In [1]:

```
# Import our relevant libraries
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
%matplotlib inline
```

In [2]:

```
data = pd.read_csv('Breast_cancer_wisconsin_diagnosis.csv')
data
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothnes
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	
...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	
565	926682	M	20.13	28.25	131.20	1261.0	
566	926954	M	16.60	28.08	108.30	858.1	
567	927241	M	20.60	29.33	140.10	1265.0	
568	92751	B	7.76	24.54	47.92	181.0	

569 rows × 33 columns

In [3]:

```
# Drop the id column
data = data.drop('id', axis=1)
data = data.drop('Unnamed: 32', axis=1)
# Convert the diagnosis column to numeric format
data['diagnosis'] = data['diagnosis'].factorize()[0]
# Fill all Null values with zero
data = data.fillna(value=0)
# Store the diagnosis column in a target object and then drop it
X = data.drop('diagnosis', axis=1)
y = data['diagnosis']
```

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 = 0.3, random
```

In [5]:

```
print('X_train.shape:', X_train.shape)
print('X_test.shape:', X_test.shape)
print('y_train.shape:', y_train.shape)
print('y_test.shape:',y_test.shape)
```

```
X_train.shape: (398, 30)
X_test.shape: (171, 30)
y_train.shape: (398,)
y_test.shape: (171,)
```

VISUALISING PCA AND TSNE PLOTS¶

Let's get to the meat of this notebook which is to produce high-level PCA and TSNE visuals

In [6]:

```
from sklearn.decomposition import PCA # Principal Component Analysis module
from sklearn.manifold import TSNE # TSNE module
```

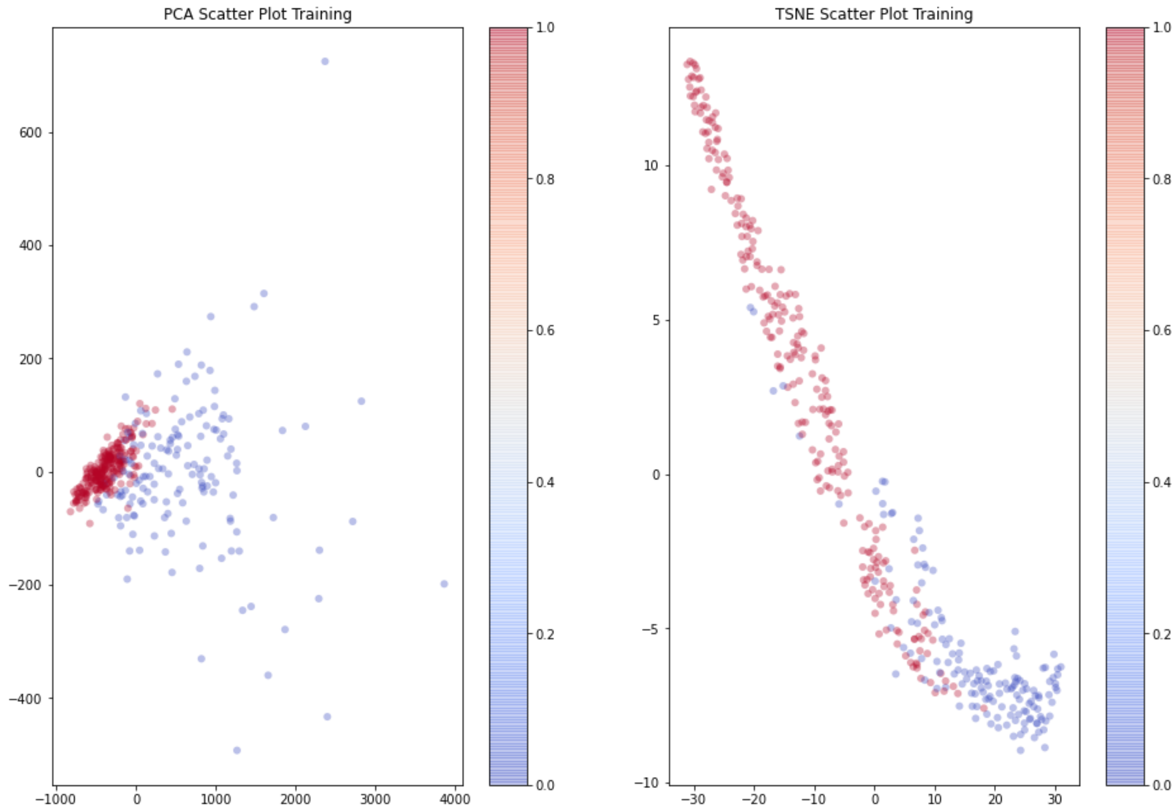
In [7]:

```
# Turn dataframe into arrays
X_train = X_train.values
X_test = X_test.values

# Invoke the PCA method. Since this is a binary classification problem
# let's call n_components = 2
pca = PCA(n_components=2)
train_pca_2d = pca.fit_transform(X_train)
test_pca_2d = pca.fit_transform(X_test)

# Invoke the TSNE method
tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=2000)
train_tsne_results = tsne.fit_transform(X_train)
test_tsne_results = tsne.fit_transform(X_test)
```

```
[t-SNE] Computing 121 nearest neighbors...
[t-SNE] Indexed 398 samples in 0.001s...
[t-SNE] Computed neighbors for 398 samples in 0.015s...
[t-SNE] Computed conditional probabilities for sample 398 / 398
[t-SNE] Mean sigma: 17.977994
[t-SNE] KL divergence after 250 iterations with early exaggeration: 5
0.973656
[t-SNE] KL divergence after 1800 iterations: 0.173023
[t-SNE] Computing 121 nearest neighbors...
[t-SNE] Indexed 171 samples in 0.000s...
[t-SNE] Computed neighbors for 171 samples in 0.007s...
[t-SNE] Computed conditional probabilities for sample 171 / 171
[t-SNE] Mean sigma: 11.093358
[t-SNE] KL divergence after 250 iterations with early exaggeration: 5
0.035530
[t-SNE] KL divergence after 900 iterations: 0.130900
```

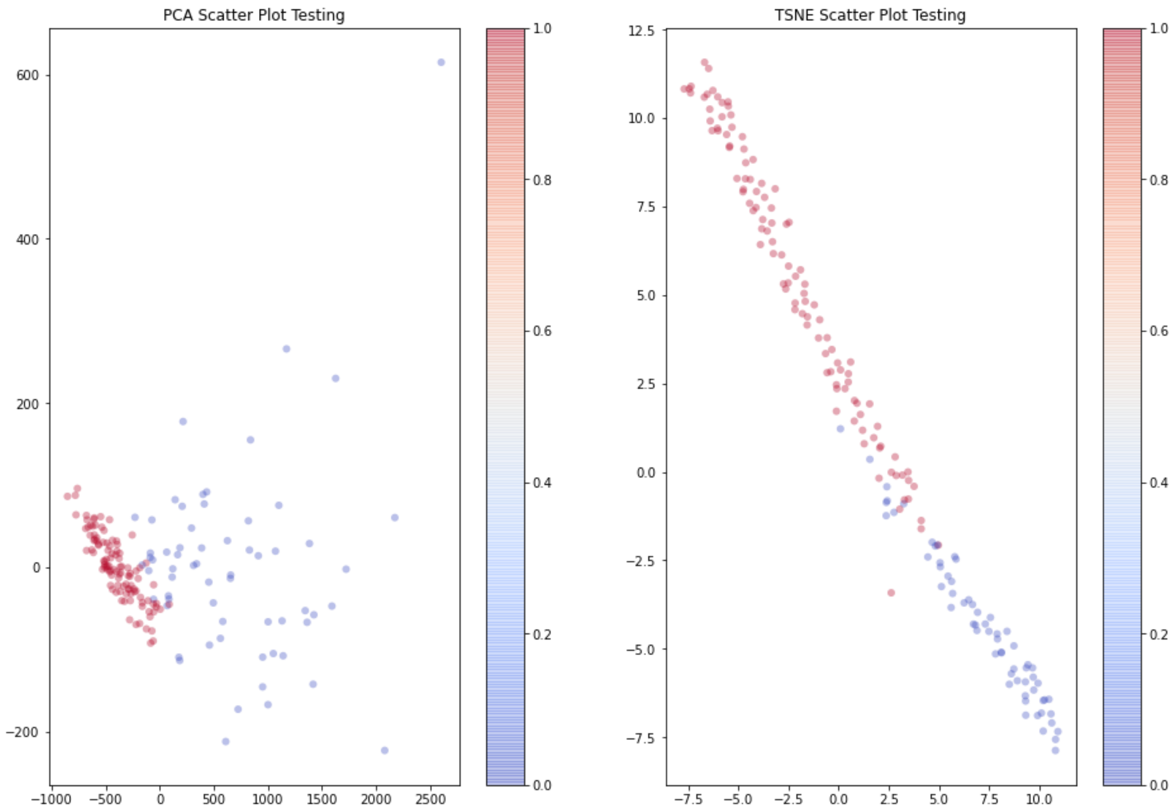
In [8]:

```
# Plot the TSNE and PCA visuals side-by-side
plt.figure(figsize = (16,11))
plt.subplot(121)
plt.scatter(train_pca_2d[:,0],train_pca_2d[:,1], c = y_train,
            cmap = "coolwarm", edgecolor = "None", alpha=0.35)
plt.colorbar()
plt.title('PCA Scatter Plot Training')
plt.subplot(122)
plt.scatter(train_tsne_results[:,0],train_tsne_results[:,1],  c = y_train,
            cmap = "coolwarm", edgecolor = "None", alpha=0.35)
plt.colorbar()
plt.title('TSNE Scatter Plot Training')
plt.show()
```



In [9]:

```python
# Plot the TSNE and PCA visuals side-by-side
plt.figure(figsize = (16,11))
plt.subplot(121)
plt.scatter(test_pca_2d[:,0],test_pca_2d[:,1], c = y_test,
            cmap = "coolwarm", edgecolor = "None", alpha=0.35)
plt.colorbar()
plt.title('PCA Scatter Plot Testing')
plt.subplot(122)
plt.scatter(test_tsne_results[:,0],test_tsne_results[:,1],  c = y_test,
            cmap = "coolwarm", edgecolor = "None", alpha=0.35)
plt.colorbar()
plt.title('TSNE Scatter Plot Testing')
plt.show()
```



As one can see from these high-level plots, even though PCA does quite a decent job of visualising our two target clusters (M for Malignant and B for Benign - cheating a bit here with the labels), the visuals in TSNE is much more obvious in terms of the demarcation in the target.

STANDARDISATION AND VISUALISATION

Let's now try scaling (or standardising) our features and see if we can get even more obvious/intuitive clusters in our plots.

In [10]:

```
# Calling Sklearn scaling method
from sklearn.preprocessing import StandardScaler
X_train_std = StandardScaler().fit_transform(X_train)
X_test_std = StandardScaler().fit_transform(X_test)
```

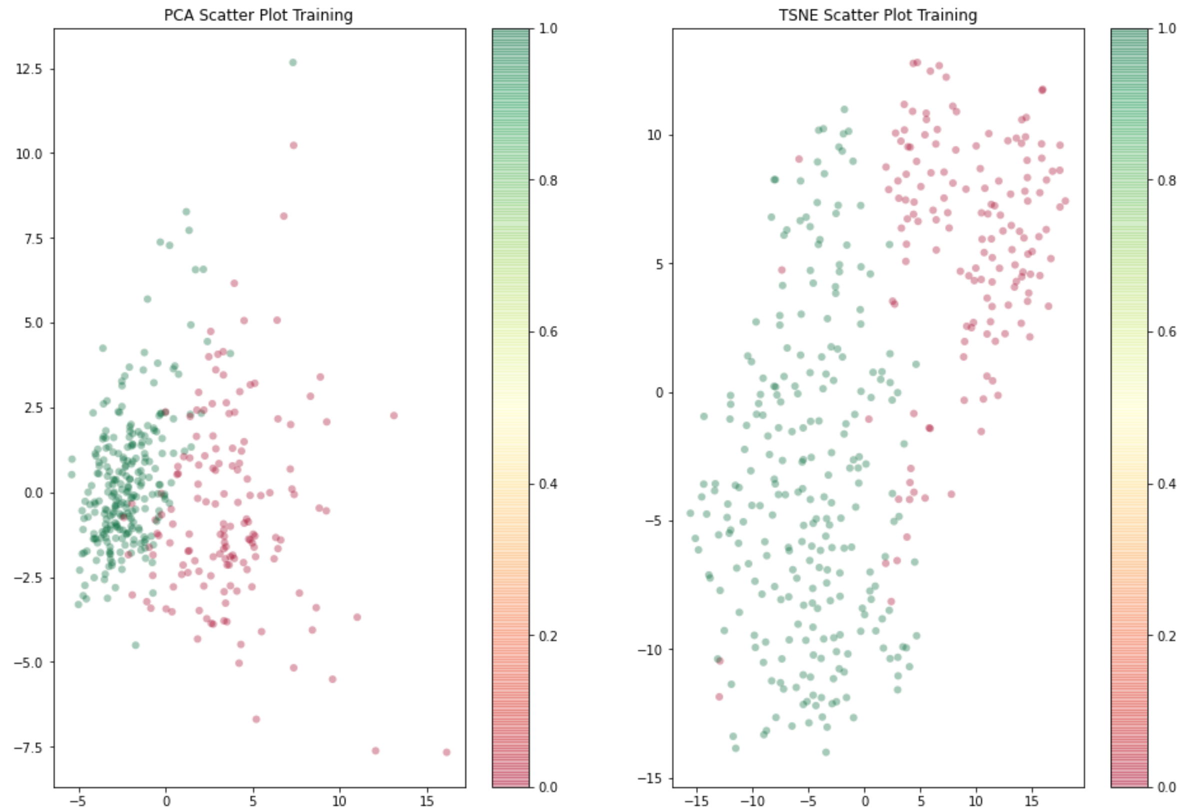
In [11]:

```
# Invoke the PCA method on the standardised data
pca = PCA(n_components=2)
train_pca_2d_std = pca.fit_transform(X_train_std)
test_pca_2d_std = pca.fit_transform(X_test_std)
# Invoke the TSNE method
tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=2000)
train_tsne_results_std = tsne.fit_transform(X_train_std)
test_tsne_results_std = tsne.fit_transform(X_test_std)
```

```
[t-SNE] Computing 121 nearest neighbors...
[t-SNE] Indexed 398 samples in 0.001s...
[t-SNE] Computed neighbors for 398 samples in 0.021s...
[t-SNE] Computed conditional probabilities for sample 398 / 398
[t-SNE] Mean sigma: 1.678350
[t-SNE] KL divergence after 250 iterations with early exaggeration: 6
3.064606
[t-SNE] KL divergence after 900 iterations: 0.727239
[t-SNE] Computing 121 nearest neighbors...
[t-SNE] Indexed 171 samples in 0.000s...
[t-SNE] Computed neighbors for 171 samples in 0.003s...
[t-SNE] Computed conditional probabilities for sample 171 / 171
[t-SNE] Mean sigma: 2.245331
[t-SNE] KL divergence after 250 iterations with early exaggeration: 5
7.437195
[t-SNE] KL divergence after 1000 iterations: 0.436906
```

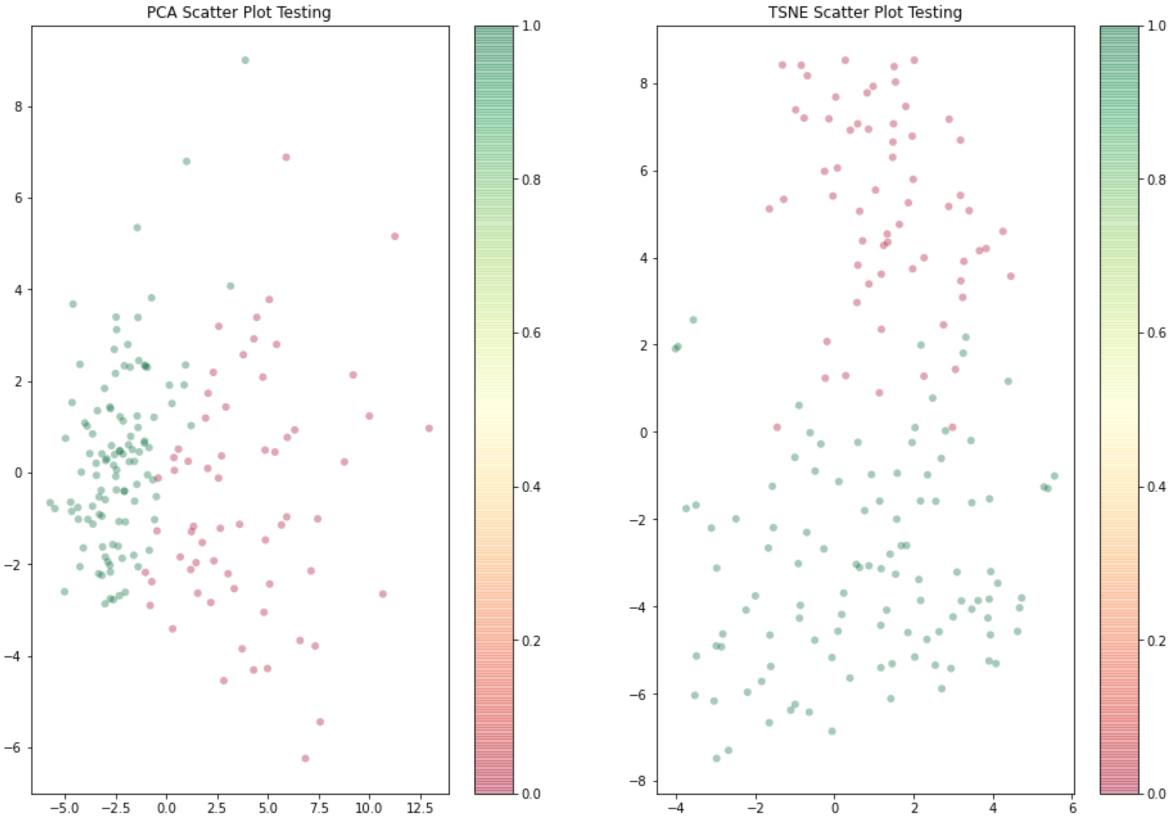
In [12]:

```
# Plot the TSNE and PCA visuals side-by-side
plt.figure(figsize = (16,11))
plt.subplot(121)
plt.scatter(train_pca_2d_std[:,0],train_pca_2d_std[:,1], c = y_train,
            cmap = "RdYlGn", edgecolor = "None", alpha=0.35)
plt.colorbar()
plt.title('PCA Scatter Plot Training')
plt.subplot(122)
plt.scatter(train_tsne_results_std[:,0],train_tsne_results_std[:,1],  c = y_tra
            cmap = "RdYlGn", edgecolor = "None", alpha=0.35)
plt.colorbar()
plt.title('TSNE Scatter Plot Training')
plt.show()
```



In [13]:

```
# Plot the TSNE and PCA visuals side-by-side
plt.figure(figsize = (16,11))
plt.subplot(121)
plt.scatter(test_pca_2d_std[:,0],test_pca_2d_std[:,1], c = y_test,
            cmap = "RdYlGn", edgecolor = "None", alpha=0.35)
plt.colorbar()
plt.title('PCA Scatter Plot Testing')
plt.subplot(122)
plt.scatter(test_tsne_results_std[:,0],test_tsne_results_std[:,1],  c = y_test,
            cmap = "RdYlGn", edgecolor = "None", alpha=0.35)
plt.colorbar()
plt.title('TSNE Scatter Plot Testing')
plt.show()
```



In [14]:

```
X_train = train_tsne_results_std
X_test = test_tsne_results_std
```

Normalisation

In [15]:

```
X_train = (X_train - np.min(X_train))/(np.max(X_train) - np.min(X_train))
X_test = (X_test - np.min(X_test))/(np.max(X_test) - np.min(X_test))
```

LogisticRegression

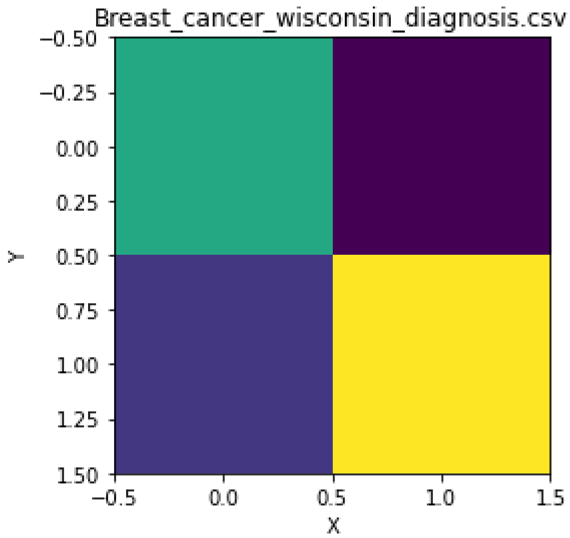
In [16]:

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(random_state=0)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

In [17]:

```
from sklearn.metrics import confusion_matrix


conf_matrix = confusion_matrix(y_test, y_pred)
print('\n\nThe Confusion Matrix for our TSNE Logistic Regression is:\n')
print(conf_matrix)

plt.imshow(conf_matrix)
plt.title('Breast_cancer_wisconsin_diagnosis.csv')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```

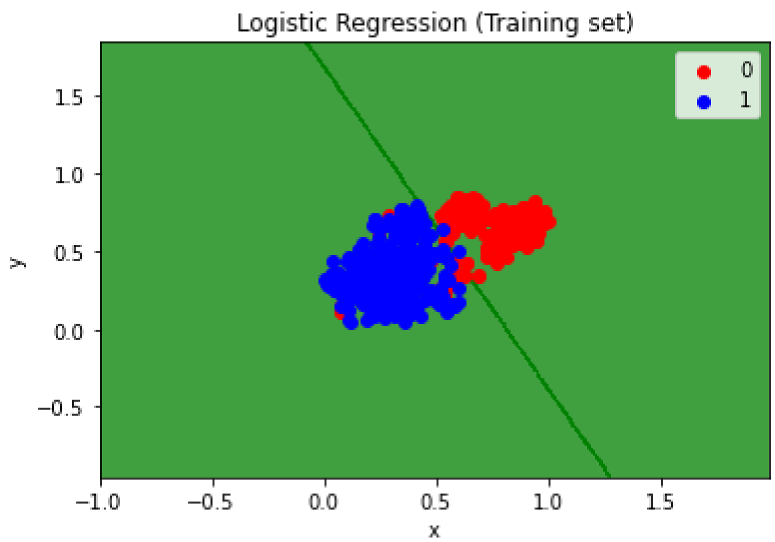
```


The Confusion Matrix for our TSNE Logistic Regression is:

[[56  7]
 [20 88]]
```



In [18]:

```
# Visualising 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
                     np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1
plt.contourf(X1, X2, model.predict(np.array([X1.ravel(), X2.ravel()]).T).reshap
             alpha = 0.75, cmap = ListedColormap(('green', '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', 'blue'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

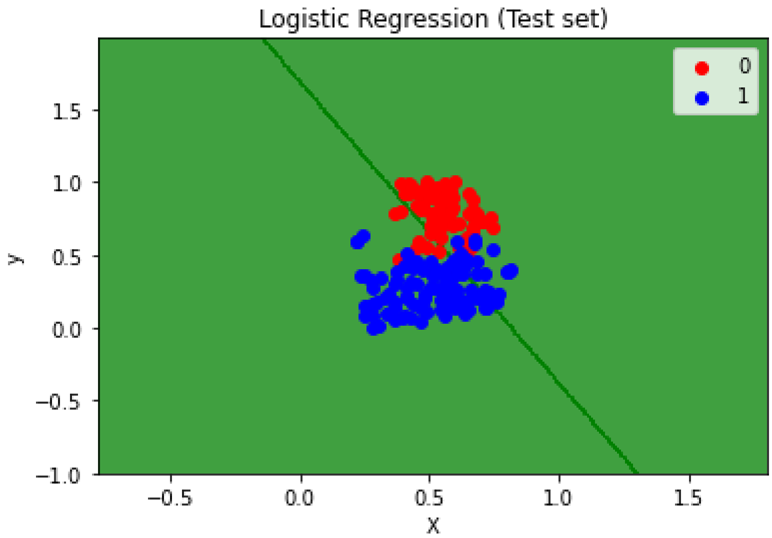
```
*c* argument looks like a single numeric RGB or RGBA sequence, which s
hould be avoided as value-mapping will have precedence in case its len
gth matches with *x* & *y*.  Please use the *color* keyword-argument o
r provide a 2-D array with a single row if you intend to specify the s
ame RGB or RGBA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



In [19]:

```
# Visualising the Test 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
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1
plt.contourf(X1, X2, model.predict(np.array([X1.ravel(), X2.ravel()]).T).reshap
             alpha = 0.75, cmap = ListedColormap(('green', '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', 'blue'))(i), label = j)
plt.title('Logistic Regression (Test set)')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()

from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test,y_pred)
print('\n\n\n Hence the accuracy of the t-sne for Logistic Regression is:',accu
print('\n\n Done :)')
```

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




```
 Hence the accuracy of the t-sne for Logistic Regression is: 0.8421052
631578947


 Done :)
```