Task 3

- Part B - Principal Component Analysis

In [1]:
```
# PCA

#Importing libraries
import  pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
# Importing and reading csv file
df=pd.read_csv("E:HIGGS_6M.csv")
```

In [3]:
```
# Anaylsing the distribution of classes
# data["column_name"].value_counts(), returns unique values in that column

print(df['1.0000000000000000e+00'].value_counts())
print('Zeros', round(df['1.0000000000000000e+00'].value_counts()[0]/len(df) * 1
print('Ones', round(df['1.0000000000000000e+00'].value_counts()[1]/len(df) * 10
```

```
1.0    3178344
0.0    2821655
Name: 1.0000000000000000e+00, dtype: int64
Zeros 47.03 % of the dataset
Ones 52.97 % of the dataset
```

In [4]:
```
Ones_df = df.loc[df['1.0000000000000000e+00'] == 1][0:2500] # smaples which hav
Zeros_df = df.loc[df['1.0000000000000000e+00'] == 0][0:2500] #

normal_distributed_df = pd.concat([Ones_df, Zeros_df])

# Shuffle dataframe rows
df_new= normal_distributed_df.sample(frac=1, random_state=100)

print(df_new['1.0000000000000000e+00'].value_counts()/len(df))
```

```
0.0    0.000417
1.0    0.000417
Name: 1.0000000000000000e+00, dtype: float64
```

In [5]:
```
# separating dependent and independent feature

X = df_new.drop("1.0000000000000000e+00",axis=1)
y = df_new["1.0000000000000000e+00"]
```

In [6]:
```
#Splitting the dataset into the training and test sets
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 [7]:
```
#Putting the values in the same scale
from sklearn.preprocessing import StandardScaler
scalar = StandardScaler()
X_train = scalar.fit_transform(X_train)
X_test = scalar.transform(X_test)

#Adding x0 = 1 to each instance for the bias term

X_train = np.concatenate((np.ones((X_train.shape[0],1)),X_train),axis=1)
X_test = np.concatenate((np.ones((X_test.shape[0],1)),X_test),axis=1)
```

In [8]:
```
# Applying PCA
from sklearn.decomposition import PCA
# put none to n_componenets to create explained variance vector
# ( contain the percentage of variance explained by each of the principal compone
pca = PCA(n_components=2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
expained_variance = pca.explained_variance_ratio_
print(expained_variance)
```

```
[0.14717072 0.06633983]
```

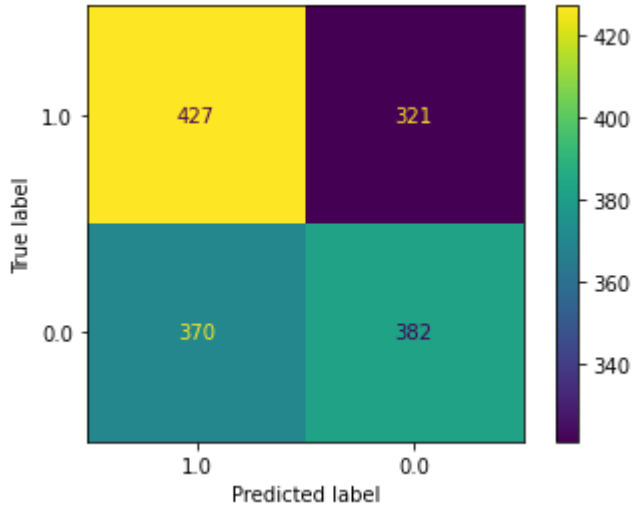
In [9]:
```
#Fitting logistic Regression to the training set
from sklearn.linear_model import LogisticRegression
logmodel = LogisticRegression(random_state = 0)
logmodel.fit(X_train, y_train)
```

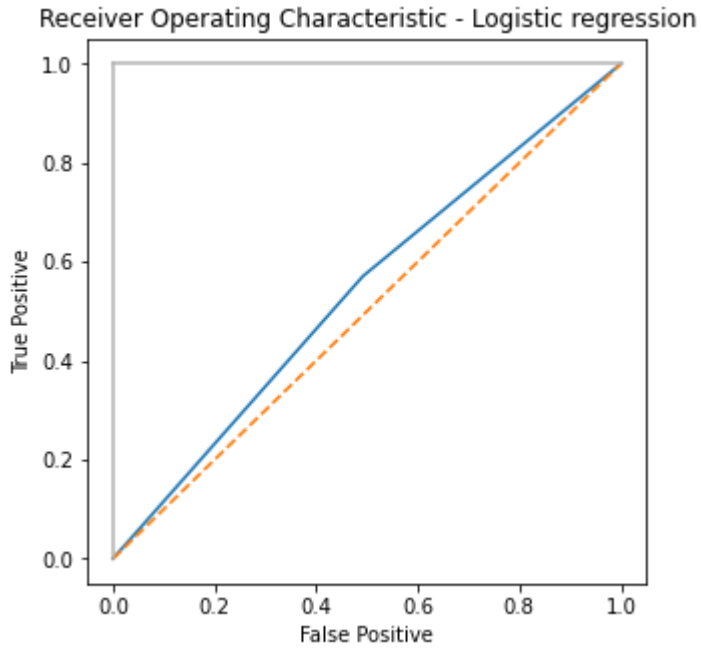
Out[9]:
```
LogisticRegression(random_state=0)
```

In [10]:
```
#Prdicting the test set results
y_pred = logmodel.predict(X_test)
```

In [11]:
```
from sklearn.metrics import roc_curve
false_positive, true_positive, threshold1 = roc_curve(y_test, y_pred)
```

In [12]:
```
# Plotting
# Confusion Matrix

from sklearn.metrics import accuracy_score, classification_report, confusion_matr

class_labels = df_new['1.0000000000000000e+00'].unique()
cm = confusion_matrix(y_test, y_pred, labels=class_labels)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_labels)
disp.plot()

# ROC Plot
plt.subplots(1, figsize=(5,5))
plt.title('Receiver Operating Characteristic - Logistic regression')
plt.plot(false_positive, true_positive)
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive')
plt.xlabel('False Positive')
plt.show()

# Calculating model accuarcy
accuracy=accuracy_score(y_test,y_pred)
print("Accuracy Obtained= ",accuracy)
```

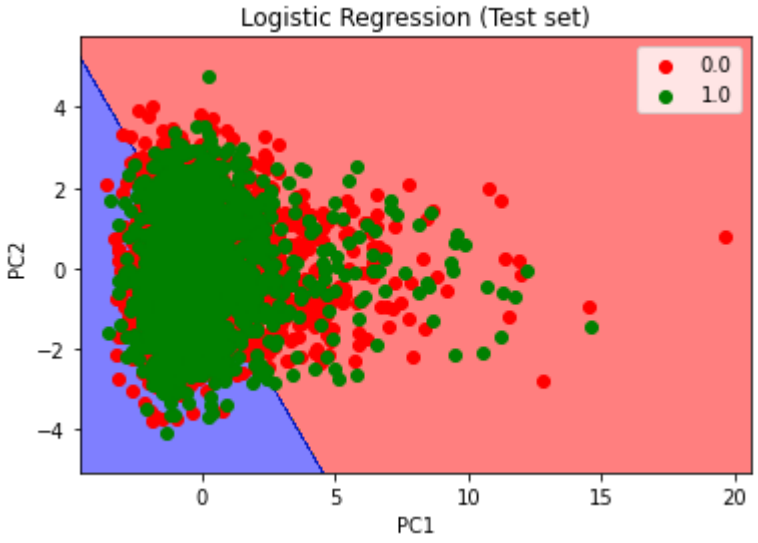





```
Accuracy Obtained=  0.5393333333333333
```

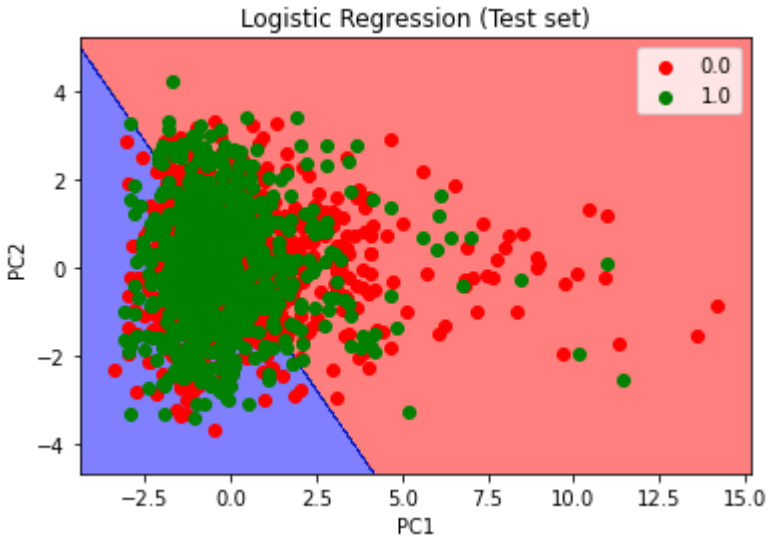
In [13]:
```
#Visualizing the train set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:,0].min() - 1, stop = X_set[:, 0].m
                     np.arange(start = X_set[:,1].min() - 1, stop = X_set[:, 1].m
plt.contourf(X1, X2, logmodel.predict(np.array([X1.ravel(), X2.ravel()]).T).resha
                                      alpha = 0.5, cmap = ListedColormap(('red'
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', 'blue'))(i), label = j)
plt.title('Logistic Regression (Test set)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.show()
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 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all point
s.
*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 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all point
s.
```



In [14]:
```
#Visualizing 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].m
                     np.arange(start = X_set[:,1].min() - 1, stop = X_set[:, 1].m
plt.contourf(X1, X2, logmodel.predict(np.array([X1.ravel(), X2.ravel()]).T).resha
                                      alpha = 0.5, cmap = ListedColormap(('red'
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', 'blue'))(i), label = j)
plt.title('Logistic Regression (Test set)')
plt.xlabel('PC1')
plt.ylabel('PC2')
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 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all point
s.
*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 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all point
s.
```



In [15]:
```
df_test=df[5499999:]

# separating dependent and independent feature

X_final_test= df_test.drop("1.0000000000000000e+00",axis=1)
y_final_test = df_test["1.0000000000000000e+00"]
```

In [16]:
```
#Feature Normalization
# standardization-or-mean-removal-and-variance-scaling
X_final_test = scalar.transform(X_final_test)

#Adding x0 = 1 to each instance for the bias term
X_final_test = np.concatenate((np.ones((X_final_test.shape[0],1)),X_final_test),a
```

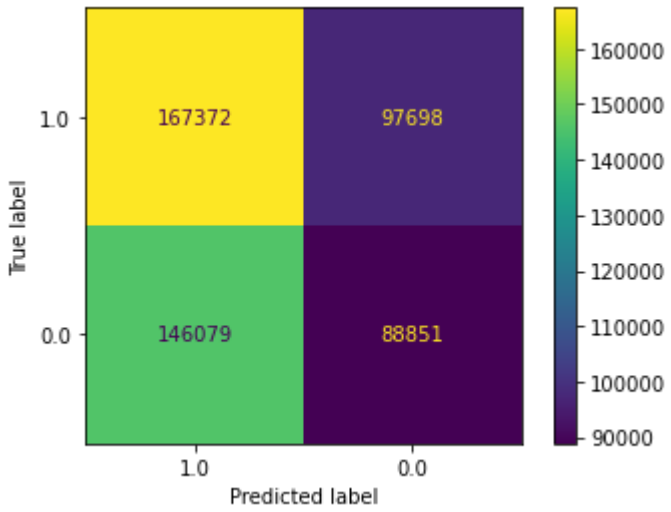
In [17]:
```
# Applying PCA
from sklearn.decomposition import PCA
# put none to n_componenets to create explained variance vector
# ( contain the percentage of variance explained by each of the principal compone
pca = PCA(n_components=2)
X_final_test = pca.fit_transform(X_final_test)
expained_variance = pca.explained_variance_ratio_
print(expained_variance)
```

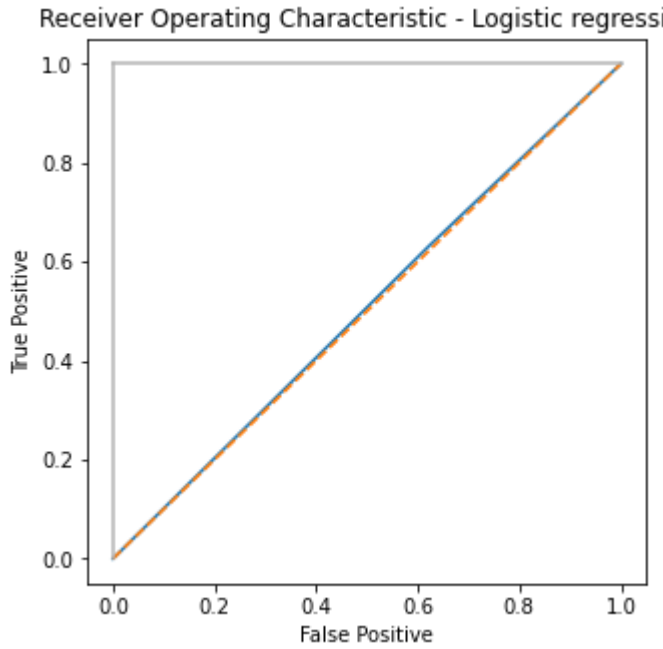
```
[0.15009816 0.06642647]
```

In [18]:
```
#Prdicting the test set results
y_final_pred = logmodel.predict(X_final_test)
```

In [19]:
```
false_positive_test, true_positive_test, threshold1 = roc_curve(y_final_test, y_f
```

In [20]:
```
# Plotting
# Confusion Matrix

class_labels = df_test['1.0000000000000000e+00'].unique()
cm_new = confusion_matrix(y_final_test,y_final_pred, labels=class_labels)
disp = ConfusionMatrixDisplay(confusion_matrix=cm_new, display_labels=class_label
disp.plot()

# ROC Plot
plt.subplots(1, figsize=(5,5))
plt.title('Receiver Operating Characteristic - Logistic regression')
plt.plot(false_positive_test, true_positive_test)
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive')
plt.xlabel('False Positive')
plt.show()

# Calculating model accuarcy
accuracy=accuracy_score(y_final_test,y_final_pred)
print("Accuracy Obtained= ",accuracy)
```






```
Accuracy Obtained=  0.512446
```

In [21]:
```
#Visualizing the fina_test set results
from matplotlib.colors import ListedColormap
X_final_set, y_final_set = X_final_test, y_final_test
X1_new, X2_new = np.meshgrid(np.arange(start = X_final_set[:,0].min() - 1, stop =
                     np.arange(start = X_final_set[:,1].min() - 1, stop = X_final
plt.contourf(X1_new, X2_new, logmodel.predict(np.array([X1_new.ravel(), X2_new.ra
                                          alpha = 0.5, cmap = ListedColormap(('red'
plt.xlim(X1_new.min(), X1_new.max())
plt.ylim(X2_new.min(), X2_new.max())
for i, j in enumerate(np.unique(y_final_set)):
    plt.scatter(X_final_set[y_final_set == j, 0], X_final_set[y_final_set == j, 1
                c= ListedColormap(('red', 'green', 'blue'))(i), label = j)
plt.title('Logistic Regression (Final Test set)')
plt.xlabel('PC1')
plt.ylabel('PC2')
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 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all point
s.
*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 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all point
s.
```