In this notebook, my goal is to handle unbalanced data by Under sampling the majority class, over sampling minority class, over sampling minority class using SMOTE. There are other techniques there but I am strict with those three.

Table of Content:

Importing the Dependencies

In [431]:
```
#!pip install -U imbalanced-learn
```

In [432]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from matplotlib import style
%matplotlib inline
import seaborn as sns
from imblearn.over_sampling import SMOTE
from sklearn import metrics
```

Read Data from csv

In [433]:

```
data = pd.read_csv('/kaggle/input/creditcardfraud/creditcard.csv')
data.head()
```

Out[433]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.

5 rows × 31 columns

In [434]:

```
data.tail()
```

Out[434]:

	Time	V1	V2	V3	V4	V5	V6	V7	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.

5 rows × 31 columns

Take a Look at the Data Structure

In [435]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

We can see that all data is in float64 without Class and that is in Int64.

Checking Missing Value

In [436]:
```
data.isnull().sum()
```

Out[436]:
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

There are no null values so we donot need to handle it

distribution of legit transactions & fraudulent transactions

In [437]:
```
data['Class'].value_counts()
```

Out[437]:
```
0    284315
1       492
Name: Class, dtype: int64
```

0 --> Normal Transaction

1 --> fraudulent transaction

In [438]:
```
# separating the data for analysis
legit = data[data.Class == 0]
fraud = data[data.Class == 1]
```

In [439]:
```
print(legit.shape)
print(fraud.shape)
```

```
(284315, 31)
(492, 31)
```

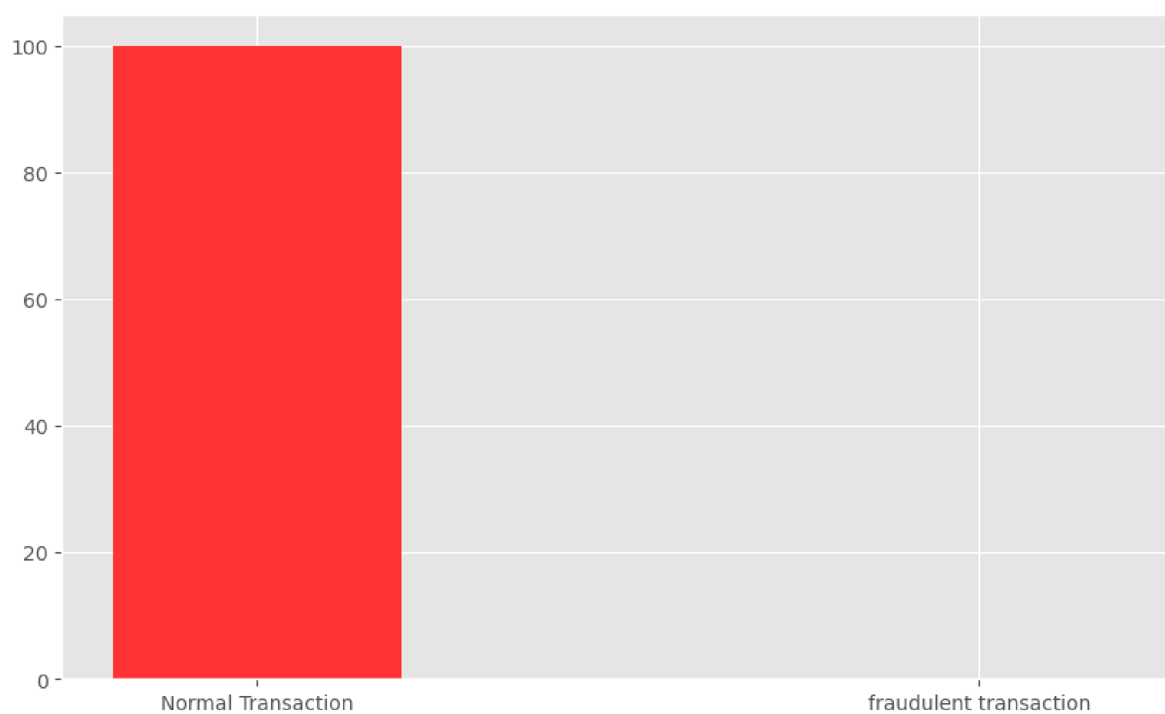
Legit and fraud data ratio

In [440]:
```
legit_fraud_percentance = []
legit_percentance = (len(legit)/len(data))*100
fraud_percentance = (len(fraud)/len(data))*100
legit_fraud_percentance.append(legit_percentance)
legit_fraud_percentance.append(fraud_percentance)
print('Normal Transaction in percentance:',legit_percentance)
print('Fraud Transaction in percentance:',fraud_percentance)
```

```
Normal Transaction in percentance: 99.82725143693798
Fraud Transaction in percentance: 0.1727485630620034
```

In [441]:
```
style.use('ggplot')
plt.figure(figsize=(10,6))
plt.bar(['Normal Transaction','fraudulent transaction'],legit_fraud_percentanc
```

Out[441]: <BarContainer object of 2 artists>



Note: this data is highly unbalance

In [442]:
```
# statistical measures of the data
legit.Amount.describe()
```

Out[442]:
```
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
```

In [443]:
```
fraud.Amount.describe()
```

Out[443]:
```
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
```

We can see that mean of legit and fraud data varies.

In [444]:
```
# compare the values for both transactions
data.groupby('Class').mean()
```

Out[444]:

	Time	V1	V2	V3	V4	V5	V6	V7
Class								
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731

2 rows × 30 columns

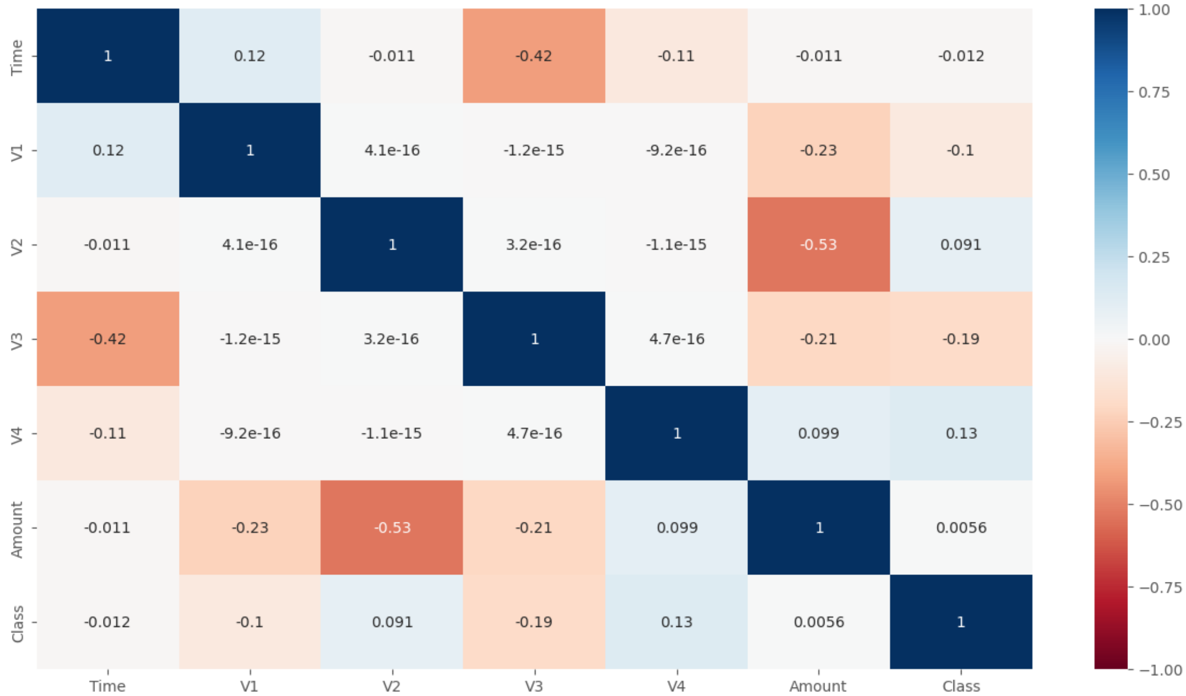
In [445]:
```
data.keys()
```

Out[445]:
```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

Corelation Between Some data

```
In [446]: corr = data[['Time', 'V1', 'V2', 'V3', 'V4','Amount','Class']].corr()
          plt.figure(figsize=(15,8))
          sns.heatmap(corr, cmap='RdBu', vmin=-1, vmax= 1 , annot=True)

Out[446]: <AxesSubplot:>
```



Creating Model to check all Unbalace data solving technic

```
In [447]: def model_create_and_get_score(X,Y):
              X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, s

              model = LogisticRegression(max_iter=10000)
              model.fit(X_train, Y_train)

              X_train_prediction = model.predict(X_train)
              training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
              print('Accuracy on Training data : ', training_data_accuracy)

              X_test_prediction = model.predict(X_test)
              test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
              print('Accuracy score on Test Data : ', test_data_accuracy)

              return Y_test,X_test_prediction
```

Confusion Matrix

In [448]:
```
def draw_confusion_matrix(y_test,test_predict):
    confusion_matrix = metrics.confusion_matrix(y_test, test_predict)

    cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_ma

    cm_display.plot()
    plt.show()
```

Under-Sampling majority class

Now I am going to apply Under-Sampling majority class. Build a sample dataset containing similar distribution of normal transactions and Fraudulent Transactions

In [449]:
```
legit_sample = legit.sample(n=len(fraud))
```

Concatenating two DataFrames

In [450]:
```
new_dataset = pd.concat([legit_sample, fraud], axis=0)
```

In [451]:
```
new_dataset.head()
```

Out[451]:

	Time	V1	V2	V3	V4	V5	V6	V7	
158416	111201.0	1.913813	-0.160000	-1.069179	0.644357	0.249109	-0.324692	0.085130	-0.3
103741	68780.0	0.384771	-1.093859	1.613258	3.046840	-1.518739	0.769120	-0.629265	0.3
235327	148341.0	1.703300	-0.863266	0.104905	1.247179	-0.141436	2.679679	-1.520703	1.0
151304	95300.0	-0.354216	1.193045	0.224017	-0.242087	-0.041323	-1.261796	0.604900	0.1
252173	155680.0	-1.645492	2.257946	-0.892133	-0.804311	0.321567	-1.090017	1.107703	-0.2

5 rows × 31 columns

In [452]:
```
new_dataset.tail()
```

Out[452]:

	Time	V1	V2	V3	V4	V5	V6	V7	
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.69
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.24
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.21
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.05
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.06

5 rows × 31 columns

In [453]:
```
new_dataset['Class'].value_counts()
```

Out[453]:
```
0    492
1    492
Name: Class, dtype: int64
```

In [454]:
```
new_dataset.groupby('Class').mean()
```

Out[454]:

	Time	V1	V2	V3	V4	V5	V6	V7	
Class									
0	92875.128049	0.094923	0.038078	0.075879	-0.123313	0.030663	-0.001247	0.010952	0
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0

2 rows × 30 columns

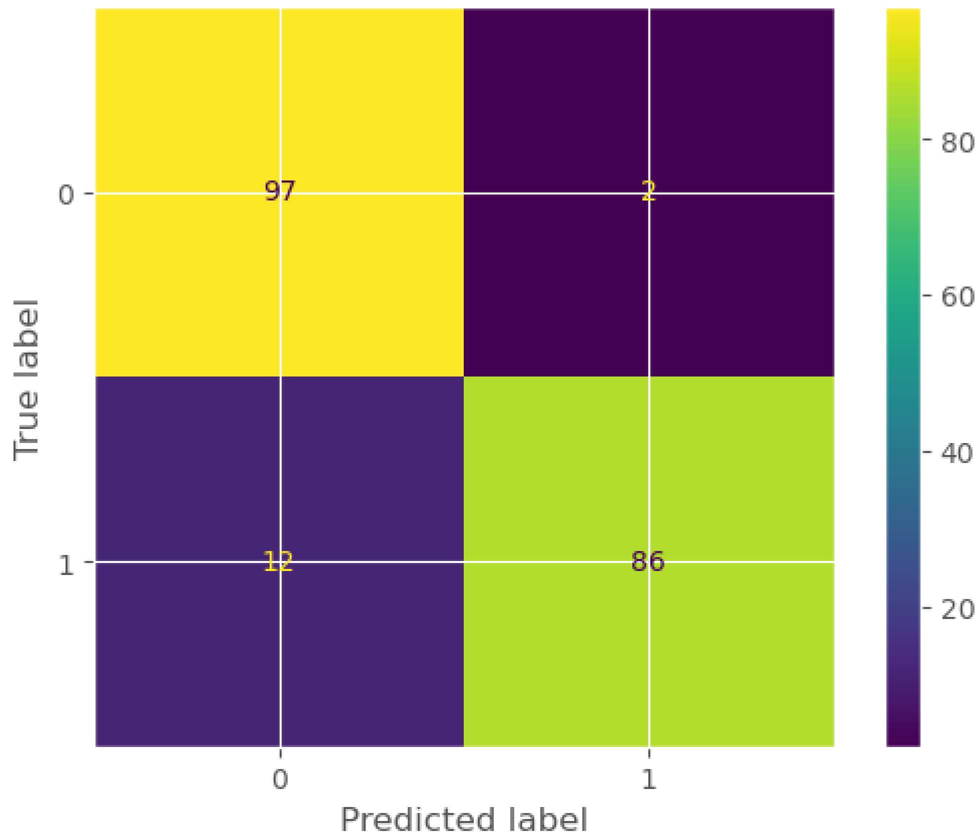
Splitting the data into Features & Targets

In [455]:
```
X = new_dataset.drop(columns='Class', axis=1)
Y = new_dataset['Class']
```

In [456]:
```
Y_test,X_test_prediction = model_create_and_get_score(X,Y)
```

```
Accuracy on Training data :  0.9440914866581956
Accuracy score on Test Data :  0.9289340101522843
```

In [457]:
```
draw_confusion_matrix(Y_test,X_test_prediction)
```



Over-Sampling minority class

In [458]:
```
legit = data[data.Class == 0]
fraud = data[data.Class == 1]
```

In [459]:
```
fraud_over = fraud.sample(len(legit), replace=True)
new_data = pd.concat([fraud_over, legit], axis=0)
```

In [460]:
```
new_data['Class'].value_counts()
```

Out[460]:
```
1    284315
0    284315
Name: Class, dtype: int64
```

In [461]:
```
new_data.groupby('Class').mean()
```

Out[461]:

	Time	V1	V2	V3	V4	V5	V6	V7
Class								
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637
1	80831.807305	-4.775089	3.623517	-7.041394	4.547975	-3.145516	-1.400052	-5.571603

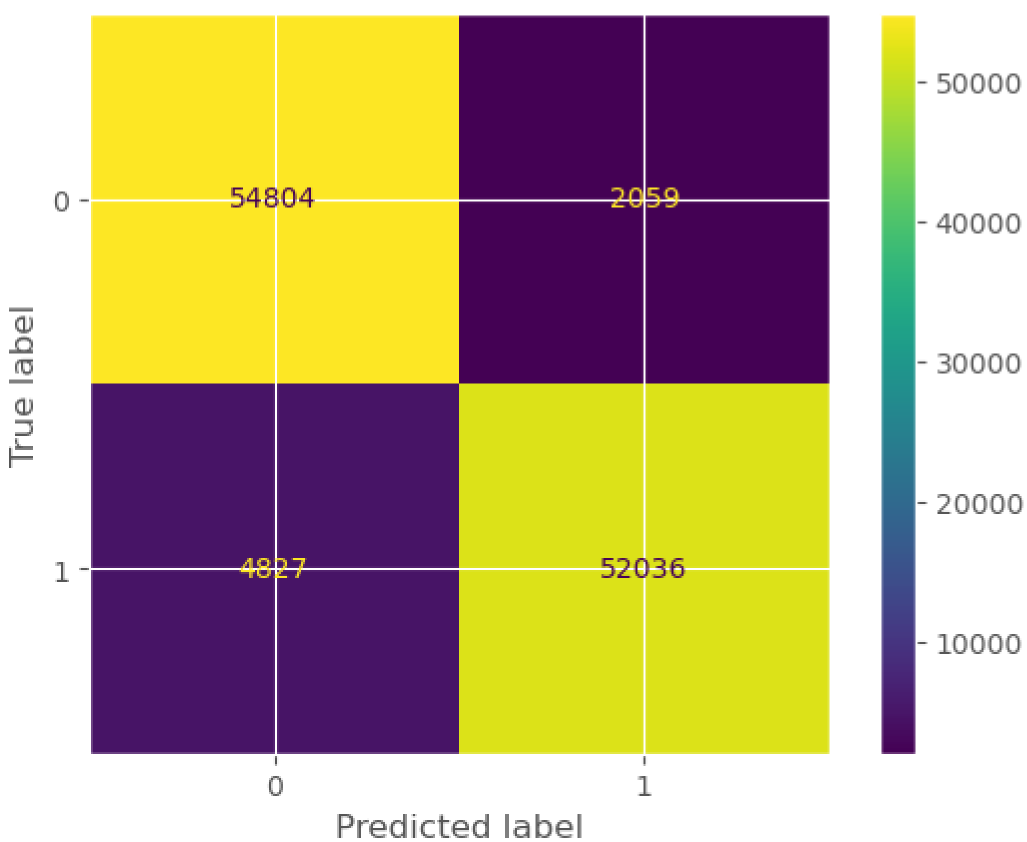
2 rows × 30 columns

In [462]:
```
X = new_data.drop(columns='Class', axis=1)
Y = new_data['Class']
```

In [463]:
```
Y_test,X_test_prediction = model_create_and_get_score(X,Y)
```

```
Accuracy on Training data :  0.93879807607759
Accuracy score on Test Data :  0.9394509610818986
```

In [464]:
```
draw_confusion_matrix(Y_test,X_test_prediction)
```



Over-Sampling minority class using SMOTE

In [465]:
```
X = data.drop(columns='Class', axis=1)
Y = data['Class']
```

In [466]:
```
Y.value_counts()
```

Out[466]:
```
0    284315
1       492
Name: Class, dtype: int64
```

In [467]:
```
smote = SMOTE(sampling_strategy='minority')
X_sm, y_sm = smote.fit_resample(X, Y)

y_sm.value_counts()
```

Out[467]:
```
0    284315
1    284315
Name: Class, dtype: int64
```

In [468]:
```
X_sm.describe()
```

Out[468]:

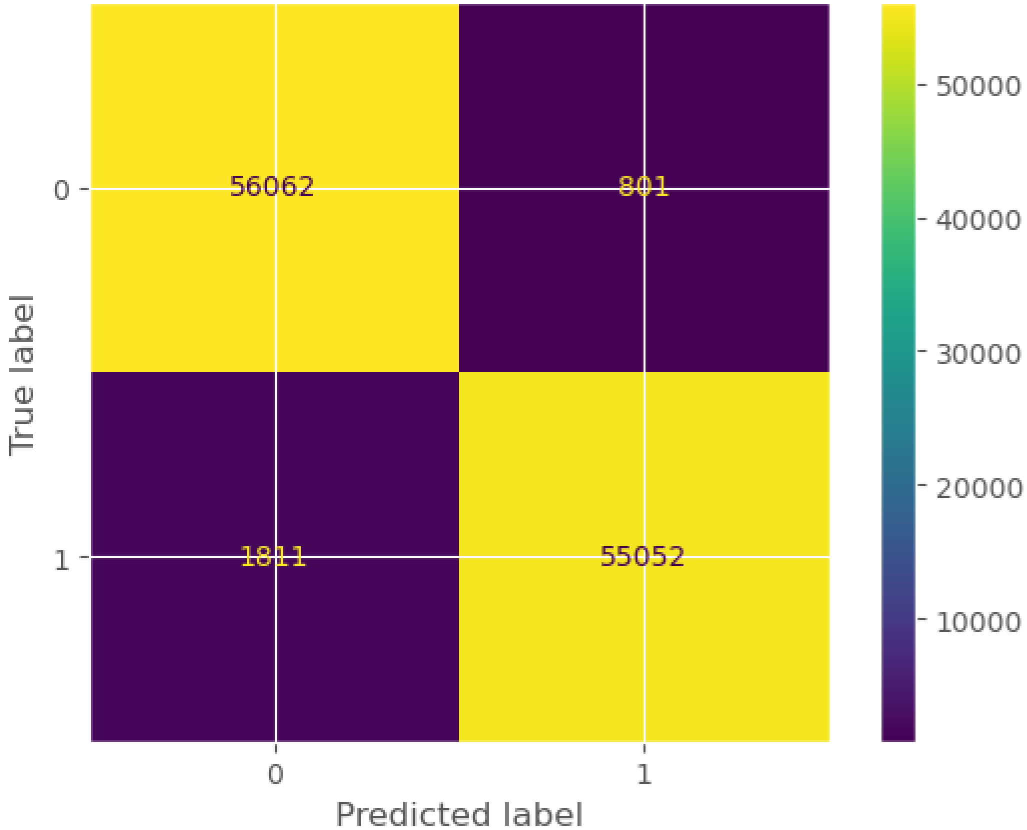
	Time	V1	V2	V3	V4	V
count	568630.000000	568630.000000	568630.000000	568630.000000	568630.000000	568630.00000
mean	87757.258132	-2.482433	1.913966	-3.652227	2.327014	-1.63253
std	48123.884137	5.456757	3.632952	6.161305	3.145203	4.13730
min	0.000000	-56.407510	-72.715728	-48.325589	-5.683171	-113.74330
25%	45928.463767	-3.072119	-0.098078	-5.237142	-0.055423	-1.82819
50%	80181.500000	-0.820666	1.015029	-1.549349	1.482337	-0.44452
75%	134649.750000	0.824701	2.892352	0.268893	4.340007	0.42783
max	172792.000000	2.454930	22.057729	9.382558	16.875344	34.80166

8 rows × 30 columns

In [469]:
```
Y_test,X_test_prediction = model_create_and_get_score(X_sm,y_sm)
```

```
Accuracy on Training data :  0.9756036438457345
Accuracy score on Test Data :  0.977032516750787
```

In [470]:
```
draw_confusion_matrix(Y_test,X_test_prediction)
```



In [ ]:
```
```