Assignment 1

Name: Gautam Kumar

Roll Number: 21CS30020

In [1]:

```
# import all the necessary libraries here
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from numpy.linalg import inv
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
```

In [2]:

```
df = pd.read_excel('../../dataset/logistic-regression/Pumpkin_Seeds_Dataset.xlsx')
print(df.shape)
```

```
(2500, 13)
```

In [3]:

```
df.head()
```

Out[3]:

	Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	Convex_Area	Equiv_Diameter
0	56276	888.242	326.1485	220.2388	56831	267.6805
1	76631	1068.146	417.1932	234.2289	77280	312.3614
2	71623	1082.987	435.8328	211.0457	72663	301.9822
3	66458	992.051	381.5638	222.5322	67118	290.8899
4	66107	998.146	383.8883	220.4545	67117	290.1207

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2500 entries, 0 to 2499
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Area               2500 non-null   int64  
 1   Perimeter          2500 non-null   float64
 2   Major_Axis_Length  2500 non-null   float64
 3   Minor_Axis_Length  2500 non-null   float64
 4   Convex_Area        2500 non-null   int64  
 5   Equiv_Diameter     2500 non-null   float64
 6   Eccentricity       2500 non-null   float64
 7   Solidity           2500 non-null   float64
 8   Extent             2500 non-null   float64
 9   Roundness          2500 non-null   float64
 10  Aspect_Ration      2500 non-null   float64
 11  Compactness        2500 non-null   float64
 12  Class              2500 non-null   object 
dtypes: float64(10), int64(2), object(1)
memory usage: 254.0+ KB
```

In [5]:

```
df.describe()
```

Out[5]:

	Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	Convex_Area	I
count	2500.000000	2500.000000	2500.000000	2500.000000	2500.000000	
mean	80658.220800	1130.279015	456.601840	225.794921	81508.084400	
std	13664.510228	109.256418	56.235704	23.297245	13764.092788	
min	47939.000000	868.485000	320.844600	152.171800	48366.000000	
25%	70765.000000	1048.829750	414.957850	211.245925	71512.000000	
50%	79076.000000	1123.672000	449.496600	224.703100	79872.000000	
75%	89757.500000	1203.340500	492.737650	240.672875	90797.750000	
max	136574.000000	1559.450000	661.911300	305.818000	138384.000000	

In [6]:

```
print(df["Class"].unique())
```

```
['Çerçevelik' 'Ürgüp Sivrisi']
```

In [7]:

```
mapping = {'Çerçevelik': 0 , 'Ürgüp Sivrisi' : 1}
df.replace({'Class': mapping} , inplace=True)
df.head()
```

Out[7]:

	Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	Convex_Area	Equiv_Diameter
0	56276	888.242	326.1485	220.2388	56831	267.6805
1	76631	1068.146	417.1932	234.2289	77280	312.3614
2	71623	1082.987	435.8328	211.0457	72663	301.9822
3	66458	992.051	381.5638	222.5322	67118	290.8899
4	66107	998.146	383.8883	220.4545	67117	290.1207

In [8]:

```
X = df.iloc[:,:-1].values
Y = df.iloc[:,-1].values
X_train , X,Y_train,Y = train_test_split(X,Y,test_size=0.5,random_state=0)
X_val,X_test,Y_val,Y_test = train_test_split(X,Y,test_size = 0.4,random_state = 0)
```

In [9]:

```
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
X_train= st_x.fit_transform(X_train)
X_test= st_x.transform(X_test)
```

In [10]:

```
print(X_train.shape, Y_train.shape)
print(X_val.shape, Y_val.shape)
print(X_test.shape, Y_test.shape)
```

```
(1250, 12) (1250,)
(750, 12) (750,)
(500, 12) (500,)
```

In [11]:

```
class logistic_regression():
    def __init__(self, epoch= 15000, learning_rate = 0.001 ):
        self.epoch = epoch
        self.learning_rate = learning_rate
        self.cost = []
        self.init_weight = None
        self.final_weight = None

    def initialize_weight(self,n_feature):
        limit = np.sqrt(1/n_feature)
        weight = np.random.uniform(-limit,limit,(n_feature,1))
        b = 0
        self.init_weight = np.insert(weight,0,b,axis = 0)


    def train(self, X,Y,X_val,Y_val):
        n_sample ,n_feature = X.shape
        X = np.insert(X,0,1,axis = 1)
        Y = np.reshape(Y,(n_sample,1))
        nv_sample = X_val.shape[0];
        X_val = np.insert(X_val,0,1,axis = 1);
        Y_val = np.reshape(Y_val,(nv_sample,1));
        self.initialize_weight(n_feature)
        self.fit(X,Y,X_val,Y_val)


    def fit(self,X,Y,X_val,Y_val):
        _weight = self.init_weight.copy()
        y_pred = self.sigmoid(np.dot(X,_weight))
        self.cost.append(self.gradient_cost(X,Y,_weight))

        for iter in range(self.epoch):
            y_pred = self.sigmoid(np.dot(X,_weight))
            grad = np.dot(X.T, y_pred - Y)
            _weight = _weight - self.learning_rate*grad
            self.cost.append(self.gradient_cost(X,Y,_weight))
            if iter%100 ==0:
                print(f"The training cost for iteration ::{iter} is ")
        self.final_weight = _weight
        return


    def predict(self,X):
        out = np.dot(X,self.final_weight)
        out = self.sigmoid(out)
        out = (out >= 0.5)*1
        return out

    def sigmoid(self,Y):
        sig = 1 + np.exp(-1*Y)
        sig = 1/sig
        return sig

    def gradient_cost(self,X,Y,_weight):
        y_pred = self.sigmoid(np.dot(X,_weight))
        return np.mean(-1*(Y*np.log(y_pred) + (1-Y)*np.log(1 - y_pred)))

    def viswalize_loss(self):
        figure, ax = plt.subplots()
```

```
        nums = np.arange(len(self.cost))
        ax.plot(nums, np.array(self.cost).reshape((len(self.cost,))))
        ax.set_xlabel('Epoch')
        ax.set_ylabel('Cost')
        plt.show()

    def metrics_loss(self,X,Y):
        n_sample,n_feature = X.shape
        X = np.insert(X,0,1,axis = 1)
        Y = np.reshape(Y,(n_sample,1))
        y_pred = self.sigmoid(np.dot(X,self.final_weight))
        y_pred = (y_pred >= 0.5)
        con_matrix = confusion_matrix(Y,y_pred)
        cm_display = ConfusionMatrixDisplay(confusion_matrix = con_matrix, display_label
        cm_display.plot()
        plt.show()
        recall = con_matrix[1][1]/(con_matrix[1][0] + con_matrix[1][1])
        precison = con_matrix[1][1] /(con_matrix[1][1] + con_matrix[0][1])
        accuracy = (con_matrix[0][0] + con_matrix[1][1])/(con_matrix[0][0] + con_matrix[
        df = pd.DataFrame([[recall, precison, accuracy]], columns=['Recall','Precision',
        return df

    def print_loss(self):
        print(self.cost)
```

In [12]:

```
logistic_regressor = logistic_regression()
```

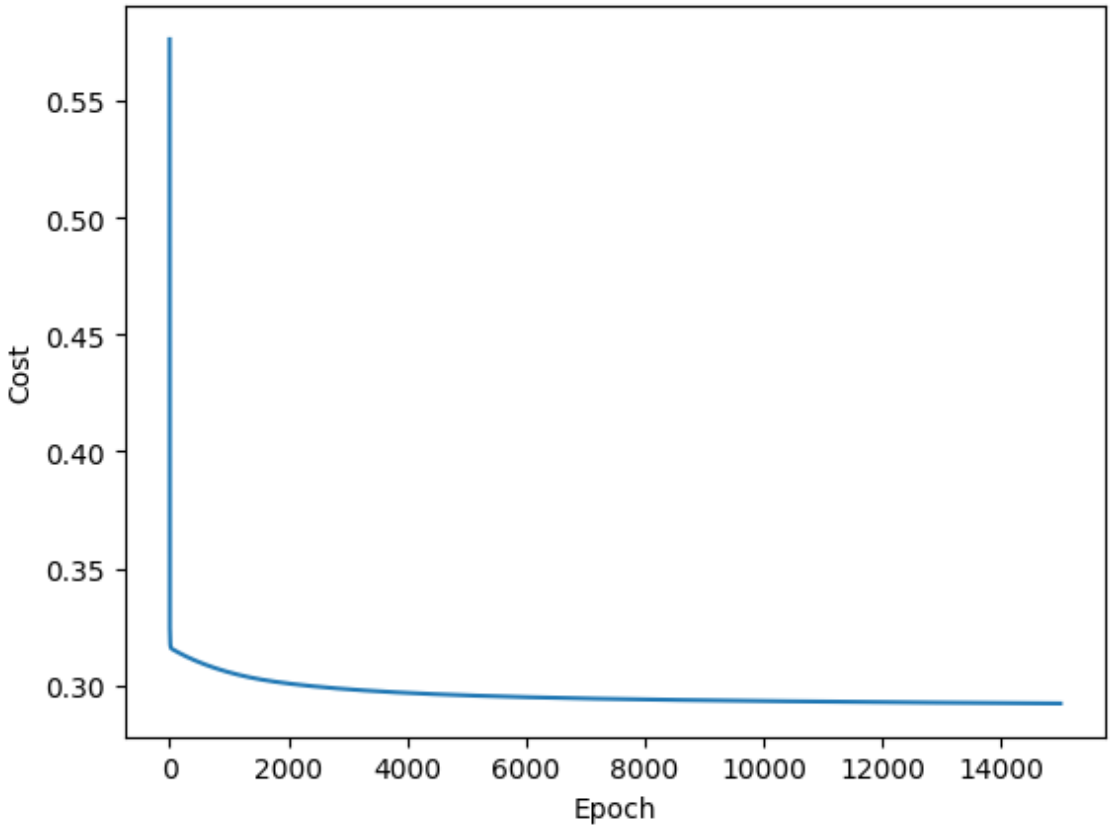
In [13]:

```
logistic_regressor.train(X_train,Y_train,X_val,Y_val)
```

```
The training cost for iteration ::0 is ____________________________________
__________________0.3461075834614365
......
The training cost for iteration ::100 is ____________________________________
____________________0.3145598695236912
......
The training cost for iteration ::200 is ____________________________________
____________________0.31320452265407783
......
The training cost for iteration ::300 is ____________________________________
____________________0.31195386603362807
......
The training cost for iteration ::400 is ____________________________________
____________________0.3107977525253971
......
The training cost for iteration ::500 is ____________________________________
____________________0.30972757178894045
......
The training cost for iteration ::600 is
```

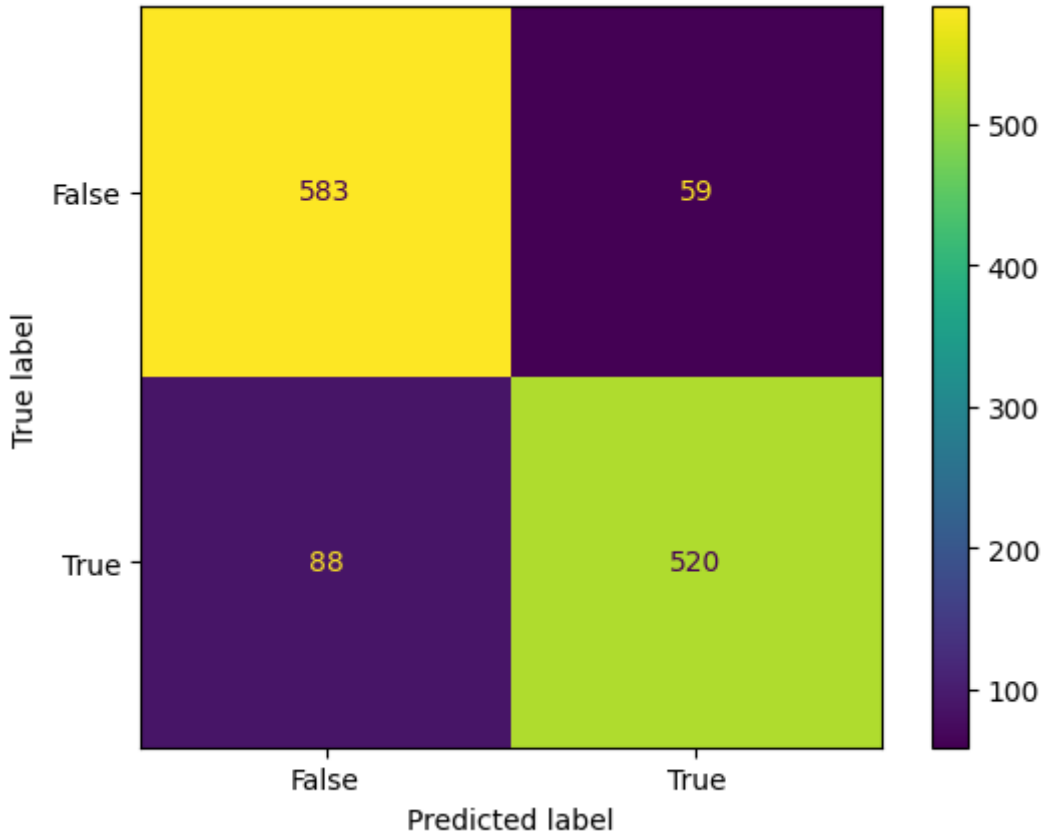
In [14]:

```
logistic_regressor.viswalize_loss()
```



In [15]:

```
mat_loss_df = logistic_regressor.metrics_loss(X_train,Y_train)
mat_loss_df.rename(index={0:'Train_data'},inplace=True)
```



In [16]:

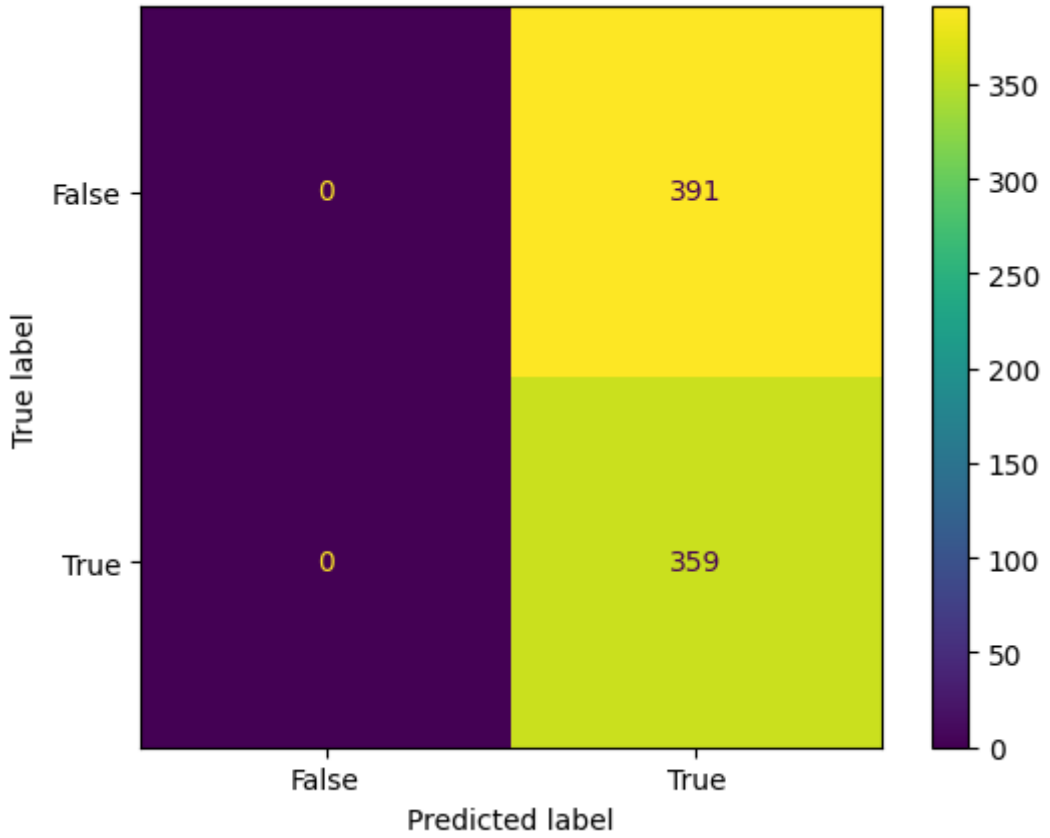
```
mat_loss_df
```

Out[16]:

	Recall	Precision	Mean Accuracy
Train_data	0.855263	0.8981	0.8824

In [17]:

```
mat_loss_df = logistic_regressor.metrics_loss(X_val,Y_val)
mat_loss_df.rename(index={0:'Validation_data'},inplace=True)
```



In [18]:

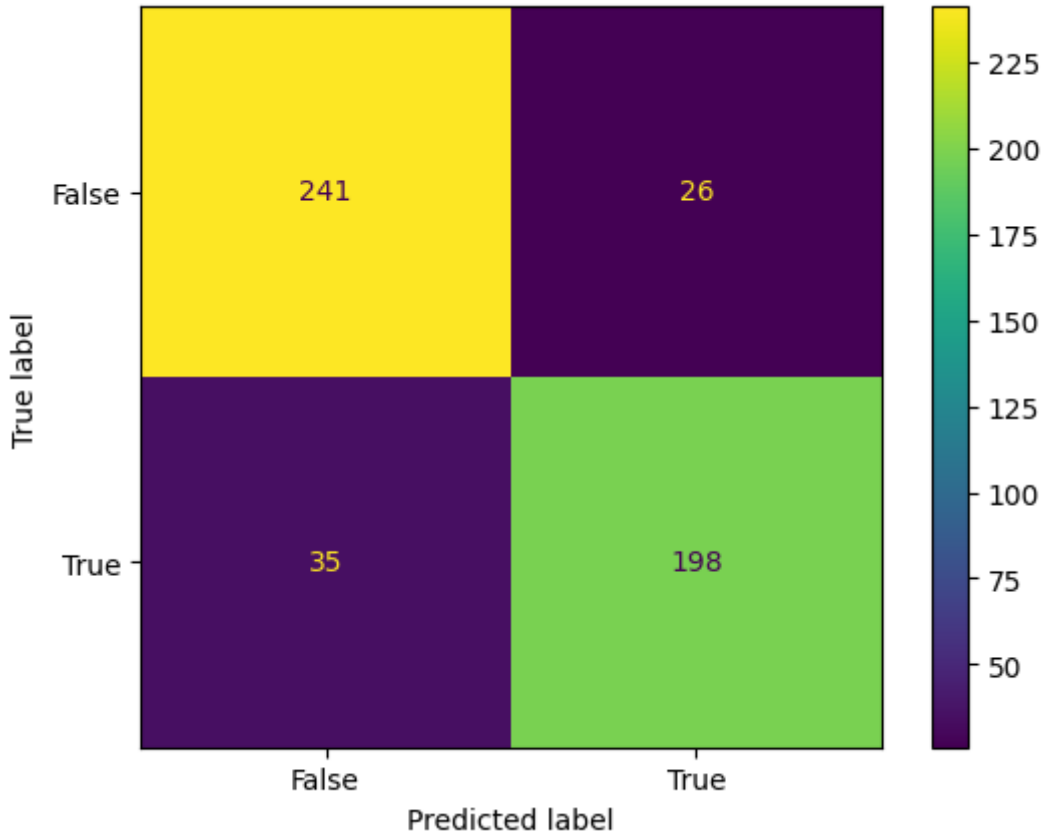
```
mat_loss_df
```

Out[18]:

	Recall	Precision	Mean Accuracy
Validation_data	1.0	0.478667	0.478667

In [19]:

```
mat_loss_df = logistic_regressor.metrics_loss(X_test,Y_test)
mat_loss_df.rename(index={0:'Test_data'},inplace=True)
```



In [20]:

```
mat_loss_df
```

Out[20]:

	Recall	Precision	Mean Accuracy
Test_data	0.849785	0.883929	0.878

In [ ]:

```

```