fraud-credit-card-detection

August 21, 2023

1 About Dataset

1.1 Context

It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

1.2 Content

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

Given the class imbalance ratio, we recommend measuring the accuracy using the Area Under the Precision-Recall Curve (AUPRC). Confusion matrix accuracy is not meaningful for unbalanced classification.

1.3 Source and link

Source: Kaggel Link: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

Author: Bao Thai

2 Import needed libraries and packages

[1]:
```
import numpy as np
import pandas as pd
!pip install matplotlib
import matplotlib.pyplot as plt
!pip install seaborn
```

```
import seaborn as sns
!pip install scipy
from scipy import stats
!pip install scikit-learn
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix, accuracy_score,␣
↪classification_report
from sklearn.metrics import r2_score,f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
!pip install xgboost
from xgboost import XGBClassifier
from csv import reader
```

```
[notice] A new release of pip available: 22.3.1 -> 23.2.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
Requirement already satisfied: matplotlib in c:\users\admin\lib\site-packages
(3.7.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\admin\lib\site-
packages (from matplotlib) (1.0.7)
Requirement already satisfied: cycler>=0.10 in c:\users\admin\lib\site-packages
(from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\admin\lib\site-
packages (from matplotlib) (4.39.3)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\admin\lib\site-
packages (from matplotlib) (1.4.4)
Requirement already satisfied: numpy>=1.20 in c:\users\admin\lib\site-packages
(from matplotlib) (1.24.2)
Requirement already satisfied: packaging>=20.0 in c:\users\admin\lib\site-
packages (from matplotlib) (23.0)
Requirement already satisfied: pillow>=6.2.0 in c:\users\admin\lib\site-packages
(from matplotlib) (9.5.0)
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in c:\users\admin\lib\site-
packages (from matplotlib) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\admin\lib\site-
packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in c:\users\admin\lib\site-packages
(from python-dateutil>=2.7->matplotlib) (1.16.0)
```

```
[notice] A new release of pip available: 22.3.1 -> 23.2.1
[notice] To update, run: python.exe -m pip install --upgrade pip

Requirement already satisfied: seaborn in c:\users\admin\lib\site-packages
(0.12.2)
Requirement already satisfied: numpy!=1.24.0,>=1.17 in c:\users\admin\lib\site-
packages (from seaborn) (1.24.2)
Requirement already satisfied: pandas>=0.25 in c:\users\admin\lib\site-packages
(from seaborn) (1.5.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in
c:\users\admin\lib\site-packages (from seaborn) (3.7.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\admin\lib\site-
packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.0.7)
Requirement already satisfied: cycler>=0.10 in c:\users\admin\lib\site-packages
(from matplotlib!=3.6.1,>=3.1->seaborn) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\admin\lib\site-
packages (from matplotlib!=3.6.1,>=3.1->seaborn) (4.39.3)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\admin\lib\site-
packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\admin\lib\site-
packages (from matplotlib!=3.6.1,>=3.1->seaborn) (23.0)
Requirement already satisfied: pillow>=6.2.0 in c:\users\admin\lib\site-packages
(from matplotlib!=3.6.1,>=3.1->seaborn) (9.5.0)
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in c:\users\admin\lib\site-
packages (from matplotlib!=3.6.1,>=3.1->seaborn) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\admin\lib\site-
packages (from matplotlib!=3.6.1,>=3.1->seaborn) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\admin\lib\site-packages
(from pandas>=0.25->seaborn) (2022.7.1)
Requirement already satisfied: six>=1.5 in c:\users\admin\lib\site-packages
(from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.1->seaborn) (1.16.0)
Requirement already satisfied: scipy in c:\users\admin\lib\site-packages
(1.11.1)


[notice] A new release of pip available: 22.3.1 -> 23.2.1
[notice] To update, run: python.exe -m pip install --upgrade pip


Requirement already satisfied: numpy<1.28.0,>=1.21.6 in c:\users\admin\lib\site-
packages (from scipy) (1.24.2)
Requirement already satisfied: scikit-learn in c:\users\admin\lib\site-packages
(1.3.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\admin\lib\site-packages
(from scikit-learn) (1.24.2)
Requirement already satisfied: scipy>=1.5.0 in c:\users\admin\lib\site-packages
(from scikit-learn) (1.11.1)
Requirement already satisfied: joblib>=1.1.1 in c:\users\admin\lib\site-packages
```

```
(from scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\admin\lib\site-
packages (from scikit-learn) (3.2.0)


[notice] A new release of pip available: 22.3.1 -> 23.2.1
[notice] To update, run: python.exe -m pip install --upgrade pip
Requirement already satisfied: xgboost in c:\users\admin\lib\site-packages
(1.7.6)
Requirement already satisfied: numpy in c:\users\admin\lib\site-packages (from
xgboost) (1.24.2)
Requirement already satisfied: scipy in c:\users\admin\lib\site-packages (from
xgboost) (1.11.1)


[notice] A new release of pip available: 22.3.1 -> 23.2.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

2.1 Loading Dataset

[2]:
```python
data_original = pd.read_csv('C:/Users/Admin/Desktop/Projects/creditcard.csv')
df = data_original.copy()
df.head(20)
```

[2]:
```
    Time        V1        V2        V3        V4        V5        V6  \
0    0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388
1    0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361
2    1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499
3    1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203
4    2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921
5    2.0 -0.425966  0.960523  1.141109 -0.168252  0.420987 -0.029728
6    4.0  1.229658  0.141004  0.045371  1.202613  0.191881  0.272708
7    7.0 -0.644269  1.417964  1.074380 -0.492199  0.948934  0.428118
8    7.0 -0.894286  0.286157 -0.113192 -0.271526  2.669599  3.721818
9    9.0 -0.338262  1.119593  1.044367 -0.222187  0.499361 -0.246761
10  10.0  1.449044 -1.176339  0.913860 -1.375667 -1.971383 -0.629152
11  10.0  0.384978  0.616109 -0.874300 -0.094019  2.924584  3.317027
12  10.0  1.249999 -1.221637  0.383930 -1.234899 -1.485419 -0.753230
13  11.0  1.069374  0.287722  0.828613  2.712520 -0.178398  0.337544
14  12.0 -2.791855 -0.327771  1.641750  1.767473 -0.136588  0.807596
15  12.0 -0.752417  0.345485  2.057323 -1.468643 -1.158394 -0.077850
16  12.0  1.103215 -0.040296  1.267332  1.289091 -0.735997  0.288069
17  13.0 -0.436905  0.918966  0.924591 -0.727219  0.915679 -0.127867
18  14.0 -5.401258 -5.450148  1.186305  1.736239  3.049106 -1.763406
19  15.0  1.492936 -1.029346  0.454795 -1.438026 -1.555434 -0.720961

          V7        V8        V9  …       V21       V22       V23       V24  \
0   0.239599  0.098698  0.363787  … -0.018307  0.277838 -0.110474  0.066928
```

```
1  -0.078803  0.085102 -0.255425  … -0.225775 -0.638672  0.101288 -0.339846
2   0.791461  0.247676 -1.514654  …  0.247998  0.771679  0.909412 -0.689281
3   0.237609  0.377436 -1.387024  … -0.108300  0.005274 -0.190321 -1.175575
4   0.592941 -0.270533  0.817739  … -0.009431  0.798278 -0.137458  0.141267
5   0.476201  0.260314 -0.568671  … -0.208254 -0.559825 -0.026398 -0.371427
6  -0.005159  0.081213  0.464960  … -0.167716 -0.270710 -0.154104 -0.780055
7   1.120631 -3.807864  0.615375  …  1.943465 -1.015455  0.057504 -0.649709
8   0.370145  0.851084 -0.392048  … -0.073425 -0.268092 -0.204233  1.011592
9   0.651583  0.069539 -0.736727  … -0.246914 -0.633753 -0.120794 -0.385050
10 -1.423236  0.048456 -1.720408  … -0.009302  0.313894  0.027740  0.500512
11  0.470455  0.538247 -0.558895  …  0.049924  0.238422  0.009130  0.996710
12 -0.689405 -0.227487 -2.094011  … -0.231809 -0.483285  0.084668  0.392831
13 -0.096717  0.115982 -0.221083  … -0.036876  0.074412 -0.071407  0.104744
14 -0.422911 -1.907107  0.755713  …  1.151663  0.222182  1.020586  0.028317
15 -0.608581  0.003603 -0.436167  …  0.499625  1.353650 -0.256573 -0.065084
16 -0.586057  0.189380  0.782333  … -0.024612  0.196002  0.013802  0.103758
17  0.707642  0.087962 -0.665271  … -0.194796 -0.672638 -0.156858 -0.888386
18 -1.559738  0.160842  1.233090  … -0.503600  0.984460  2.458589  0.042119
19 -1.080664 -0.053127 -1.978682  … -0.177650 -0.175074  0.040002  0.295814

         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  -0.232794  0.105915  0.253844  0.081080    3.67      0
6   0.750137 -0.257237  0.034507  0.005168    4.99      0
7  -0.415267 -0.051634 -1.206921 -1.085339   40.80      0
8   0.373205 -0.384157  0.011747  0.142404   93.20      0
9  -0.069733  0.094199  0.246219  0.083076    3.68      0
10  0.251367 -0.129478  0.042850  0.016253    7.80      0
11 -0.767315 -0.492208  0.042472 -0.054337    9.99      0
12  0.161135 -0.354990  0.026416  0.042422  121.50      0
13  0.548265  0.104094  0.021491  0.021293   27.50      0
14 -0.232746 -0.235557 -0.164778 -0.030154   58.80      0
15 -0.039124 -0.087086 -0.180998  0.129394   15.99      0
16  0.364298 -0.382261  0.092809  0.037051   12.99      0
17 -0.342413 -0.049027  0.079692  0.131024    0.89      0
18 -0.481631 -0.621272  0.392053  0.949594   46.80      0
19  0.332931 -0.220385  0.022298  0.007602    5.00      0

[20 rows x 31 columns]
```

2.2 Explore Data

[53]:
```
df.shape
```

[53]:
```
(284807, 31)
```

[54]:
```
df.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
```

[55]:
```
df.describe()
```

[55]:

	Time	V1	V2	V3	V4	\
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	

	V5	V6	V7	V8	V9	\
count	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	
mean	9.604066e-16	1.487313e-15	-5.556467e-16	1.213481e-16	-2.406331e-15	
std	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+00	
min	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+01	
25%	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01	
50%	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-02	
75%	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-01	
max	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+01	

	…	V21	V22	V23	V24	\
count	…	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	
mean	…	1.654067e-16	-3.568593e-16	2.578648e-16	4.473266e-15	
std	…	7.345240e-01	7.257016e-01	6.244603e-01	6.056471e-01	
min	…	-3.483038e+01	-1.093314e+01	-4.480774e+01	-2.836627e+00	
25%	…	-2.283949e-01	-5.423504e-01	-1.618463e-01	-3.545861e-01	
50%	…	-2.945017e-02	6.781943e-03	-1.119293e-02	4.097606e-02	
75%	…	1.863772e-01	5.285536e-01	1.476421e-01	4.395266e-01	
max	…	2.720284e+01	1.050309e+01	2.252841e+01	4.584549e+00	

	V25	V26	V27	V28	Amount	\
count	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	284807.000000	
mean	5.340915e-16	1.683437e-15	-3.660091e-16	-1.227390e-16	88.349619	
std	5.212781e-01	4.822270e-01	4.036325e-01	3.300833e-01	250.120109	
min	-1.029540e+01	-2.604551e+00	-2.256568e+01	-1.543008e+01	0.000000	
25%	-3.171451e-01	-3.269839e-01	-7.083953e-02	-5.295979e-02	5.600000	
50%	1.659350e-02	-5.213911e-02	1.342146e-03	1.124383e-02	22.000000	
75%	3.507156e-01	2.409522e-01	9.104512e-02	7.827995e-02	77.165000	
max	7.519589e+00	3.517346e+00	3.161220e+01	3.384781e+01	25691.160000	

	Class
count	284807.000000
mean	0.001727
std	0.041527
min	0.000000
25%	0.000000
50%	0.000000

```
75%          0.000000
max          1.000000

[8 rows x 31 columns]
```

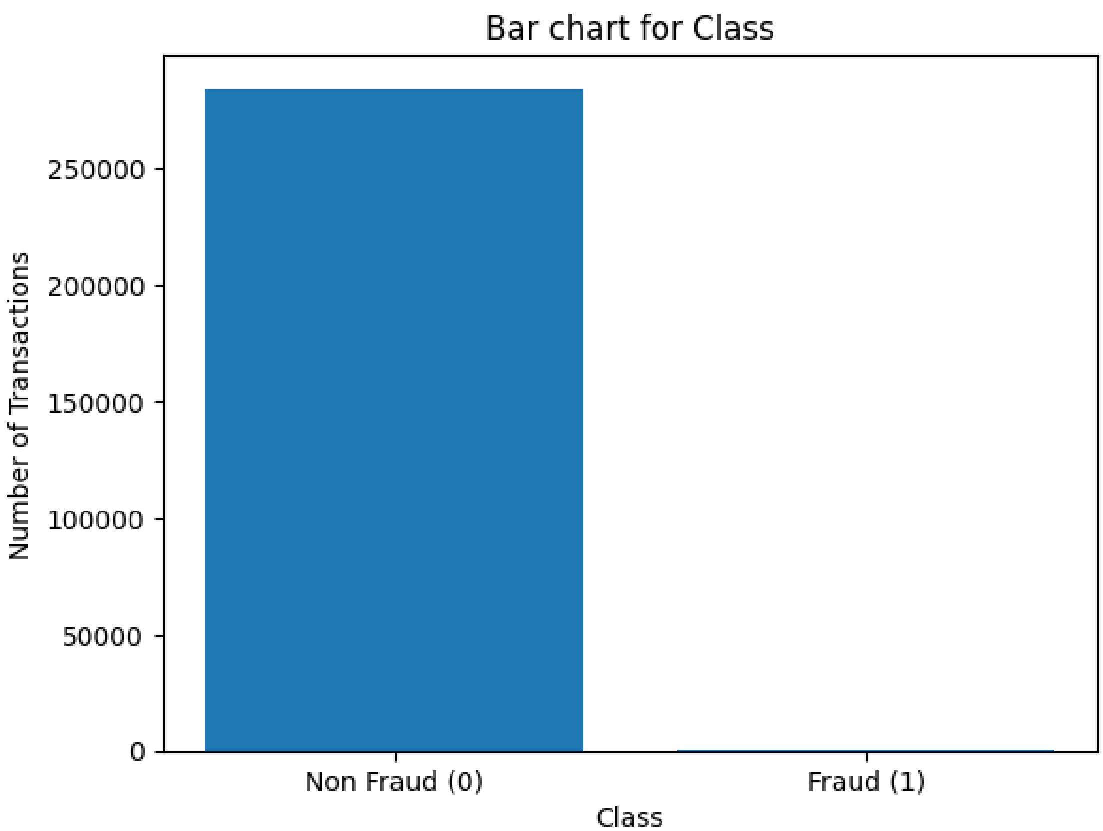
[56]:
```
df.isna().sum()
```

[56]:
```
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
```

[4]:
```
Class_count = df['Class'].value_counts()
Class_count
```

[4]:
```
0    284315
1       492
Name: Class, dtype: int64
```

[5]:
```
plt.bar(Class_count.index, Class_count)
plt.title ('Bar chart for Class')
plt.xlabel('Class')
plt.ylabel('Number of Transactions')
plt.xticks([0,1],['Non Fraud (0)','Fraud (1)'])
plt.show()
```



This bar chart illustrates the target column named 'Class' with a significant imbalance between two values: 0 and 1, Non Fraud and Frau respectively. I will address this issue later

2.3 Preprocessing and Visualizing

[3]:
```
df['Time'] = pd.to_datetime(df['Time'], unit = 'h')
df['Time'] = df['Time'].dt.hour
df.tail()
```

[3]:
```
        Time         V1         V2        V3        V4        V5        V6  \
284802    10 -11.881118  10.071785 -9.834783 -2.066656 -5.364473 -2.606837
284803    11  -0.732789  -0.055080  2.035030 -0.738589  0.868229  1.058415
284804    12   1.919565  -0.301254 -3.249640 -0.557828  2.630515  3.031260
```

```
284805    12  -0.240440   0.530483  0.702510  0.689799 -0.377961  0.623708
284806    16  -0.533413  -0.189733  0.703337 -0.506271 -0.012546 -0.649617

              V7        V8        V9  …       V21       V22       V23  \
284802 -4.918215  7.305334  1.914428  …  0.213454  0.111864  1.014480
284803  0.024330  0.294869  0.584800  …  0.214205  0.924384  0.012463
284804 -0.296827  0.708417  0.432454  …  0.232045  0.578229 -0.037501
284805 -0.686180  0.679145  0.392087  …  0.265245  0.800049 -0.163298
284806  1.577006 -0.414650  0.486180  …  0.261057  0.643078  0.376777

             V24       V25       V26       V27       V28  Amount  Class
284802 -0.509348  1.436807  0.250034  0.943651  0.823731    0.77      0
284803 -1.016226 -0.606624 -0.395255  0.068472 -0.053527   24.79      0
284804  0.640134  0.265745 -0.087371  0.004455 -0.026561   67.88      0
284805  0.123205 -0.569159  0.546668  0.108821  0.104533   10.00      0
284806  0.008797 -0.473649 -0.818267 -0.002415  0.013649  217.00      0

[5 rows x 31 columns]
```

[4]:
```python
Count_0 = df[df['Class']==0].groupby('Time').count()['Class']
Count_1 = df[df['Class']==1].groupby('Time').count()['Class']
df_counts = pd.concat([Count_0, Count_1], axis = 1, keys = ['Class 0', 'Class 1'])
df_counts = df_counts.reset_index()
df_counts
```

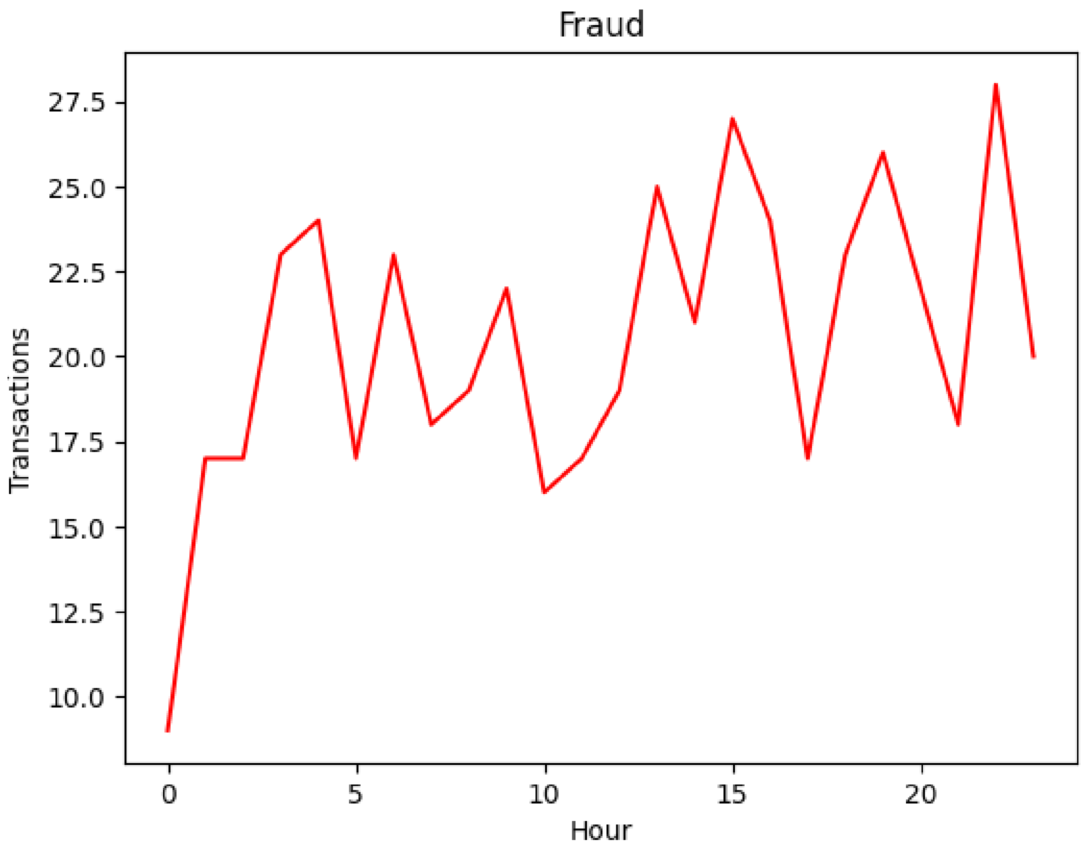
[4]:

	Time	Class 0	Class 1
0	0	11655	9
1	1	11650	17
2	2	11799	17
3	3	11806	23
4	4	11960	24
5	5	11887	17
6	6	11870	23
7	7	11709	18
8	8	11806	19
9	9	11944	22
10	10	11865	16
11	11	11640	17
12	12	12015	19
13	13	11985	25
14	14	11893	21
15	15	11817	27
16	16	11779	24
17	17	11746	17
18	18	11723	23
19	19	11999	26

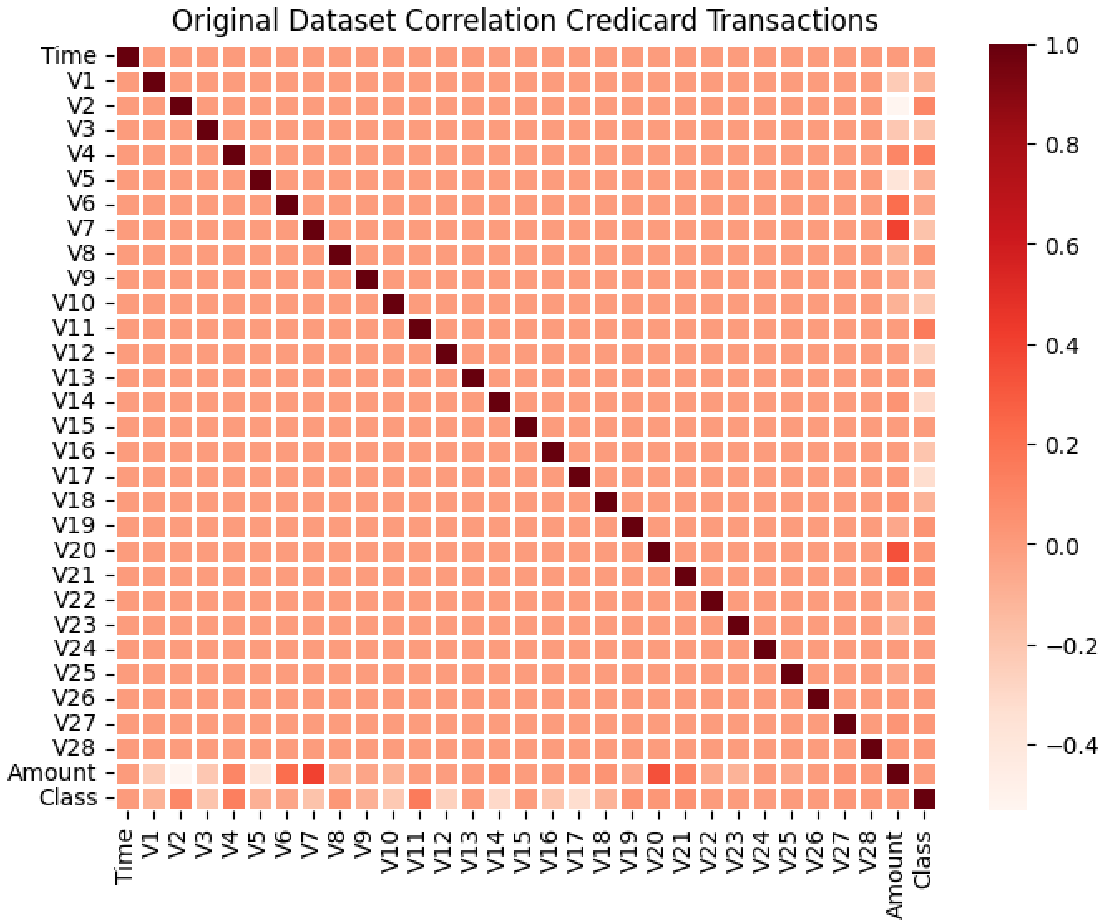
20	20	11908	22
21	21	12062	18
22	22	11911	28
23	23	11886	20

[5]:
```
plt.plot(df_counts['Time'], df_counts['Class 0'])
plt.title('Non Fraud')
plt.xlabel('Hour')
plt.ylabel('Transactions')
plt.show()
```

[27]:
```
plt.plot(df_counts['Time'],df_counts['Class 1'], color = 'red')
plt.title('Fraud')
plt.xlabel('Hour')
plt.ylabel('Transactions')
plt.show()
```



[17]:
```
plt.figure(figsize = (8,6))
plt.title('Original Dataset Correlation Credicard Transactions')
sns.heatmap(df.corr(), linewidths = 1 ,cmap = 'Reds')
plt.show()
```



2.4 Resampling Data

[10]:
```
Class_0 = df[df['Class'] == 0]
sample_size = 492
sample_Class_0 = Class_0.sample(sample_size, random_state = 0)
print(sample_Class_0.value_counts(sample_Class_0['Class']))
print(sample_Class_0)
```

```
Class
0    492
dtype: int64
        Time        V1        V2        V3        V4        V5        V6  \
266085    15  2.049094  0.186189 -1.707198  0.530768  0.160589 -1.448570
172120     7  2.125540 -0.030714 -1.527653  0.121046  0.543172 -0.347988
15136     12 -4.155859 -5.705748  0.274699 -0.993262 -6.059393  5.210848
96393     15 -0.566420 -0.579576  0.823503 -1.451240 -0.583587  0.206381
208225     9  0.060858 -0.261762 -1.699493 -1.202327  3.699527  3.196249
…       …         …         …         …         …         …         …
```

```
170744     20 -0.995867  1.201192  0.395282 -0.886059  1.706389 -0.367065
222921     19 -1.008598 -0.075940  2.004425 -0.505601  0.509548  0.171588
188275      3 -1.773275 -1.718423  1.455868 -2.174910  1.570504 -0.734998
44186       2 -0.524514  1.353868  0.217128  1.241104 -0.045413 -0.880130
147585      8 -1.731798  1.418551  0.565101  1.196712  0.365576 -0.299215

              V7        V8        V9  …       V21       V22       V23  \
266085  0.239310 -0.353611  0.634425  …  0.197782  0.741141 -0.009744
172120  0.157221 -0.229126  0.477999  … -0.336497 -0.838932  0.275173
15136   5.811316  0.367888  1.750710  …  1.371671  1.195815  4.188762
96393   1.601392 -0.370446 -1.910354  … -0.065082 -0.761357  0.641524
208225  0.437208  0.421541  0.492435  …  0.008303  0.534602  0.089602
…            …         …         … …       …         …
170744  1.089408 -0.210834 -0.608644  … -0.379122 -1.111202 -0.469886
222921  0.411154  0.066247 -0.024477  …  0.295670  0.738921 -0.229421
188275 -0.974958  0.386686 -1.026933  …  0.360589  0.305160 -0.122747
44186   0.197110  0.432705 -0.448191  …  0.133762  0.252190  0.073953
147585  1.092355 -0.242769 -0.089824  … -0.185723 -0.035647 -0.702781

             V24       V25       V26       V27       V28   Amount  Class
266085 -0.085057  0.228384 -0.097292 -0.001028 -0.032390     2.99      0
172120  0.049145 -0.156765  0.205919 -0.072321 -0.059009     1.98      0
15136  -1.091077  1.033044  0.224493 -0.486741  0.194275  1937.66      0
96393  -0.568974 -0.053164 -0.690995 -0.228630 -0.157254   320.05      0
208225  0.667918  0.017798  0.611584 -0.469946 -0.514370    11.50      0
…            …         …         …         …         …       …
170744 -0.016865  0.621809  0.303837 -0.258417  0.056865     1.29      0
222921  0.770200  0.591492 -0.124258 -0.142162 -0.142230    82.86      0
188275  0.175912  0.810150 -0.155587 -0.073359  0.068604    34.70      0
44186   0.240549 -0.303189 -0.321691 -0.144621 -0.001791     1.00      0
147585 -0.237518  0.556394 -0.449458 -0.493690 -0.354016   108.00      0

[492 rows x 31 columns]
```

I created a new DataFrame for values with Class 0. Then I randomly choosed sample 492 elements, compared with Class 1 also 492 elements.Next, I will merge the new DataFrame with Class 0 and the DataFrame with Class 1 to create a balanced DataFrame. Let's check!

[11]:
```
Class_1 = df[df['Class'] == 1]
df_balanced = pd.concat([Class_1, sample_Class_0], axis = 0, ignore_index =␣
 ↪True)
df_balanced.head(5)
```

[11]:
```
   Time        V1        V2        V3        V4        V5        V6        V7  \
0    22 -2.312227  1.951992 -1.609851  3.997906 -0.522188 -1.426545 -2.537387
1    16 -3.043541 -3.157307  1.088463  2.288644  1.359805 -1.064823  0.325574
2    22 -2.303350  1.759247 -0.359745  2.330243 -0.821628 -0.075788  0.562320
3     2 -4.397974  1.358367 -2.592844  2.679787 -1.128131 -1.706536 -3.496197
```

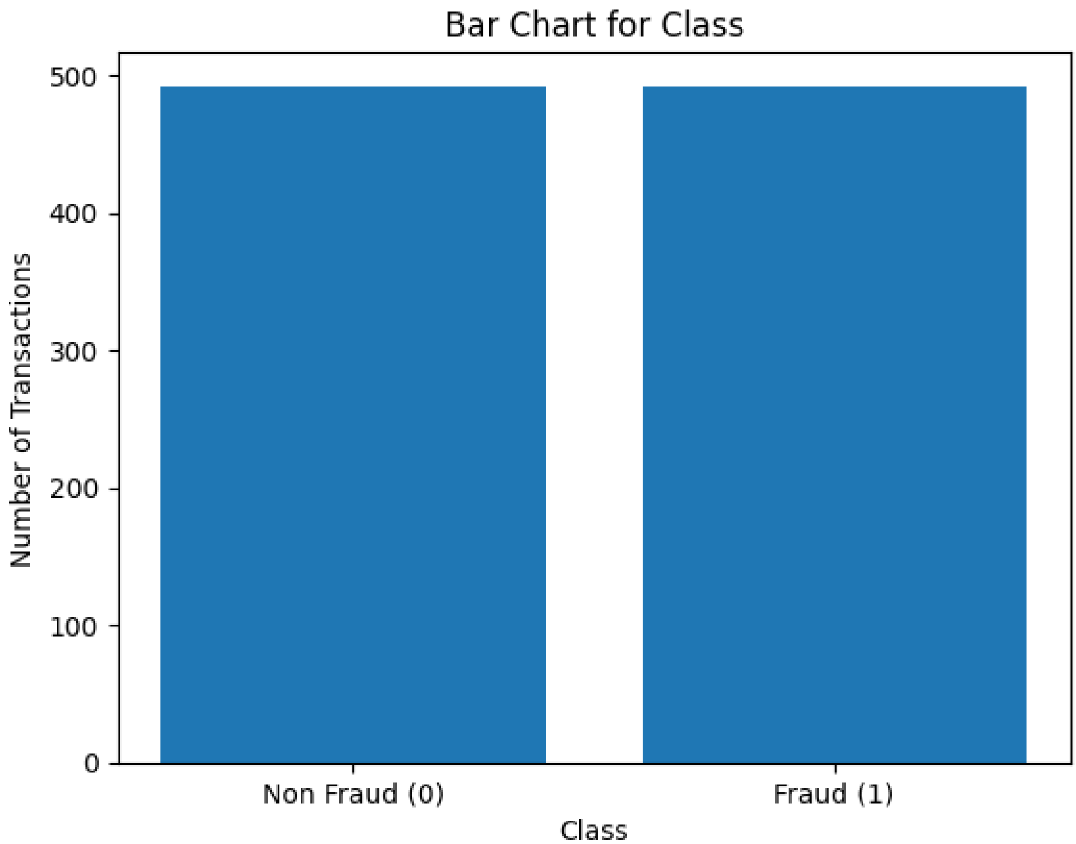
```
4     7  1.234235  3.019740 -4.304597  4.732795  3.624201 -1.357746  1.713445

         V8        V9  …       V21       V22       V23       V24       V25  \
0  1.391657 -2.770089  …  0.517232 -0.035049 -0.465211  0.320198  0.044519
1 -0.067794 -0.270953  …  0.661696  0.435477  1.375966 -0.293803  0.279798
2 -0.399147 -0.238253  … -0.294166 -0.932391  0.172726 -0.087330 -0.156114
3 -0.248778 -0.247768  …  0.573574  0.176968 -0.436207 -0.053502  0.252405
4 -0.496358 -1.282858  … -0.379068 -0.704181 -0.656805 -1.632653  1.488901

        V26       V27       V28  Amount  Class
0  0.177840  0.261145 -0.143276    0.00      1
1 -0.145362 -0.252773  0.035764  529.00      1
2 -0.542628  0.039566 -0.153029  239.93      1
3 -0.657488 -0.827136  0.849573   59.00      1
4  0.566797 -0.010016  0.146793    1.00      1

[5 rows x 31 columns]
```

[12]:
```python
new_class = df_balanced['Class'].value_counts()
new_class
```

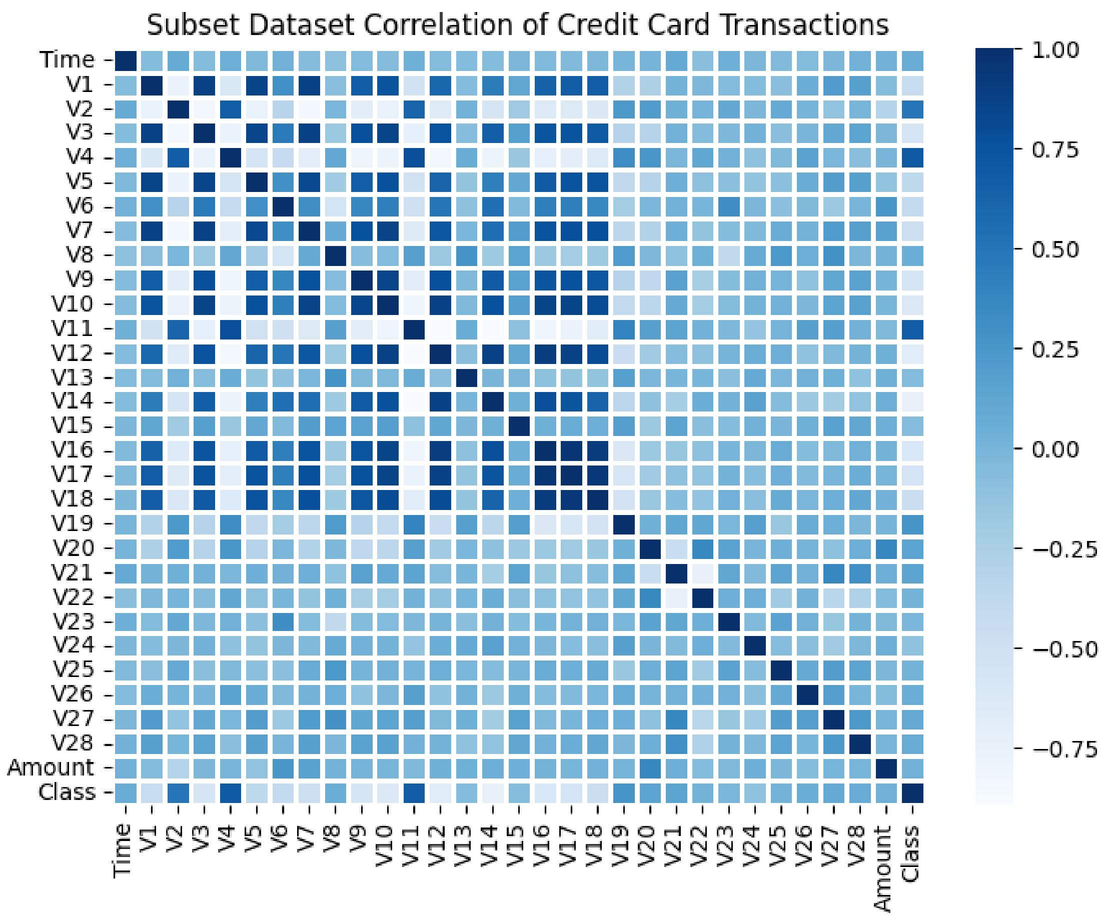
[12]:
```
1    492
0    492
Name: Class, dtype: int64
```

I successfully address the imbalanced data. Now, the result show that both values 0 and 1 have 492 instances. To make it easier to visualize, I will put them on a bar chart below. Let's check!

[13]:
```python
plt.bar(new_class.index, new_class)
plt.title('Bar Chart for Class')
plt.xlabel ('Class')
plt.ylabel('Number of Transactions')
plt.xticks([0, 1], ['Non Fraud (0)', 'Fraud (1)'])
plt.show()
```



[14]:
```
plt.figure(figsize = (8,6))
plt.title('Subset Dataset Correlation of Credit Card Transactions')
sns.heatmap(df_balanced.corr(),linewidths = 1, cmap = 'Blues')
plt.show()
```



2.5 Data Preprocessing On Subset Dataset

In this part, I split the dataset into training and testing sets. X is features for classification variables to the target column y. It means, y contain 'Class' within there are 0 and 1. And the models will use the rest columns to classificate whether a variable is in 0 or is in 1.

[28]:
```
X = df_balanced.drop(['Class'], axis = 1)
y = df_balanced['Class']
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.
↪2,random_state = 10)
```

Next, I do standardizate data

[29]:
```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

2.6 Building Models

In this par, I will build 8 models machine learning to solve this case.

[3]:
```
lr = LogisticRegression()
dtc = DecisionTreeClassifier()
rfc = RandomForestClassifier()
gbc = GradientBoostingClassifier()
knn = KNeighborsClassifier()
svc = SVC()
gnb = GaussianNB()
xgb = XGBClassifier()
```

[37]:
```
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
Classification_rp = classification_report(y_test, y_pred)
print(Classification_rp)
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.96      0.90        95
           1       0.96      0.83      0.89       102

    accuracy                           0.89       197
   macro avg       0.90      0.90      0.89       197
weighted avg       0.90      0.89      0.89       197

[[91  4]
 [17 85]]
```

[39]:
```
dtc.fit(X_train, y_train)
y_pred = dtc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
Classification_rp = classification_report(y_test, y_pred)
print(Classification_rp)
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.93      0.90        95
           1       0.93      0.87      0.90       102

    accuracy                           0.90       197
   macro avg       0.90      0.90      0.90       197
weighted avg       0.90      0.90      0.90       197
```

```
[[88  7]
 [13 89]]
```

[40]:
```
rfc.fit(X_train, y_train)
y_pred = rfc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
Classification_rp = classification_report(y_test, y_pred)
print(Classification_rp)
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.99      0.92        95
           1       0.99      0.84      0.91       102

    accuracy                           0.91       197
   macro avg       0.92      0.92      0.91       197
weighted avg       0.92      0.91      0.91       197

[[94  1]
 [16 86]]
```

[41]:
```
gbc.fit(X_train, y_train)
y_pred = gbc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
Classification_rp = classification_report(y_test, y_pred)
print(Classification_rp)
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.98      0.91        95
           1       0.98      0.84      0.91       102

    accuracy                           0.91       197
   macro avg       0.92      0.91      0.91       197
weighted avg       0.92      0.91      0.91       197

[[93  2]
 [16 86]]
```

[42]:
```
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
Classification_rp = classification_report(y_test, y_pred)
print(Classification_rp)
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.96      0.91        95
           1       0.96      0.85      0.90       102

    accuracy                           0.90       197
   macro avg       0.91      0.91      0.90       197
weighted avg       0.91      0.90      0.90       197

[[91  4]
 [15 87]]
```

[43]:
```
svc.fit(X_train, y_train)
y_pred = svc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
Classification_rp = classification_report(y_test, y_pred)
print(Classification_rp)
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.96      0.90        95
           1       0.96      0.84      0.90       102

    accuracy                           0.90       197
   macro avg       0.90      0.90      0.90       197
weighted avg       0.90      0.90      0.90       197

[[91  4]
 [16 86]]
```

[44]:
```
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
Classification_rp = classification_report(y_test, y_pred)
print(Classification_rp)
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.97      0.90        95
           1       0.97      0.83      0.89       102

    accuracy                           0.90       197
   macro avg       0.90      0.90      0.90       197
weighted avg       0.91      0.90      0.90       197

[[92  3]
```

```
 [17 85]]
```

[45]:
```
xgb.fit(X_train, y_train)
y_pred = xgb.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
Classification_rp = classification_report(y_test, y_pred)
print(Classification_rp)
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.96      0.91        95
           1       0.96      0.87      0.91       102

    accuracy                           0.91       197
   macro avg       0.92      0.92      0.91       197
weighted avg       0.92      0.91      0.91       197

[[91  4]
 [13 89]]
```

The results indicate that the models perform quite well on this data subset. To avoid bias opinions, I have selected more 3 models with lower accuracy (Logistic Regression, SVC, Decision Tree Classifier) to compare with the top 3 models (Random Forest Classifier,Gradient Boosting Classifier, XGB Classifier) with the highest accuracy score of 91% to re-run on the original dataset in order to assess and compare the effectiveness of these 6 models.

2.7 Re-Test On Original Dataset

[4]:
```
X = df.drop(['Class'], axis = 1)
y = df['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
↪random_state = 11)
```

[5]:
```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

These are 3 top result models on data subset:

[6]:
```
rfc.fit(X_train, y_train)
y_pred = rfc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
Classification_rp = classification_report(y_test, y_pred)
print(Classification_rp)
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support
```

```
           0       1.00      1.00      1.00     56854
           1       0.98      0.77      0.86       108

    accuracy                           1.00     56962
   macro avg       0.99      0.88      0.93     56962
weighted avg       1.00      1.00      1.00     56962

[[56852     2]
 [   25    83]]
```

[7]:

```
gbc.fit(X_train, y_train)
y_pred = gbc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
Classification_rp = classification_report(y_test, y_pred)
print(Classification_rp)
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56854
           1       0.77      0.80      0.79       108

    accuracy                           1.00     56962
   macro avg       0.89      0.90      0.89     56962
weighted avg       1.00      1.00      1.00     56962

[[56829    25]
 [   22    86]]
```

[8]:

```
xgb.fit(X_train, y_train)
y_pred = xgb.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
Classification_rp = classification_report(y_test, y_pred)
print(Classification_rp)
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56854
           1       0.98      0.79      0.87       108

    accuracy                           1.00     56962
   macro avg       0.99      0.89      0.94     56962
weighted avg       1.00      1.00      1.00     56962

[[56852     2]
 [   23    85]]
```

These are 3 lower result models on data subset:

[9]:
```
svc.fit(X_train, y_train)
y_pred = svc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
Classification_rp = classification_report(y_test, y_pred)
print(Classification_rp)
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56854
           1       0.97      0.67      0.79       108

    accuracy                           1.00     56962
   macro avg       0.99      0.83      0.90     56962
weighted avg       1.00      1.00      1.00     56962

[[56852     2]
 [   36    72]]
```

[10]:
```
dtc.fit(X_train, y_train)
y_pred = dtc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
Classification_rp = classification_report(y_test, y_pred)
print(Classification_rp)
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56854
           1       0.87      0.80      0.83       108

    accuracy                           1.00     56962
   macro avg       0.93      0.90      0.92     56962
weighted avg       1.00      1.00      1.00     56962

[[56841    13]
 [   22    86]]
```

[11]:
```
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
Classification_rp = classification_report(y_test, y_pred)
print(Classification_rp)
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support
```

```
           0       1.00      1.00      1.00     56854
           1       0.82      0.68      0.74       108

    accuracy                           1.00     56962
   macro avg       0.91      0.84      0.87     56962
weighted avg       1.00      1.00      1.00     56962

[[56838    16]
 [   35    73]]
```

We can observe that the three models with lower accuracy rates (Logistic Regression, SVC, Decision Tree) on the subset of the data are likely to yield poor results when predicting the value 1 (‘Fraud’), as the prediction rates are quite low. This suggests that these three models are not effective on the original full dataset. Conversely, the three models with the best results (Random Forest Classifier, Gradient Boosting, XGB) exhibit relatively high prediction rates for the value 1 (‘Fraud’). This demonstrates that on this dataset, these three models have performed exceptionally well in identifying cases of ‘Fraud,’ even though there is a considerable imbalance in the dataset.

2.8 In Conclusion

From the conclusion above, we are only considering the top-performing three models on the original full dataset (Random Forest Classifier, Gradient Boosting, XGB). We can observe that the most effective performance comes from XGB, as this model has accurately identified cases of 1 (‘Fraud’) the best among the three models. Therefore, XGB stands out as the best-performing model for addressing this particular scenario.