https://towardsdatascience.com/machine-learning-basics-logistic-regression-890ef5e3a272 (https://towardsdatascience.com/machine-learning-basics-logistic-regression-890ef5e3a272)

Importing the Libraries

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

Importing the dataset

In [2]:
```
dataset = pd.read_csv('https://raw.githubusercontent.com/mk-gurucharan
X = dataset.iloc[:, [0, 1]].values
y = dataset.iloc[:, 2].values
```

In [3]:
```
dataset.head(5)
```

Out[3]:

	DMV_Test_1	DMV_Test_2	Results
0	34.623660	78.024693	0
1	30.286711	43.894998	0
2	35.847409	72.902198	0
3	60.182599	86.308552	1
4	79.032736	75.344376	1

Splitting the dataset into the Training set and Test set

In this the test_size=0.25 denotes that 25% of the data will be kept as the Test set and the remaining 75% will be used for training as the Training set.

In [4]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
```

Feature Scaling

This is an additional step that is used to normalize the data within a particular range. It also aids in speeding up the calculations. As the data is widely varying, we use this function to limit the range of the data within a small limit (-2,2). For example, the score 62.0730638 is normalized to -0.21231162 and the score 96.51142588 is normalized to 1.55187648. In this way, the scores of X_train and X_test are normalized to a smaller range.

In [6]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Training the Logistic Regression model on the Training Set

In this step, the class LogisticRegression is imported and is assigned to the variable “classifier”. The classifier.fit() function is fitted with X_train and Y_train on which the model will be trained.

In [7]:
```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X_train, y_train)
```

Out[7]:
```
LogisticRegression()
```

Predicting the Test set results

In this step, the classifier.predict() function is used to predict the values for the Test set and the values are stored to the variable y_pred.

In [8]:
```
y_pred = classifier.predict(X_test)
y_pred
```

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

Confusion Matrix and Accuracy

The confusion matrix is a table that is used to show the number of correct and incorrect predictions on a classification problem when the real values of the Test Set are known. It is of the format

The True values are the number of correct predictions made.

In [9]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
from sklearn.metrics import accuracy_score
print ("Accuracy : ", accuracy_score(y_test, y_pred))
cm
```

```
Accuracy :  0.88
```

Out[9]:
```
array([[11,  0],
       [ 3, 11]])
```

Comparing the Real Values with Predicted Values

In this step, a Pandas DataFrame is created to compare the classified values of both the original Test set (y_test) and the predicted results (y_pred).

In [10]:
```
df = pd.DataFrame({'Real Values':y_test, 'Predicted Values':y_pred})
df
```

Out[10]:

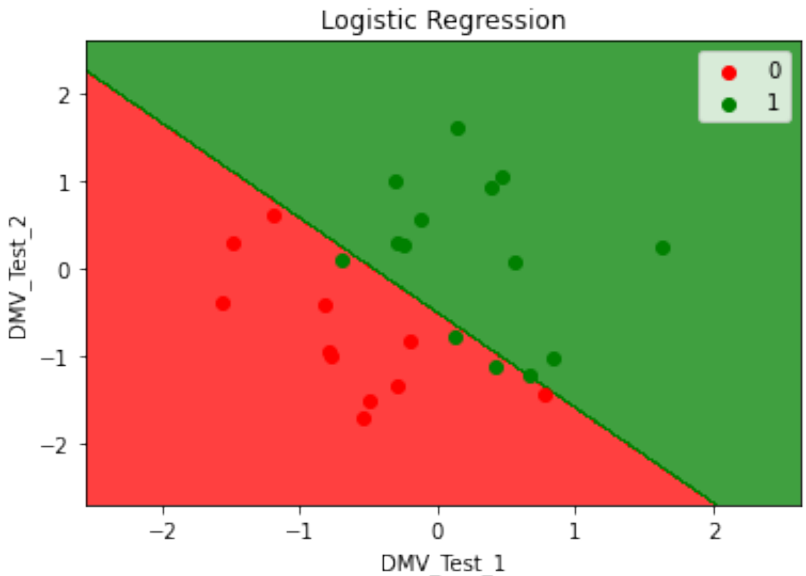
	Real Values	Predicted Values
0	1	1
1	0	0
2	0	0
3	0	0
4	1	1
5	1	1
6	1	0
7	1	1
8	0	0
9	1	1
10	0	0
11	0	0
12	0	0
13	1	1
14	1	0
15	1	1
16	0	0
17	1	1
18	1	0
19	1	1
20	0	0
21	0	0
22	1	1
23	1	1
24	0	0

Visualising the Results

In this last step, we visualize the results of the Logistic Regression model on a graph that is plotted along with the two regions

In [11]:
```
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
                     np.arange(start = X_set[:, 1].min() - 1, stop = X
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel
             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('Logistic Regression')
plt.xlabel('DMV_Test_1')
plt.ylabel('DMV_Test_2')
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-argum
ent or provide a 2-D array with a single row if you intend to specif
y 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*.  Please use the *color* keyword-argum
ent or provide a 2-D array with a single row if you intend to specif
y the same RGB or RGBA value for all points.
```



In [ ]: