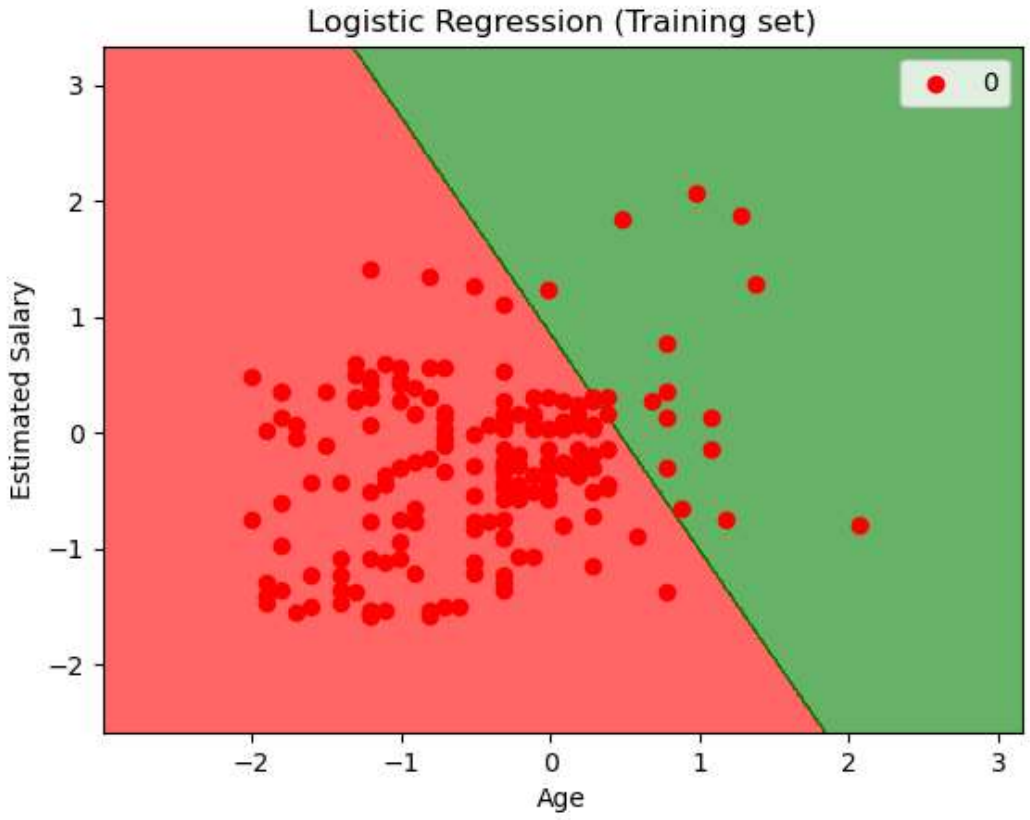
```
[0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0]
```

In [38]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

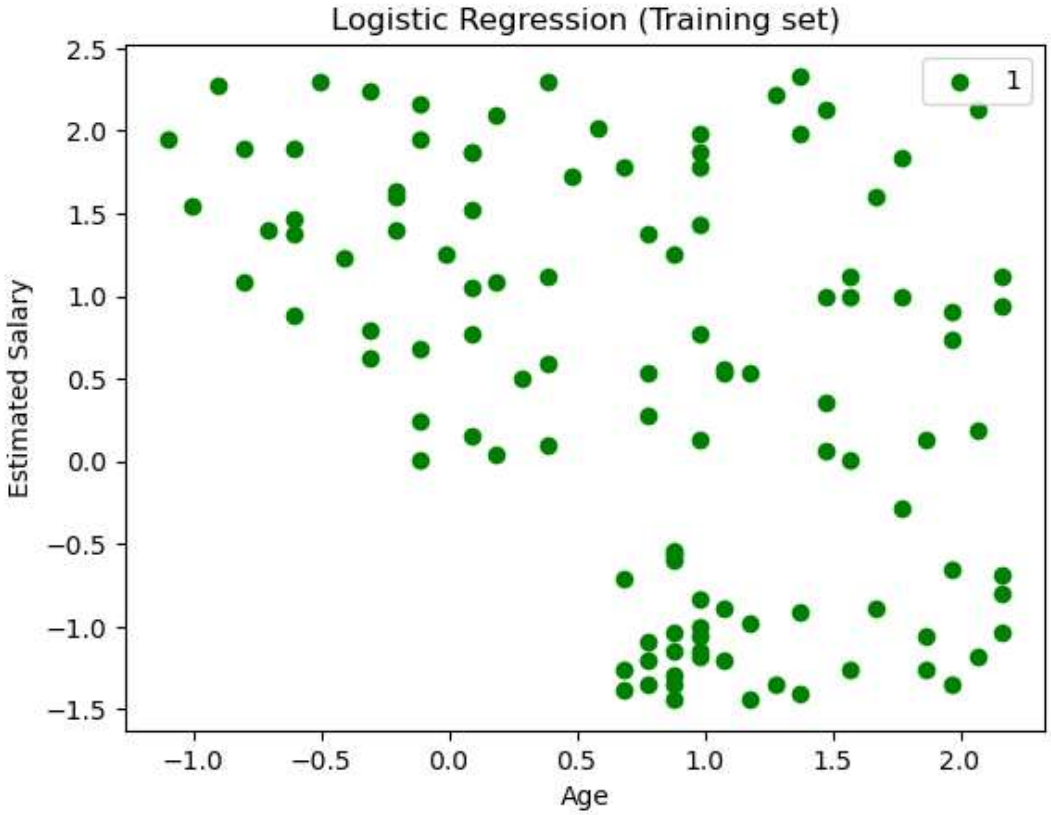
```
[[65  3]
 [ 8 24]]
```

In [40]:
```
# Visualizing the Training set results
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[:, 0].max() +
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() +
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1
             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* 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*.  Pleas
e use the *color* keyword-argument or provide a 2D array with a single row if you inten
d to specify the same RGB or RGBA value for all points.
```

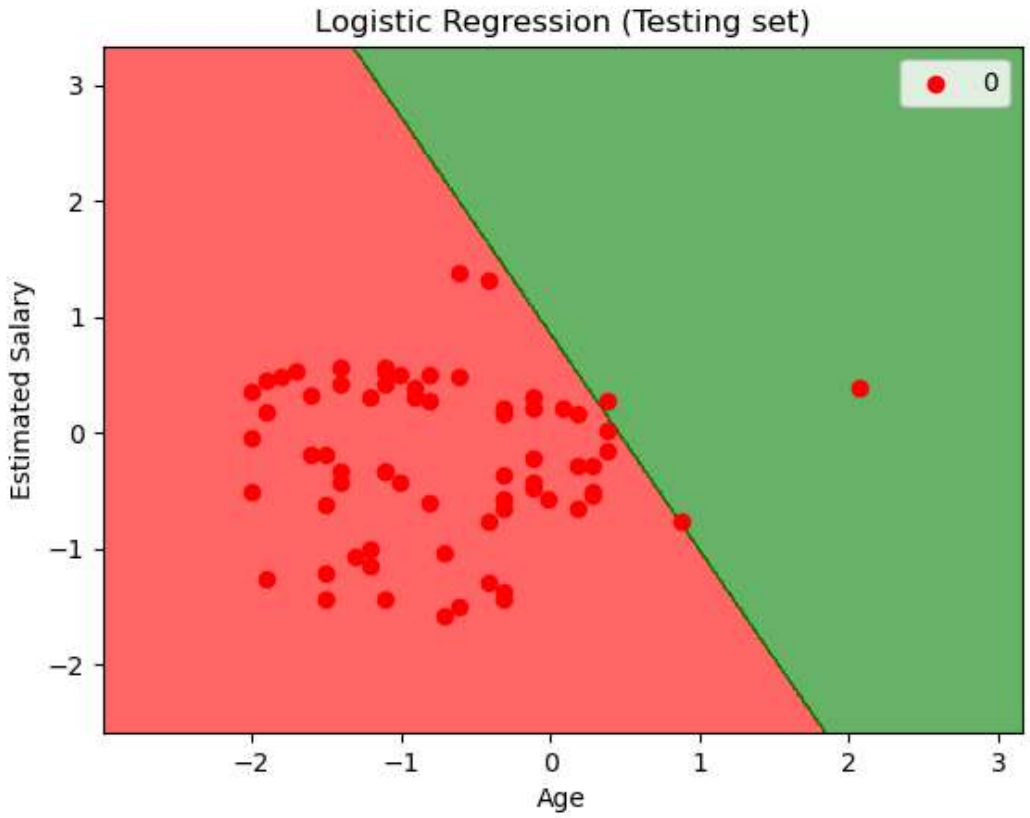



```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided
as value-mapping will have precedence in case its length matches with *x* & *y*.  Pleas
e use the *color* keyword-argument or provide a 2D array with a single row if you inten
d to specify the same RGB or RGBA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

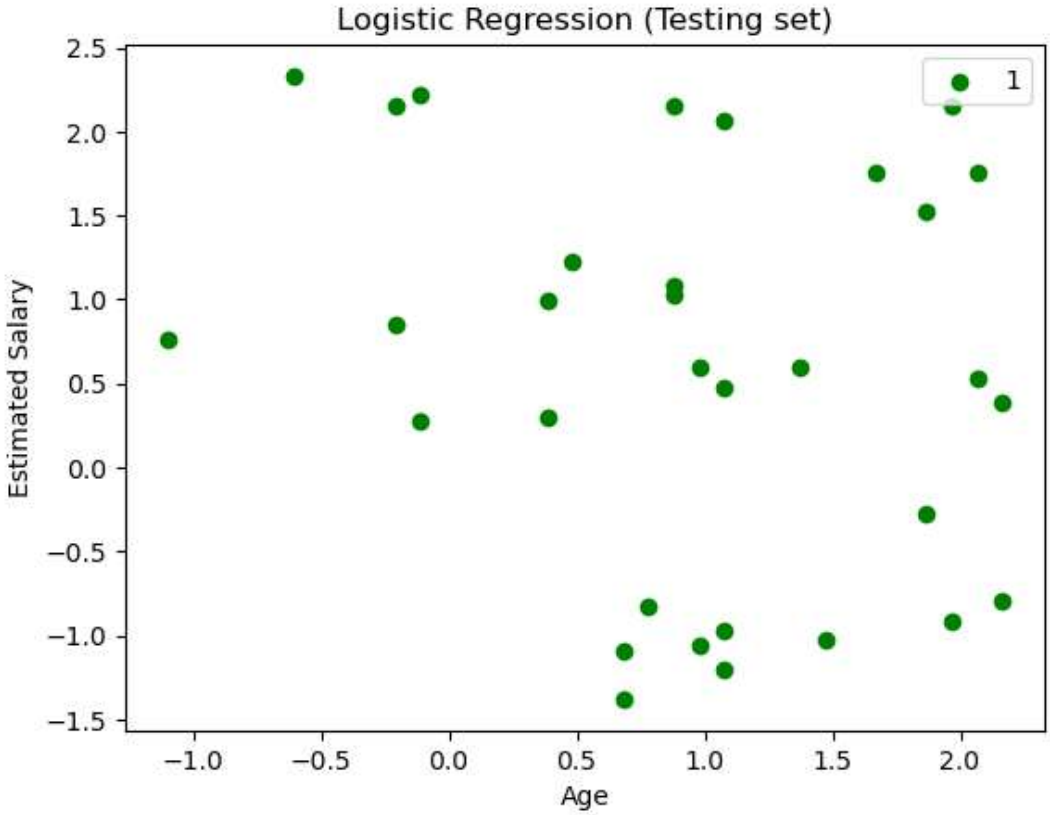

Logistic Regression (Training set)
Estimated Salary
Age
1
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

In [42]:
```
# Visualizing the Training set results
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[:, 0].max() +
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() +
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1
             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 (T 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*.  Pleas
e use the *color* keyword-argument or provide a 2D array with a single row if you inten
d to specify the same RGB or RGBA value for all points.
```




```
*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*.  Pleas
e use the *color* keyword-argument or provide a 2D array with a single row if you inten
d to specify the same RGB or RGBA value for all points.
```


Logistic Regression (Testing set)
Estimated Salary
Age
1
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0