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":"stre
am","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,
```

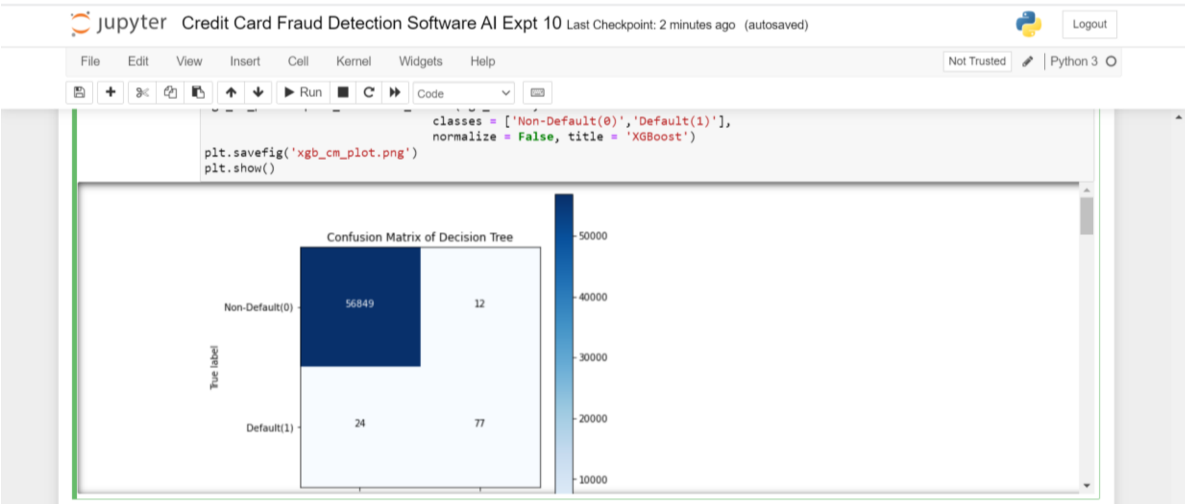
```
random_state = 0)

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']))
```

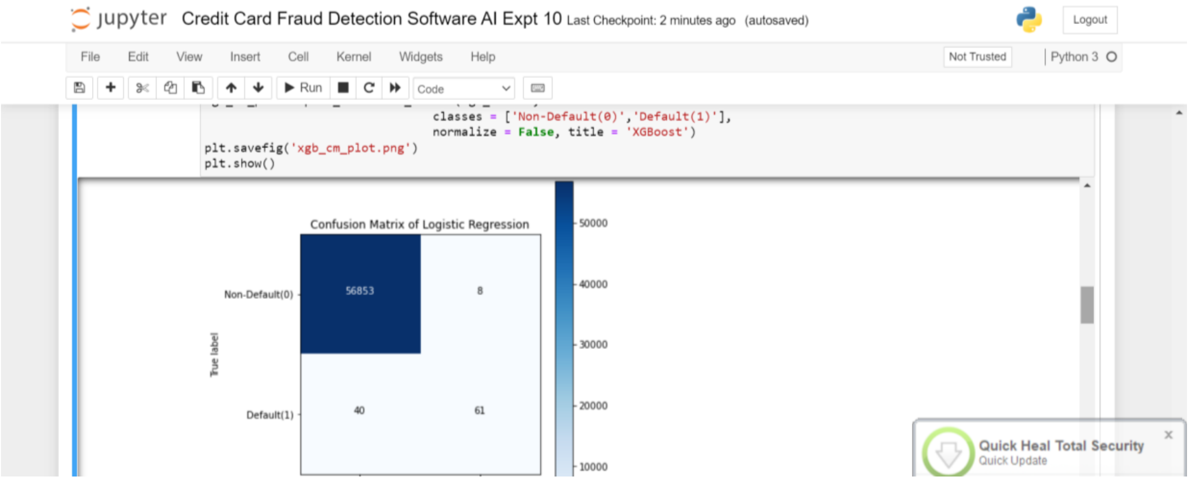
```
print(cl('----------------------------------------------------------------------', attrs =
['bold']))
print(cl('Accuracy score of the Decision Tree model is
{}'.format(accuracy_score(y_test, tree_yhat)), attrs = ['bold']))
print(cl('----------------------------------------------------------------------', 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 learningalgorithm is done successfully.

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

```
pip install termcolor
```

```
Collecting termcolor
  Downloading termcolor-1.1.0.tar.gz (3.9 kB)
Building wheels for collected packages: termcolor
  Building wheel for termcolor (setup.py): started
  Building wheel for termcolor (setup.py): finished with status 'done'
  Created wheel for termcolor: filename=termcolor-1.1.0-py3-none-
any.whl size=4835
sha256=3c605ee7cc816aba43c34b98a2d1d91e79ba4585f0eb9ea4a725dcf590fa83c
a
  Stored in directory: c:\users\hp\appdata\local\pip\cache\wheels\
a0\16\9c\5473df82468f958445479c59e784896fa24f4a5fc024b0f501
Successfully built termcolor
Installing collected packages: termcolor
Successfully installed termcolor-1.1.0
Note: you may need to restart the kernel to use updated packages.
```

```
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
```

```
pip install xgboost
```

```
Collecting xgboost
  Downloading xgboost-1.4.2-py3-none-win_amd64.whl (97.8 MB)
Requirement already satisfied: numpy in c:\users\hp\anaconda3\lib\
site-packages (from xgboost) (1.19.2)
Requirement already satisfied: scipy in c:\users\hp\anaconda3\lib\
site-packages (from xgboost) (1.5.2)
Installing collected packages: xgboost
Successfully installed xgboost-1.4.2
Note: you may need to restart the kernel to use updated packages.
```

```
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)

print(df.head())

         V1        V2        V3        V4        V5        V6
V7  \
0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388
0.239599
1  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -
0.078803
2 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499
0.791461
3 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203
0.237609
4 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921
0.592941

         V8        V9       V10  ...       V21       V22       V23
V24  \
0  0.098698  0.363787  0.090794  ... -0.018307  0.277838 -0.110474
0.066928
1  0.085102 -0.255425 -0.166974  ... -0.225775 -0.638672  0.101288 -
0.339846
2  0.247676 -1.514654  0.207643  ...  0.247998  0.771679  0.909412 -
0.689281
3  0.377436 -1.387024 -0.054952  ... -0.108300  0.005274 -0.190321 -
1.175575
4 -0.270533  0.817739  0.753074  ... -0.009431  0.798278 -0.137458
0.141267

        V25       V26       V27       V28  Amount  Class
0  0.128539 -0.189115  0.133558 -0.021053  149.62      0
1  0.167170  0.125895 -0.008983  0.014724    2.69      0
2 -0.327642 -0.139097 -0.055353 -0.059752  378.66      0
3  0.647376 -0.221929  0.062723  0.061458  123.50      0
4 -0.206010  0.502292  0.219422  0.215153   69.99      0

[5 rows x 30 columns]

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']))
```

```
CASE COUNT
--------------------------------------------
Total number of cases are 284807
Number of Non-fraud cases are 284315
Number of Non-fraud cases are 492
Percentage of fraud cases is 0.17
--------------------------------------------
```

```
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']))
```

```
CASE AMOUNT STATISTICS
--------------------------------------------
NON-FRAUD CASE AMOUNT STATS
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
--------------------------------------------
FRAUD CASE AMOUNT STATS
count     492.000000
mean      122.211321
```

```
std        256.683288
min          0.000000
25%          1.000000
50%          9.250000
75%        105.890000
max       2125.870000
Name: Amount, dtype: float64
--------------------------------------------
```

```
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])
```

```
X_train samples :  [[-1.11504743e+00  1.03558276e+00  8.00712441e-01 -
1.06039825e+00
   3.26211690e-02  8.53422160e-01 -6.14243480e-01 -3.23116112e+00
   1.53994798e+00 -8.16908791e-01 -1.30559201e+00  1.08177199e-01
  -8.59609580e-01 -7.19342108e-02  9.06655628e-01 -1.72092961e+00
   7.97853221e-01 -6.75939779e-03  1.95677806e+00 -6.44895565e-01
   3.02038533e+00 -5.39617976e-01  3.31564886e-02 -7.74945766e-01
   1.05867812e-01 -4.30853482e-01  2.29736936e-01 -7.05913036e-02
   1.29500000e+01]]
X_test samples :  [[-3.23333572e-01  1.05745525e+00 -4.83411518e-02 -
6.07204308e-01
   1.25982115e+00 -9.17607168e-02  1.15910150e+00 -1.24334606e-01
  -1.74639536e-01 -1.64440065e+00 -1.11886302e+00  2.02647310e-01
   1.14596495e+00 -1.80235956e+00 -2.47177932e-01 -6.09453515e-02
   8.46605738e-01  3.79454387e-01  8.47262245e-01  1.86409421e-01
  -2.07098267e-01 -4.33890272e-01 -2.61613283e-01 -4.66506063e-02
   2.11512300e-01  8.29721214e-03  1.08494430e-01  1.61139167e-01
   4.00000000e+01]]
y_train samples :  [0 0 0 0 0 0 0 0 0 0]
y_test samples :  [0 0 0 0 0 0 0 0 0 0]
```

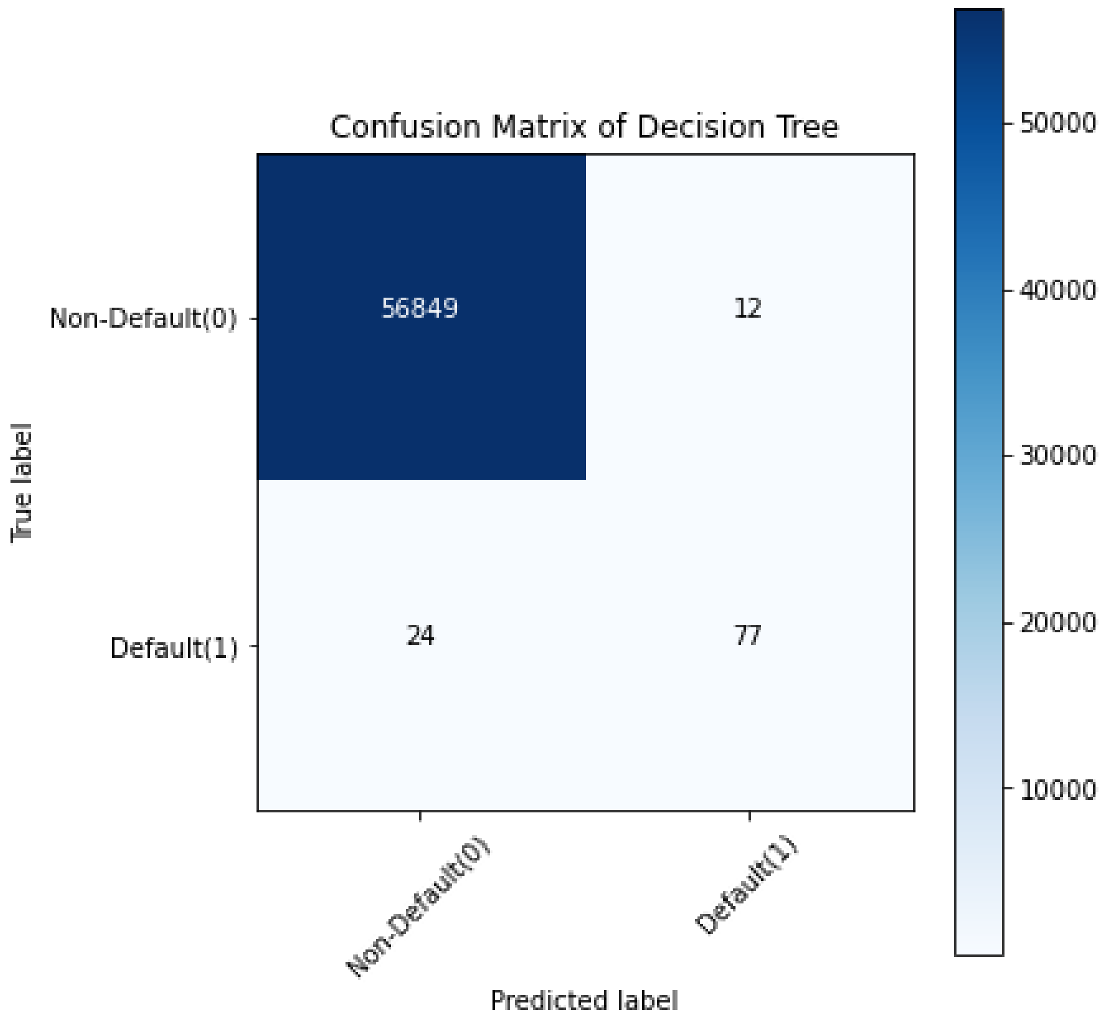
```
from sklearn.preprocessing import StandardScaler

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('------------------------------------------------------------
```

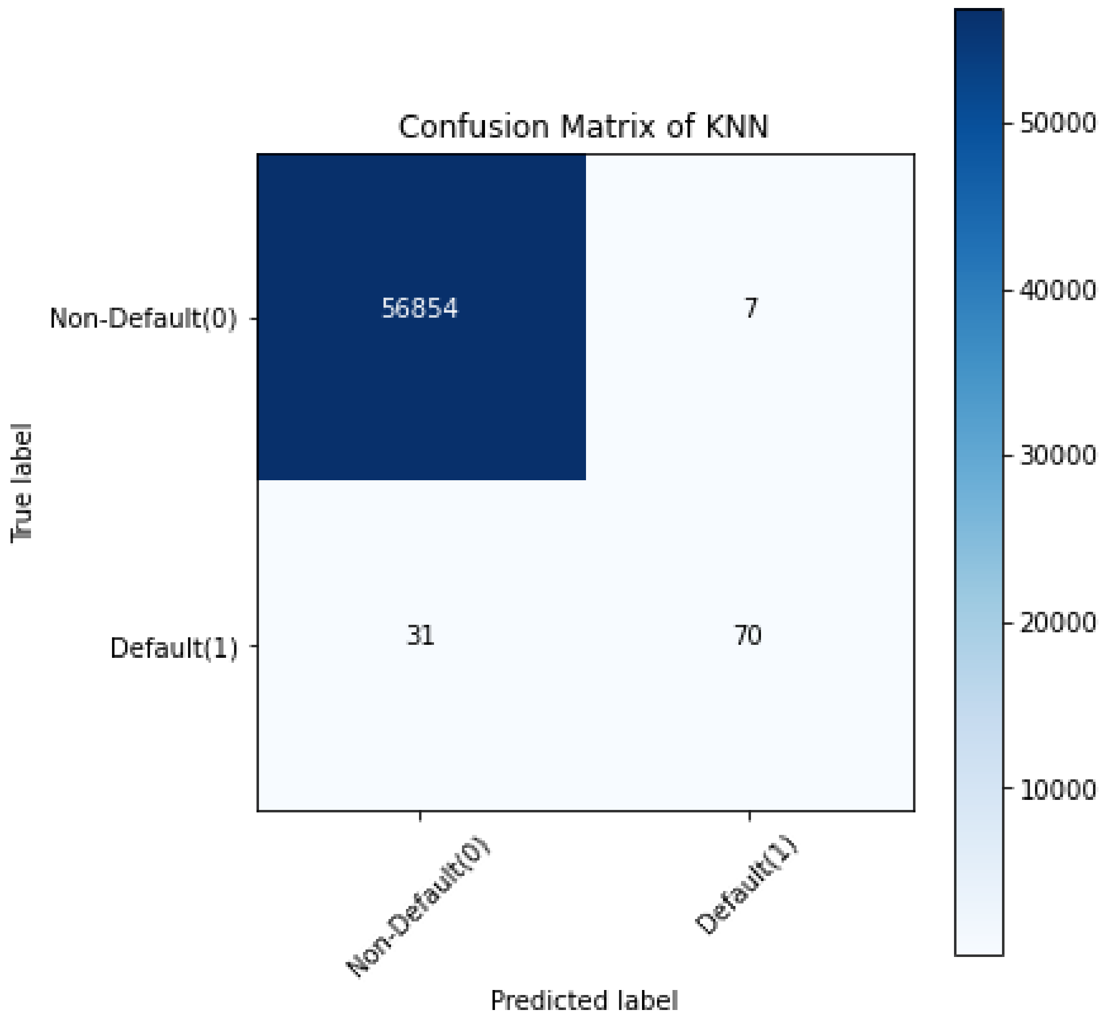
```
------------', attrs = ['bold']))
print(cl('Accuracy score of the Random Forest Tree model is
{}'.format(accuracy_score(y_test, rf_yhat)), attrs = ['bold']))
print(cl('-------------------------------------------------------------
------------', attrs = ['bold']))
print(cl('Accuracy score of the XGBoost model is
{}'.format(accuracy_score(y_test, xgb_yhat)), attrs = ['bold']))
print(cl('-------------------------------------------------------------
------------', attrs = ['bold']))

ACCURACY SCORE
----------------------------------------------------------------------
--
Accuracy score of the Decision Tree model is 0.9993679997191109
----------------------------------------------------------------------
--
Accuracy score of the KNN model is 0.9993328885923949
----------------------------------------------------------------------
--
Accuracy score of the Logistic Regression model is 0.9991573329588147
----------------------------------------------------------------------
--
Accuracy score of the SVM model is 0.998735999438222
----------------------------------------------------------------------
--
Accuracy score of the Random Forest Tree model is 0.9993153330290369
----------------------------------------------------------------------
--
Accuracy score of the XGBoost model is 0.9994908886626171
----------------------------------------------------------------------
--

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),
```

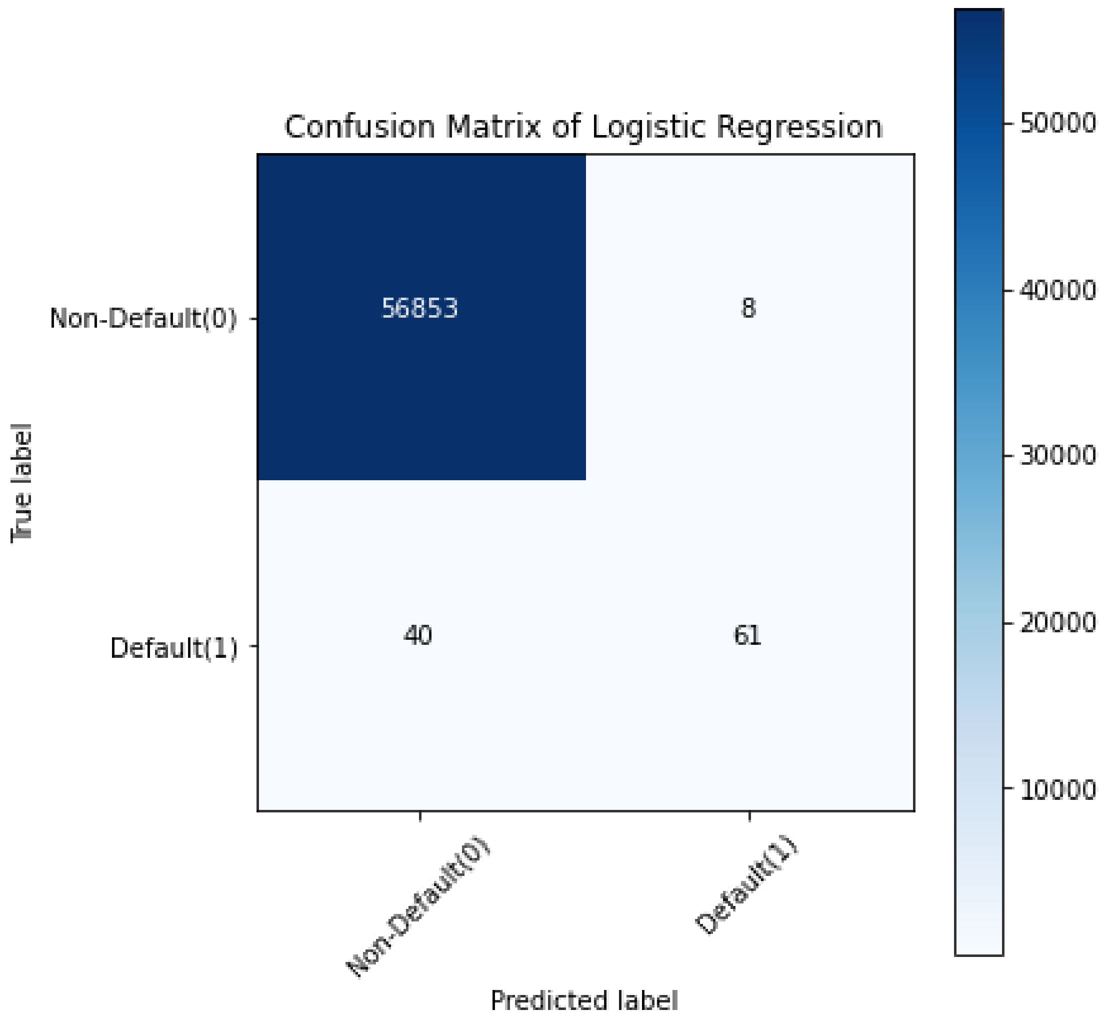
```
                 horizontalalignment = 'center',
                 color = 'white' if cm[i, j] > thresh else 'black')

    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-
```

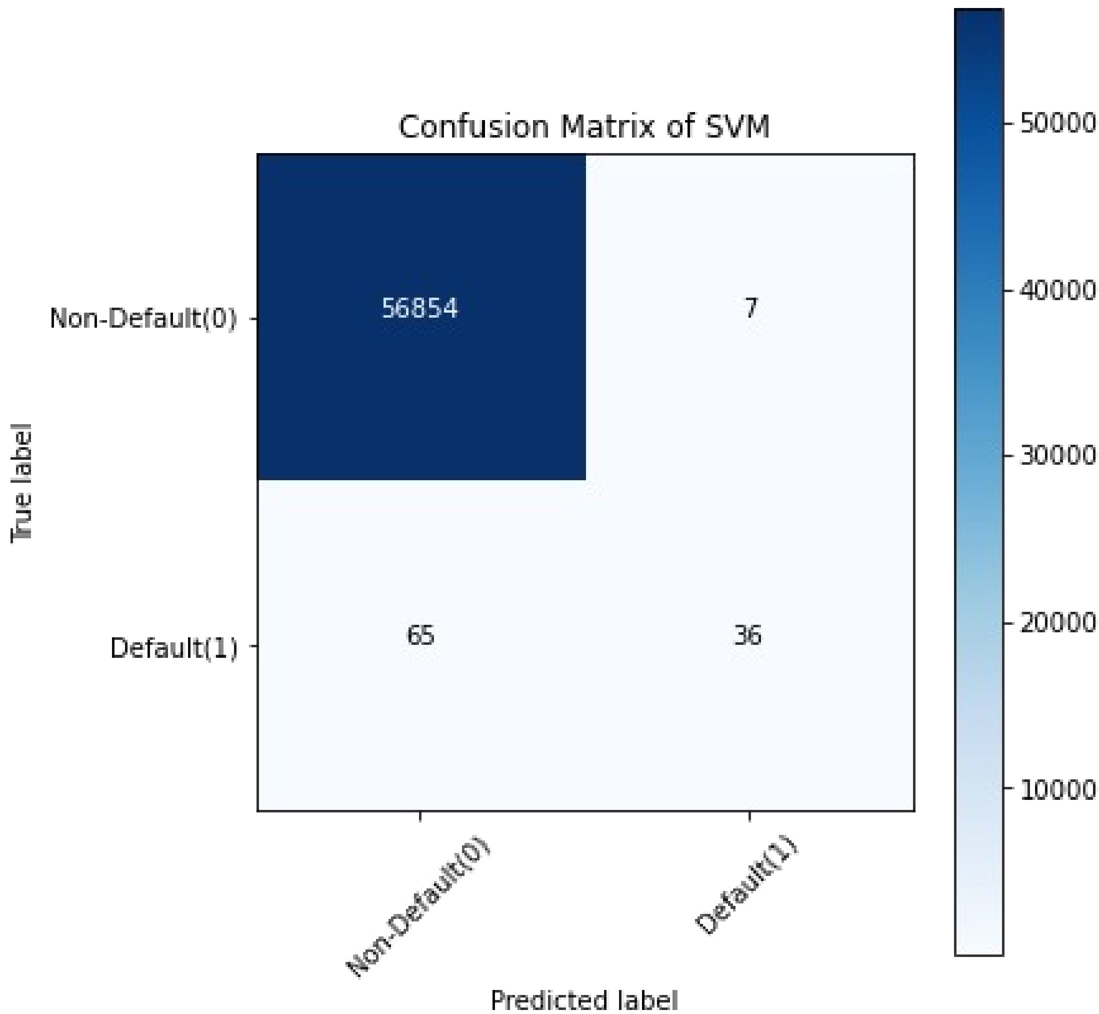
```
Default(0)','Default(1)'],
                                normalize = False, title = 'Logistic 
Regression')
plt.savefig('lr_cm_plot.png')
plt.show()

# 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()
```

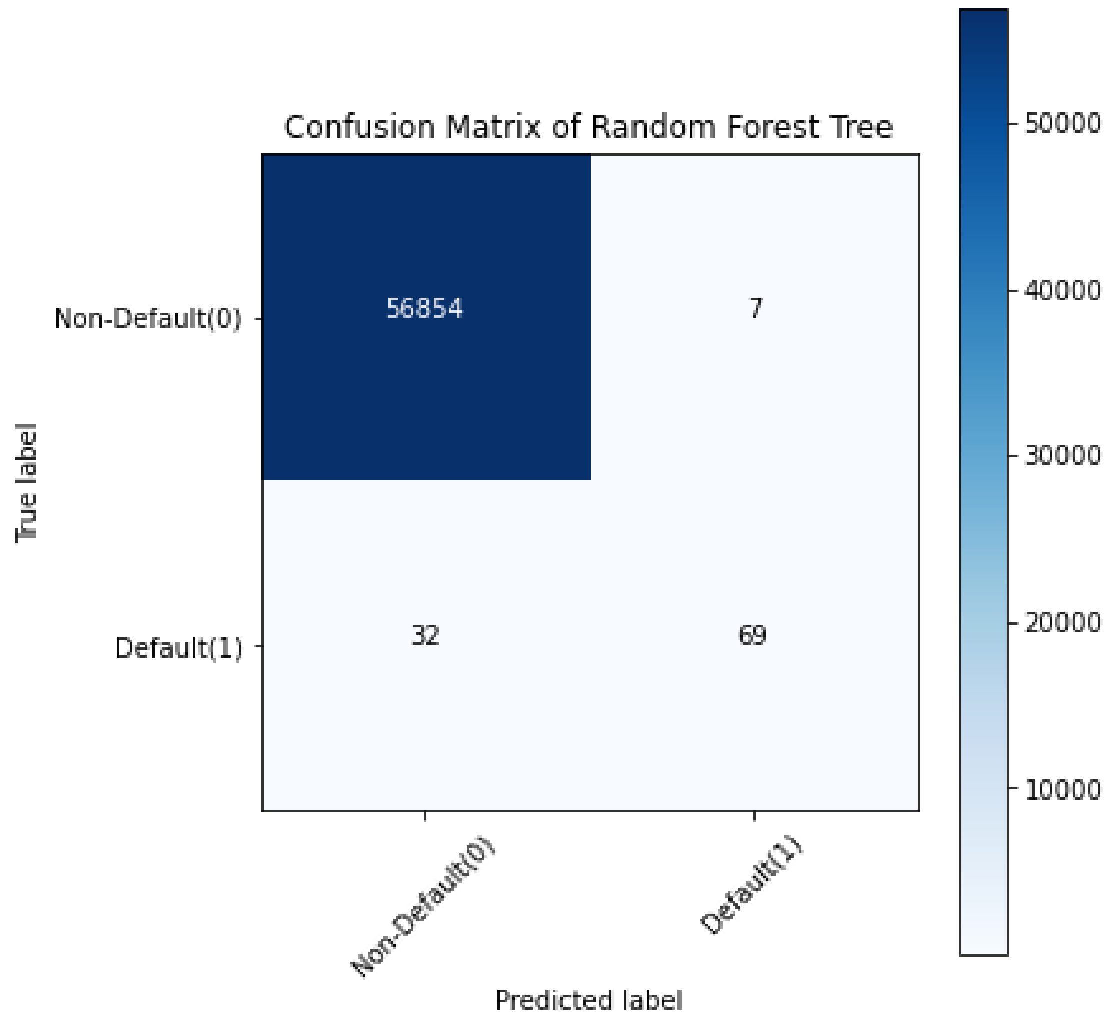


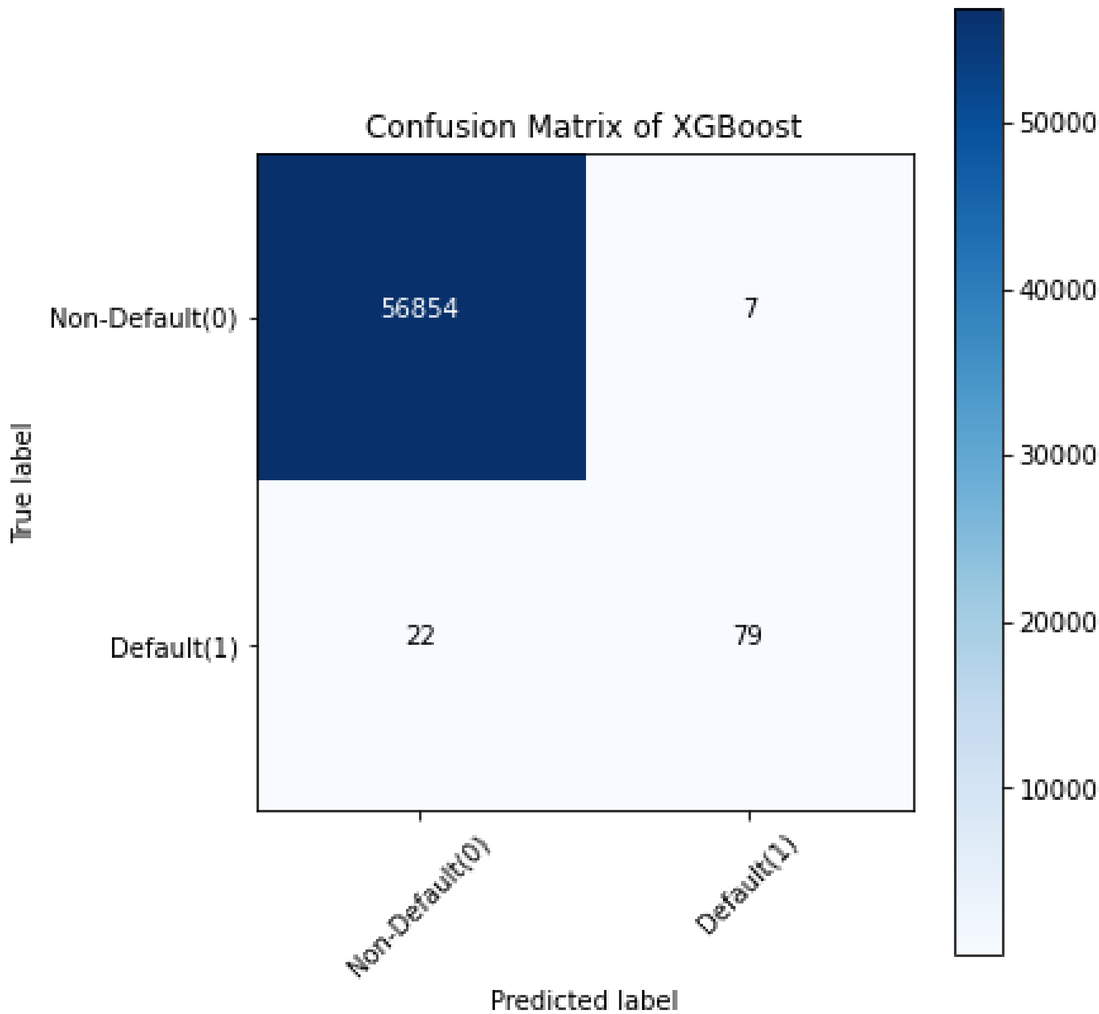
Confusion Matrix of Decision Tree
Non-Default(0)
Default(1)
True label
56849
12
24
77
Non-Default(0)
Default(1)
Predicted label
50000
40000
30000
20000
10000


Confusion Matrix of KNN
Non-Default(0)
Default(1)
True label
56854
7
31
70
Non-Default(0)
Default(1)
Predicted label
50000
40000
30000
20000
10000


Confusion Matrix of Logistic Regression
Non-Default(0)
Default(1)
True label
56853
8
40
61
Non-Default(0)
Default(1)
Predicted label
50000
40000
30000
20000
10000


Confusion Matrix of SVM
Non-Default(0)
Default(1)
True label
56854
7
65
36
Non-Default(0)
Default(1)
Predicted label
50000
40000
30000
20000
10000


Confusion Matrix of Random Forest Tree
Non-Default(0)
Default(1)
56854
7
32
69
True label
Predicted label
Non-Default(0)
Default(1)
50000
40000
30000
20000
10000


Confusion Matrix of XGBoost
Non-Default(0)
Default(1)
True label
56854
7
22
79
Non-Default(0)
Default(1)
Predicted label
50000
40000
30000
20000
10000