**Name** - Akash Kumar Singh

**Reg No** - RA1911003010667

# AI LAB EXPERIMENT N0: 10

**Implementation of a learning algorithm**

# WORKING PRINCIPLE:-

Linear regression shows the linear relationship between the independent variable (X-axis) and the dependent variable (Y-axis).To calculate best-fit line linear regression uses a traditional slope-intercept form. A regression line can be a Positive Linear Relationship or a Negative Linear Relationship.The goal of the linear regression algorithm is to get the best values for a0 and a1 to find the

best fit line and the best fit line should have the least error. In Linear Regression, Mean Squared Error (MSE) cost function is used, which helps to figure out the best possible values for a0 and a1, which provides the best fit line for the data points. Using the MSE function, we will change the values of a0 and a1 such that the MSE value settles at the minima.

Gradient descent is a method of updating a0 and a1 to minimize the cost function(MSE).

Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, the decision tree algorithm can be used for solving **regression and classification problems** too.

The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by **learning simple decision rules** inferred from prior data(training data).

In Decision Trees, for predicting a class label for a record we start from the **root** of the tree. We compare the values of the root attribute with the record’s attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node.

### Types of Decision Trees

Types of decision trees are based on the type of target variable we have. It can be of two types:

1. **Categorical Variable Decision Tree:**Decision Tree which has a categorical target variable then it called a **Categorical variable decision tree.**
2. **Continuous Variable Decision Tree:**Decision Tree has a continuous target variable then it is called **Continuous Variable Decision Tree.**

# CODE:-

pip install termcolor

import pandas as pd # data processing

import numpy as np # working with arrays

import matplotlib.pyplot as plt # visualization

from termcolor import colored as cl # text customization

import itertools # advanced tools

from sklearn.preprocessing import StandardScaler # data normalization

from sklearn.model\_selection import train\_test\_split # data split

from sklearn.tree import DecisionTreeClassifier # Decision tree algorithm

from sklearn.neighbors import KNeighborsClassifier # KNN algorithm

from sklearn.linear\_model import LogisticRegression # Logistic regression algorithm

from sklearn.svm import SVC # SVM algorithm

from sklearn.ensemble import RandomForestClassifier # Random forest tree algorithm

from xgboost import XGBClassifier # XGBoost algorithm

from sklearn.metrics import confusion\_matrix # evaluation metric

from sklearn.metrics import accuracy\_score # evaluation metric

from sklearn.metrics import f1\_score # evaluation metric

df = pd.read\_csv('creditcard.csv')

df.drop('Time', axis = 1, inplace = True)

[{"metadata":{"trusted":false},"cell\_type":"code","source":"print(df.head())\n","execution\_count":12,"outputs":[{"name":"stdout","output\_type":"stream","text":" V1 V2 V3 V4 V5 V6 V7 \\\n0 -1.359807 -0.072781 2.536347 1.378155 -0.338321 0.462388 0.239599 \n1 1.191857 0.266151 0.166480 0.448154 0.060018 -0.082361 -0.078803 \n2 -1.358354 -1.340163 1.773209 0.379780 -0.503198 1.800499 0.791461 \n3 -0.966272 -0.185226 1.792993 -0.863291 -0.010309 1.247203 0.237609 \n4 -1.158233 0.877737 1.548718 0.403034 -0.407193 0.095921 0.592941 \n\n V8 V9 V10 ... V21 V22 V23 V24 \\\n0 0.098698 0.363787 0.090794 ... -0.018307 0.277838 -0.110474 0.066928 \n1 0.085102 -0.255425 -0.166974 ... -0.225775 -0.638672 0.101288 -0.339846 \n2 0.247676 -1.514654 0.207643 ... 0.247998 0.771679 0.909412 -0.689281 \n3 0.377436 -1.387024 -0.054952 ... -0.108300 0.005274 -0.190321 -1.175575 \n4 -0.270533 0.817739 0.753074 ... -0.009431 0.798278 -0.137458 0.141267 \n\n V25 V26 V27 V28 Amount Class \n0 0.128539 -0.189115 0.133558 -0.021053 149.62 0 \n1 0.167170 0.125895 -0.008983 0.014724 2.69 0 \n2 -0.327642 -0.139097 -0.055353 -0.059752 378.66 0 \n3 0.647376 -0.221929 0.062723 0.061458 123.50 0 \n4 -0.206010 0.502292 0.219422 0.215153 69.99 0 \n\n[5 rows x 30 columns]\n"}]}]

cases = len(df)

nonfraud\_count = len(df[df.Class == 0])

fraud\_count = len(df[df.Class == 1])

fraud\_percentage = round(fraud\_count/nonfraud\_count\*100, 2)

print(cl('CASE COUNT', attrs = ['bold']))

print(cl('--------------------------------------------', attrs = ['bold']))

print(cl('Total number of cases are {}'.format(cases), attrs = ['bold']))

print(cl('Number of Non-fraud cases are {}'.format(nonfraud\_count), attrs = ['bold']))

print(cl('Number of Non-fraud cases are {}'.format(fraud\_count), attrs = ['bold']))

print(cl('Percentage of fraud cases is {}'.format(fraud\_percentage), attrs = ['bold']))

print(cl('--------------------------------------------', attrs = ['bold']))

nonfraud\_cases = df[df.Class == 0]

fraud\_cases = df[df.Class == 1]

print(cl('CASE AMOUNT STATISTICS', attrs = ['bold']))

print(cl('--------------------------------------------', attrs = ['bold']))

print(cl('NON-FRAUD CASE AMOUNT STATS', attrs = ['bold']))

print(nonfraud\_cases.Amount.describe())

print(cl('--------------------------------------------', attrs = ['bold']))

print(cl('FRAUD CASE AMOUNT STATS', attrs = ['bold']))

print(fraud\_cases.Amount.describe())

print(cl('--------------------------------------------', attrs = ['bold']))

X = df.drop('Class', axis = 1).values

y = df['Class'].values

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0)

print(cl('X\_train samples : ', attrs = ['bold']), X\_train[:1])

print(cl('X\_test samples : ', attrs = ['bold']), X\_test[0:1])

print(cl('y\_train samples : ', attrs = ['bold']), y\_train[0:10])

print(cl('y\_test samples : ', attrs = ['bold']), y\_test[0:10])

from sklearn.preprocessing import StandardScaler

tree\_model = DecisionTreeClassifier(max\_depth = 4, criterion = 'entropy')

tree\_model.fit(X\_train, y\_train)

tree\_yhat = tree\_model.predict(X\_test)

# 2. K-Nearest Neighbors

n = 5

knn = KNeighborsClassifier(n\_neighbors = n)

knn.fit(X\_train, y\_train)

knn\_yhat = knn.predict(X\_test)

# 3. Logistic Regression

lr = LogisticRegression()

lr.fit(X\_train, y\_train)

lr\_yhat = lr.predict(X\_test)

# 4. SVM

svm = SVC()

svm.fit(X\_train, y\_train)

svm\_yhat = svm.predict(X\_test)

# 5. Random Forest Tree

rf = RandomForestClassifier(max\_depth = 4)

rf.fit(X\_train, y\_train)

rf\_yhat = rf.predict(X\_test)

# 6. XGBoost

xgb = XGBClassifier(max\_depth = 4)

xgb.fit(X\_train, y\_train)

xgb\_yhat = xgb.predict(X\_test)

print(cl('ACCURACY SCORE', attrs = ['bold']))

print(cl('------------------------------------------------------------------------', attrs = ['bold']))

print(cl('Accuracy score of the Decision Tree model is {}'.format(accuracy\_score(y\_test, tree\_yhat)), attrs = ['bold']))

print(cl('------------------------------------------------------------------------', attrs = ['bold']))

print(cl('Accuracy score of the KNN model is {}'.format(accuracy\_score(y\_test, knn\_yhat)), attrs = ['bold'], color = 'green'))

print(cl('------------------------------------------------------------------------', attrs = ['bold']))

print(cl('Accuracy score of the Logistic Regression model is {}'.format(accuracy\_score(y\_test, lr\_yhat)), attrs = ['bold'], color = 'red'))

print(cl('------------------------------------------------------------------------', attrs = ['bold']))

print(cl('Accuracy score of the SVM model is {}'.format(accuracy\_score(y\_test, svm\_yhat)), attrs = ['bold']))

print(cl('------------------------------------------------------------------------', attrs = ['bold']))

print(cl('Accuracy score of the Random Forest Tree model is {}'.format(accuracy\_score(y\_test, rf\_yhat)), attrs = ['bold']))

print(cl('------------------------------------------------------------------------', attrs = ['bold']))

print(cl('Accuracy score of the XGBoost model is {}'.format(accuracy\_score(y\_test, xgb\_yhat)), attrs = ['bold']))

print(cl('------------------------------------------------------------------------', attrs = ['bold']))

def plot\_confusion\_matrix(cm, classes, title, normalize = False, cmap = plt.cm.Blues):

title = 'Confusion Matrix of {}'.format(title)

if normalize:

cm = cm.astype(float) / cm.sum(axis=1)[:, np.newaxis]

plt.imshow(cm, interpolation = 'nearest', cmap = cmap)

plt.title(title)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation = 45)

plt.yticks(tick\_marks, classes)

fmt = '.2f' if normalize else 'd'

thresh = cm.max() / 2.

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.text(j, i, format(cm[i, j], fmt),

horizontalalignment = 'center',

color = 'white' if cm[i, j] > thresh else 'black')

plt.tight\_layout()

plt.ylabel('True label')

plt.xlabel('Predicted label')

# Compute confusion matrix for the models

tree\_matrix = confusion\_matrix(y\_test, tree\_yhat, labels = [0, 1]) # Decision Tree

knn\_matrix = confusion\_matrix(y\_test, knn\_yhat, labels = [0, 1]) # K-Nearest Neighbors

lr\_matrix = confusion\_matrix(y\_test, lr\_yhat, labels = [0, 1]) # Logistic Regression

svm\_matrix = confusion\_matrix(y\_test, svm\_yhat, labels = [0, 1]) # Support Vector Machine

rf\_matrix = confusion\_matrix(y\_test, rf\_yhat, labels = [0, 1]) # Random Forest Tree

xgb\_matrix = confusion\_matrix(y\_test, xgb\_yhat, labels = [0, 1]) # XGBoost

# Plot the confusion matrix

plt.rcParams['figure.figsize'] = (6, 6)

# 1. Decision tree

tree\_cm\_plot = plot\_confusion\_matrix(tree\_matrix,

classes = ['Non-Default(0)','Default(1)'],

normalize = False, title = 'Decision Tree')

plt.savefig('tree\_cm\_plot.png')

plt.show()

# 2. K-Nearest Neighbors

knn\_cm\_plot = plot\_confusion\_matrix(knn\_matrix,

classes = ['Non-Default(0)','Default(1)'],

normalize = False, title = 'KNN')

plt.savefig('knn\_cm\_plot.png')

plt.show()

# 3. Logistic regression

lr\_cm\_plot = plot\_confusion\_matrix(lr\_matrix,

classes = ['Non-Default(0)','Default(1)'],

normalize = False, title = 'Logistic Regression')

plt.savefig('lr\_cm\_plot.png')

plt.show()

# 4. Support Vector Machine

svm\_cm\_plot = plot\_confusion\_matrix(svm\_matrix,

classes = ['Non-Default(0)','Default(1)'],

normalize = False, title = 'SVM')

plt.savefig('svm\_cm\_plot.png')

plt.show()

# 5. Random forest tree

rf\_cm\_plot = plot\_confusion\_matrix(rf\_matrix,

classes = ['Non-Default(0)','Default(1)'],

normalize = False, title = 'Random Forest Tree')

plt.savefig('rf\_cm\_plot.png')

plt.show()

# 6. XGBoost

xgb\_cm\_plot = plot\_confusion\_matrix(xgb\_matrix,

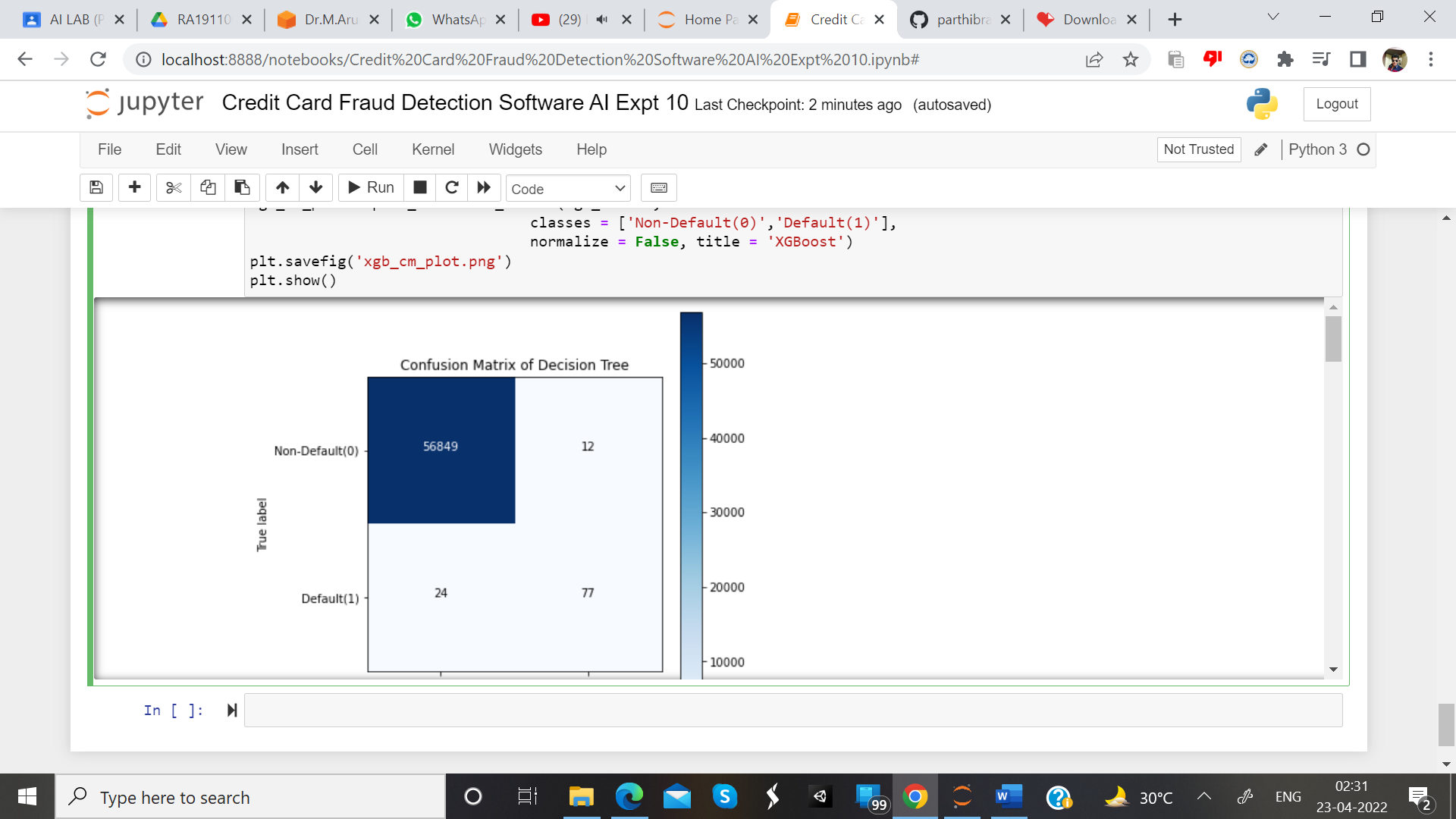
classes = ['Non-Default(0)','Default(1)'],

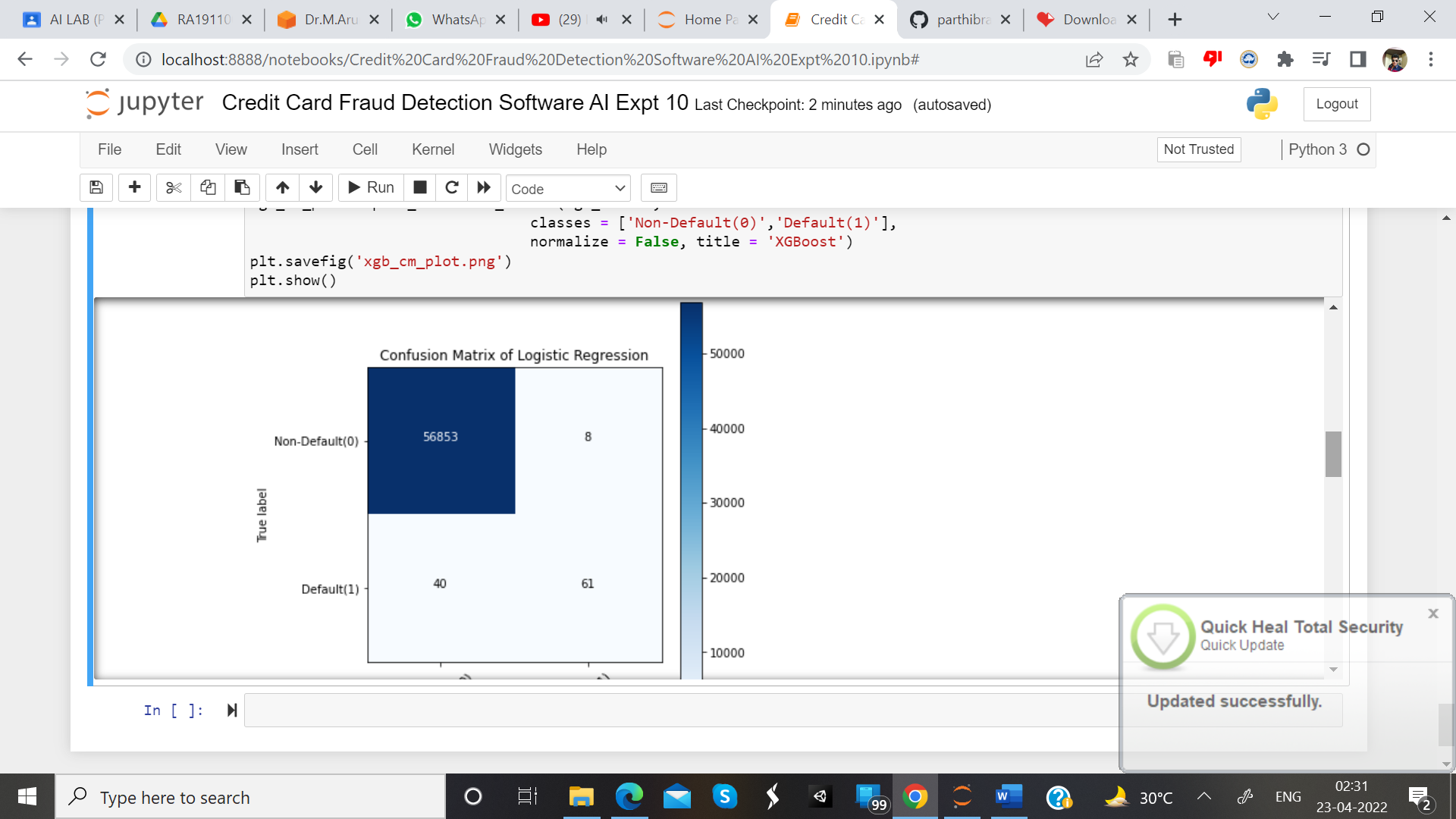
normalize = False, title = 'XGBoost')

plt.savefig('xgb\_cm\_plot.png')

plt.show()

**Output:-**





**RESULT:-**

Hence, the Implementation of a machine learning algorithm is done successfully.