Importing Libraries

In [36]:
```
import pandas as pd #To read and edit tabular data
import numpy as np #To perform array/matrix operations
import math #For mathematical opeartions
import matplotlib.pyplot as plt #To perform plotting
%matplotlib inline
import seaborn as sns  #To perform visualization
sns.set_style('whitegrid')
import warnings #To ignore warnings
warnings.filterwarnings('ignore')
from IPython.display import Image #To display images in the notebook
```

Reading the data

In [37]:
```
df = pd.read_csv('Classification_Data.csv')
```

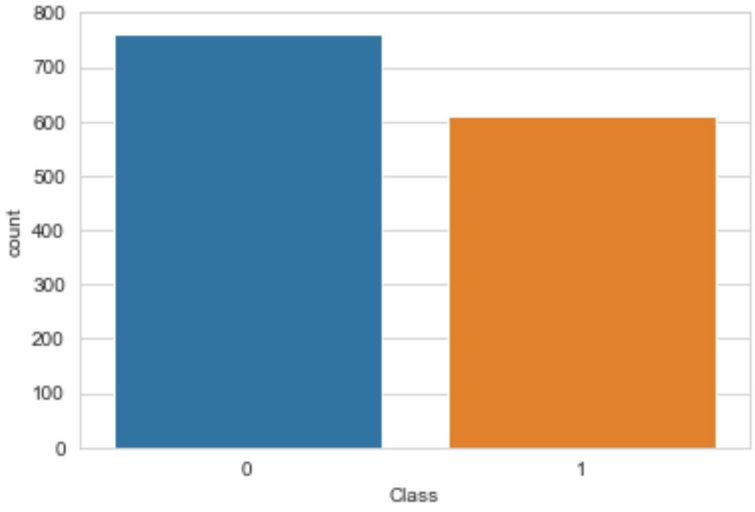
In [38]:
```
df.head()
```

Out[38]:

	Input1	Input2	Class
0	3.62160	8.6661	0
1	4.54590	8.1674	0
2	3.86600	-2.6383	0
3	3.45660	9.5228	0
4	0.32924	-4.4552	0

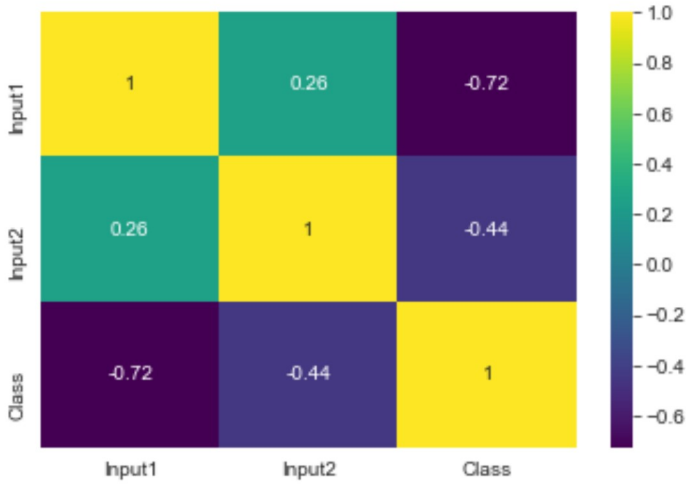
In [39]:
```
sns.countplot(x='Class', data=df)
```

Out[39]:
```
<AxesSubplot:xlabel='Class', ylabel='count'>
```



Dataset is almost well balanced

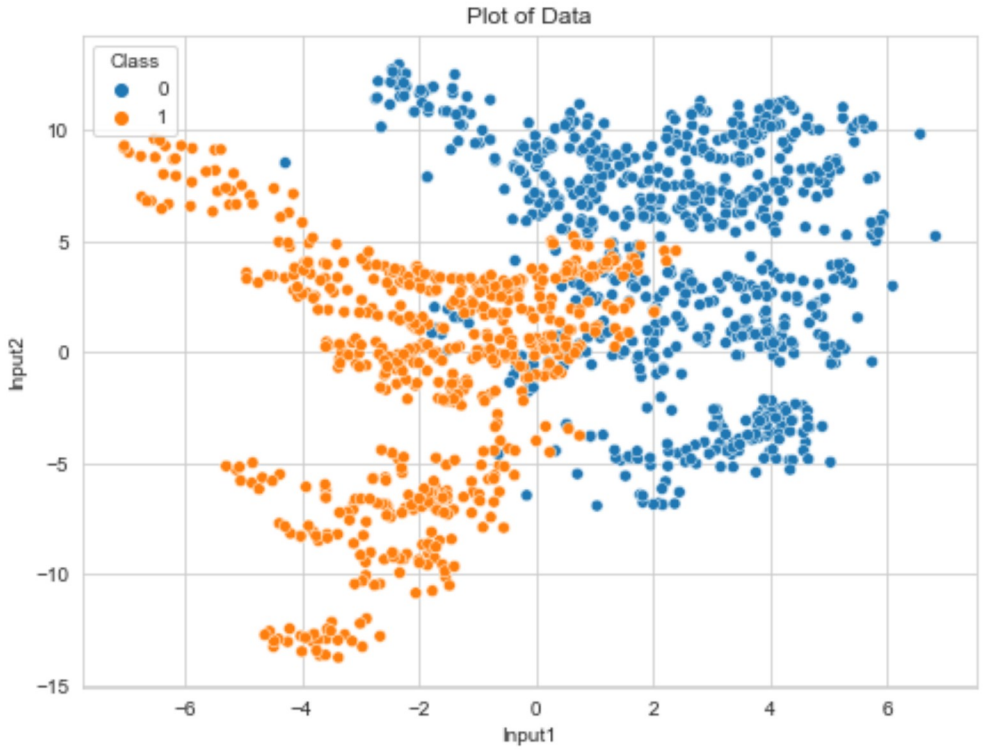
In [40]:
```
sns.heatmap(df.corr(), annot=True, cmap='viridis')
plt.show()
```



The 2 input variables have high correlation with the class labels

Plotting the data

In [41]:
```
plt.figure(figsize=(8,6))
sns.scatterplot(x='Input1', y='Input2', hue='Class', data=df, color='orange')
plt.title("Plot of Data")
plt.show()
```



In [42]:
```
x = df.drop('Class', axis=1).values
y = df['Class'].values
```

Implementing Logistic Regression using Gradient Descent

Initializing the Weights

In [43]:
```
def initialize_weights(dim):
    ''' In this function, we will initialize our weights and bias'''
    w = np.zeros((len(dim)), dtype = float)
    b = 0
    return w,b
```

Sigmoid function

$sigmoid(z) = 1/(1 + exp(-z))$

In [44]:
```
def sigmoid(z):
    ''' In this function, we will return sigmoid of z'''
    return 1/(1+math.exp(-z))
```

Logistic Loss Function

$logloss = -1 * \frac{1}{n}\Sigma_{foreachYt,Y_{pred}}(Yt * log10(Y_{pred}) + (1 - Yt) * log10(1 - Y_{pred}))$

In [45]:
```
def logloss(y_true,y_pred):
    '''In this function, we will compute log loss '''
    computed_loss = 0
    for idx in range(len(y_true)):
        computed_loss += y_true[idx] * np.log10(y_pred[idx]) + (1-y_true[idx])
    loss = (-1)*computed_loss/len(y_true)
    return loss
```

Compute gradient w.r.to 'w'

$dw^{(t)} = -x_n * y_n * (1 - \sigma((w^{(t)})^T x_n + b^t))$

In [55]:
```
def gradient_dw(x,y,w,b):
    '''In this function, we will compute the gardient w.r.to w '''
    z = np.dot(w,x)+b
    dw = -x * y*(1-sigmoid(z))
    return dw
```

Compute gradient w.r.to 'b'

$db^{(t)} = -y_n * (1 - \sigma((w^{(t)})^T x_n + b^t))$

In [47]:
```
def gradient_db(x,y,w,b):
    '''In this function, we will compute gradient w.r.to b '''
    z = np.dot(w, x) + b
    db = -y*(1 - sigmoid(z))
    return db
```

Computing Predicted probability

In [48]:
```
def pred(X, w, b):
    N = len(X)
    predict = []
    for i in range(N):
        z=np.dot(w,X[i])+b
        predict.append(sigmoid(z))
    return np.array(predict)
```

Implementing Logistic Regression with Gradient Descent

In [56]:

```
def GDLogisticRegression(x,y,epochs,eta0):
    ''' In this function, we will implement logistic regression'''
     #Initialzing weights
    w, b = initialize_weights(x[0])

    #Creating empty lists to store the loss, slope and intercept values for e
    loss_values = []
    w_values = []
    b_values = []

    for epoch in range(epochs):#Loop runs for no. of epochs
        dw = 0
        db = 0
        for idx in range(len(x)): # for each datapoint

            #Calculating Gradient Descent for slope
            dw += gradient_dw(x[idx], y[idx], w, b)
            #Calculating Gradient Descent for Intercept
            db += gradient_db(x[idx], y[idx], w, b)

        #updating the slope value
        w = w - eta0 * dw/len(x)

        #updating the intercept value
        b = b - eta0 * db/len(x)

        #appending values to the list
        w_values.append(w)
        b_values.append(b)

        #Predicting target variable for Train data and computing loss for the
        y_pred = pred(x, w, b)
        loss_epoch = logloss(y,y_pred)
        loss_values.append(loss_epoch) #Appending to the list

        print(f"epoch{epoch+1}: loss = {loss_epoch}")

    return w_values, b_values, loss_values
```

Calling the function

In [58]:

```
eta0=0.01 #Learning rate
epochs= 200 #No. of iterations

w_values, b_values, loss_values = GDLogisticRegression(x,y,epochs,eta0)
```

```
epoch1: loss = 0.29786984234961655
epoch2: loss = 0.29482440035215884
epoch3: loss = 0.29188810118775393
epoch4: loss = 0.28905564689763585
epoch5: loss = 0.2863220040967839
epoch6: loss = 0.2836823966821441
epoch7: loss = 0.2811322975045238
epoch8: loss = 0.2786674192709227
epoch9: loss = 0.2762837049081551
epoch10: loss = 0.2739773175849205
epoch11: loss = 0.2717446305584213
```

```
epoch12: loss = 0.2695822169835609
epoch13: loss = 0.26748683979762344
epoch14: loss = 0.26545544177128033
epoch15: loss = 0.2634851357974713
epoch16: loss = 0.26157319547316155
epoch17: loss = 0.25971704601484874
epoch18: loss = 0.25791425553674185
epoch19: loss = 0.25616252671064726
epoch20: loss = 0.2544596888183571
epoch21: loss = 0.2528036902006429
epoch22: loss = 0.25119259110161474
epoch23: loss = 0.24962455690288307
epoch24: loss = 0.24809785173871834
epoch25: loss = 0.24661083248080623
epoch26: loss = 0.2451619430794252
epoch27: loss = 0.24374970924652026
epoch28: loss = 0.24237273346531565
epoch29: loss = 0.24102969031065277
epoch30: loss = 0.23971932206401483
epoch31: loss = 0.2384404346072914
epoch32: loss = 0.23719189357953696
epoch33: loss = 0.23597262078138023
epoch34: loss = 0.23478159081218725
epoch35: loss = 0.23361782792567107
epoch36: loss = 0.23248040309022222
epoch37: loss = 0.2313684312408822
epoch38: loss = 0.23028106871055484
epoch39: loss = 0.2292175108286801
epoch40: loss = 0.22817698967629244
epoch41: loss = 0.22715877198701215
epoch42: loss = 0.22616215718414467
epoch43: loss = 0.22518647554468807
epoch44: loss = 0.22423108648163048
epoch45: loss = 0.22329537693646248
epoch46: loss = 0.22237875987437852
epoch47: loss = 0.22148067287514667
epoch48: loss = 0.22060057681308434
epoch49: loss = 0.21973795462003803
epoch50: loss = 0.21889231012569627
epoch51: loss = 0.21806316696993205
epoch52: loss = 0.21725006758227156
epoch53: loss = 0.2164525722239072
epoch54: loss = 0.21567025808801246
epoch55: loss = 0.214902718454411
epoch56: loss = 0.21414956189492035
epoch57: loss = 0.21341041152598045
epoch58: loss = 0.2126849043053857
epoch59: loss = 0.21197269037019095
epoch60: loss = 0.21127343241304905
epoch61: loss = 0.21058680509446043
epoch62: loss = 0.20991249448855737
epoch63: loss = 0.20925019756024532
epoch64: loss = 0.20859962167166657
epoch65: loss = 0.20796048411608506
epoch66: loss = 0.2073325116774461
epoch67: loss = 0.20671544021396263
epoch68: loss = 0.20610901426421763
epoch69: loss = 0.2055129866743669
epoch70: loss = 0.20492711824511153
epoch71: loss = 0.20435117739723607
epoch72: loss = 0.203784939854553
epoch73: loss = 0.20322818834319478
epoch74: loss = 0.20268071230626947
epoch75: loss = 0.20214230763294597
epoch76: loss = 0.2016127764011137
```

```
epoch77: loss = 0.20109192663280911
epoch78: loss = 0.20057957206166424
epoch79: loss = 0.20007553191166938
epoch80: loss = 0.19957963068660398
epoch81: loss = 0.1990916979695157
epoch82: loss = 0.198611568231685
epoch83: loss = 0.19813908065053226
epoch84: loss = 0.1976740789359747
epoch85: loss = 0.19721641116476005
epoch86: loss = 0.19676592962233722
epoch87: loss = 0.19632249065186236
epoch88: loss = 0.19588595450994278
epoch89: loss = 0.19545618522876387
epoch90: loss = 0.19503305048426287
epoch91: loss = 0.1946164214700273
epoch92: loss = 0.1942061727766158
epoch93: loss = 0.19380218227603055
epoch94: loss = 0.19340433101106957
epoch95: loss = 0.19301250308930937
epoch96: loss = 0.19262658558149662
epoch97: loss = 0.1922464684241093
epoch98: loss = 0.19187204432589966
epoch99: loss = 0.1915032086782129
epoch100: loss = 0.19113985946889775
epoch101: loss = 0.19078189719964092
epoch102: loss = 0.19042922480655938
epoch103: loss = 0.19008174758389063
epoch104: loss = 0.1897393731106446
epoch105: loss = 0.1894020111800709
epoch106: loss = 0.18906957373181785
epoch107: loss = 0.18874197478665244
epoch108: loss = 0.18841913038362984
epoch109: loss = 0.18810095851960526
epoch110: loss = 0.18778737909097087
epoch111: loss = 0.1874783138375362
epoch112: loss = 0.18717368628843853
epoch113: loss = 0.18687342171001434
epoch114: loss = 0.1865774470555298
epoch115: loss = 0.18628569091670347
epoch116: loss = 0.18599808347693297
epoch117: loss = 0.1857145564661663
epoch118: loss = 0.18543504311733608
epoch119: loss = 0.18515947812430336
epoch120: loss = 0.18488779760123586
epoch121: loss = 0.18461993904337726
epoch122: loss = 0.18435584128913401
epoch123: loss = 0.18409544448344256
epoch124: loss = 0.1838386900423508
epoch125: loss = 0.18358552061877936
epoch126: loss = 0.1833358800694017
epoch127: loss = 0.18308971342261537
epoch128: loss = 0.18284696684754287
epoch129: loss = 0.18260758762404397
epoch130: loss = 0.1823715241136774
epoch131: loss = 0.18213872573159895
epoch132: loss = 0.18190914291933744
epoch133: loss = 0.18168272711843708
epoch134: loss = 0.18145943074491844
epoch135: loss = 0.18123920716453143
epoch136: loss = 0.1810220106687791
epoch137: loss = 0.18080779645167055
epoch138: loss = 0.18059652058718548
epoch139: loss = 0.18038814000742615
epoch140: loss = 0.18018261248142167
epoch141: loss = 0.17997989659457356
```

```
epoch142: loss = 0.17977995172871017
epoch143: loss = 0.17958273804273706
epoch144: loss = 0.17938821645385075
epoch145: loss = 0.17919634861930983
epoch146: loss = 0.17900709691873654
epoch147: loss = 0.1788204244369263
epoch148: loss = 0.178636294947159
epoch149: loss = 0.1784546728949863
epoch150: loss = 0.1782755233824824
epoch151: loss = 0.17809881215293932
epoch152: loss = 0.17792450557599754
epoch153: loss = 0.1777525706331927
epoch154: loss = 0.17758297490390434
epoch155: loss = 0.17741568655169906
epoch156: loss = 0.17725067431104807
epoch157: loss = 0.17708790747440942
epoch158: loss = 0.17692735587966907
epoch159: loss = 0.17676898989791776
epoch160: loss = 0.1766127804215633
epoch161: loss = 0.1764586988527602
epoch162: loss = 0.17630671709215293
epoch163: loss = 0.17615680752791846
epoch164: loss = 0.17600894302509942
epoch165: loss = 0.17586309691522067
epoch166: loss = 0.175719242986177
epoch167: loss = 0.17557735547238817
epoch168: loss = 0.1754374090452099
epoch169: loss = 0.1752993788035939
epoch170: loss = 0.1751632402649871
epoch171: loss = 0.17502896935646747
epoch172: loss = 0.17489654240610614
epoch173: loss = 0.17476593613455055
epoch174: loss = 0.17463712764681494
epoch175: loss = 0.17451009442429075
epoch176: loss = 0.1743848143169419
epoch177: loss = 0.1742612655357105
epoch178: loss = 0.1741394266451001
epoch179: loss = 0.17401927655595073
epoch180: loss = 0.17390079451838633
epoch181: loss = 0.17378396011494215
epoch182: loss = 0.17366875325385364
epoch183: loss = 0.17355515416251324
epoch184: loss = 0.17344314338108682
epoch185: loss = 0.17333270175627785
epoch186: loss = 0.17322381043525045
epoch187: loss = 0.17311645085969216
epoch188: loss = 0.17301060476002011
epoch189: loss = 0.17290625414972416
epoch190: loss = 0.17280338131984524
epoch191: loss = 0.1727019688335845
epoch192: loss = 0.17260199952103356
epoch193: loss = 0.17250345647403664
epoch194: loss = 0.1724063230411647
epoch195: loss = 0.17231058282280914
epoch196: loss = 0.17221621966639022
epoch197: loss = 0.1721232176616732
epoch198: loss = 0.17203156113619003
epoch199: loss = 0.17194123465077427
```

Getting the best values for slope and Intercept

In [51]:

```
#Initializing weights
w_opt = 0
b_opt = 0

for idx in range(1, len(loss_values)):
    loss_diff = loss_values[idx-1] - loss_values[idx]

    if loss_diff < 0.0001 : # checking if the loss values are improved
        print('for epoch {} - Minimun loss: {}'.format(idx, loss_values[idx]))
        w_opt = w_values[idx-1]
        b_opt = b_values[idx-1]
        break # Terminating the loop if there is no improvment in the loss
    else:
        continue
print("Best slope: ", w_opt)
print("Best intercept: ", b_opt)
```

```
for epoch 191 - Minimun loss: 0.17260199952103356
Best slope:  [-0.40533935 -0.04227058]
Best intercept:  0.3009541078958172
```

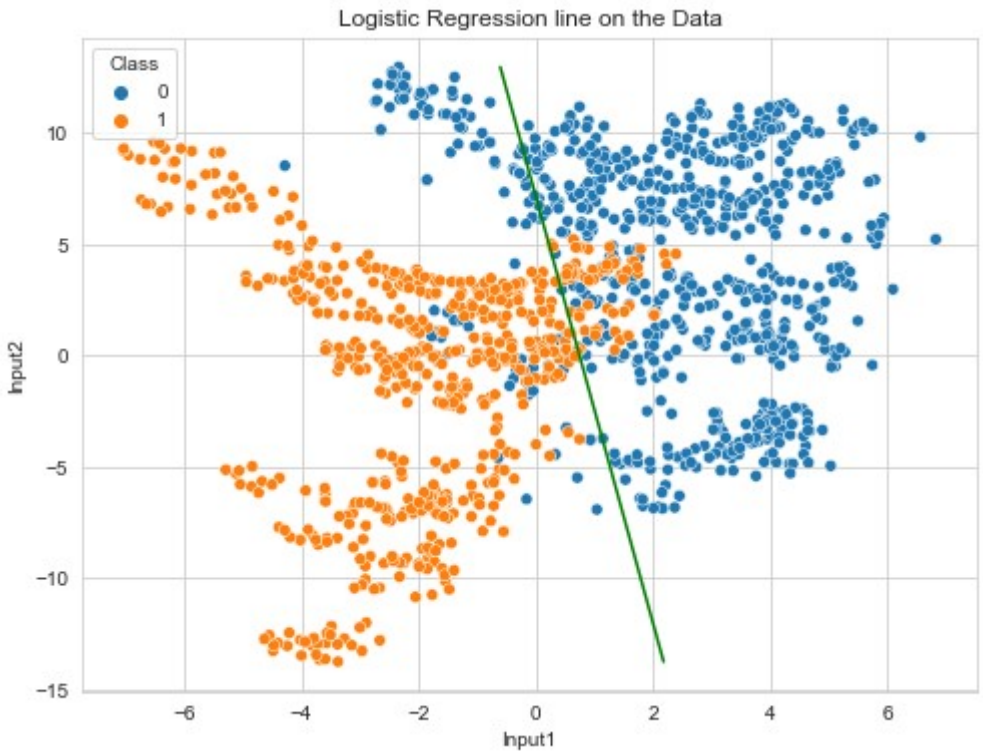
Plotting the regression line

In [60]:

```
def draw_line(coef,intercept, mi, ma):
    # for the separating hyper plane ax+by+c=0, the weights are [a, b] and the
    # to draw the hyper plane we are creating two points
    # 1. ((b*min-c)/a, min) i.e ax+by+c=0 ==> ax = (-by-c) ==> x = (-by-c)/a
    # 2. ((b*max-c)/a, max) i.e ax+by+c=0 ==> ax = (-by-c) ==> x = (-by-c)/a
    points=np.array([[((-coef[1]*mi - intercept)/coef[0]), mi],[((-coef[1]*ma
    plt.plot(points[:,0], points[:,1], c = 'green')
```

In [54]:

```
y_pred = pred(x, w_opt, b_opt)
plt.figure(figsize=(8,6))
sns.scatterplot(x='Input1', y='Input2', hue='Class', data=df, color='orange')
draw_line(w_opt, b_opt, min(x[:,1]), max(x[:,1]))
plt.title("Logistic Regression line on the Data")
plt.show()
```


Logistic Regression line on the Data
Class
0
1
10
5
0
−5
−10
−15
Input2
−6
−4
−2
0
2
4
6
Input1