In [2]:
```
# Assignment: DSC680 - Project 3 – Credit Card Fraud Detection
# Name: Bezawada, Sashidhar
# Date: 2024-05-20
# Milestone 3 : White Paper
```

In [3]:
```
# Import libraries
import pandas as pd
import numpy as np
import os
from __future__ import print_function, division
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import statistics
import plotly.express as px

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (accuracy_score, log_loss, classification_report)
```

1. Import the data frame and ensure that the data is loaded properly

In [4]:
```
#Load the dataset as a Pandas data frame
#Read train.csv
df = pd.read_csv("datasets/creditcard.csv")
#Display the first ten rows of data
df.info()
df.head(10)
```

```
<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
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	...	-0.208254	-0.559825	-0.026398
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	...	-0.167716	-0.270710	-0.154104
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	...	1.943465	-1.015455	0.057504
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.392048	...	-0.073425	-0.268092	-0.204233
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-0.736727	...	-0.246914	-0.633753	-0.120794

10 rows × 31 columns

The purpose of checking sample data is to get a quick overview of the data and identify any potential problems or areas of interest.

In [5]:

```
# Attributes and Descriptions

#V1 - V28 : Numerical features that are a result of PCA transformation.

#Time : Seconds elapsed between each transaction and the 1st transaction.

#Amount : Transaction amount.

#Class : Fraud or otherwise (1 or 0)
```

There are 31 feature columns. Using these features your model has to predict the Fraud.

Exploratory Data Analysis

In [6]:

```
#Check Shape
df.shape
```

Out[6]: (284807, 31)

In [7]:
```
#Check for Duplicates
have_duplicate_rows = df.duplicated().any()
have_duplicate_rows
```

Out[7]: True

In [8]:
```
#Check for Missing Values
missing_df = df.isnull().sum().to_frame().rename(columns={0:"Total No. of Missing Values"})
missing_df["% of Missing Values"] = round((missing_df["Total No. of Missing Values"]/len(df))*100,2)
missing_df
```

Out[8]:

	Total No. of Missing Values	% of Missing Values
Time	0	0.0
V1	0	0.0
V2	0	0.0
V3	0	0.0
V4	0	0.0
V5	0	0.0
V6	0	0.0
V7	0	0.0
V8	0	0.0
V9	0	0.0
V10	0	0.0
V11	0	0.0
V12	0	0.0
V13	0	0.0
V14	0	0.0
V15	0	0.0
V16	0	0.0
V17	0	0.0
V18	0	0.0
V19	0	0.0
V20	0	0.0
V21	0	0.0
V22	0	0.0
V23	0	0.0
V24	0	0.0

	Total No. of Missing Values	% of Missing Values
V25	0	0.0
V26	0	0.0
V27	0	0.0
V28	0	0.0
Amount	0	0.0
Class	0	0.0

In [9]:

```
#Check for NULL Values
null_train_data = round(100*(df.isnull().sum().sort_values(ascending=False)/len(df.index)),2)\
                    .to_frame().rename(columns={0:'Train Null values percentage'})[:20]
null_train_data
```

Out[9]:

	Train Null values percentage
Time	0.0
V16	0.0
Amount	0.0
V28	0.0
V27	0.0
V26	0.0
V25	0.0
V24	0.0
V23	0.0
V22	0.0
V21	0.0
V20	0.0
V19	0.0
V18	0.0
V17	0.0
V15	0.0
V1	0.0
V14	0.0
V13	0.0
V12	0.0

In [10]:

```
df_numcols = df.select_dtypes(include=['int64', 'float64'])

print(f"Feature | # 0 Values | # Null Values | # Unique Values ")
print("="*60)
for feature in df_numcols:
    zero_values = (df[feature] == 0).sum()
    null_values = df[feature].isnull().sum()
```

```
    unique_values = len(df[feature].unique())

    print(f"{feature} | {zero_values} | {zero_values} | {unique_values} ")
```

```
Feature | # 0 Values | # Null Values | # Unique Values
============================================================
Time | 2 | 2 | 124592
V1 | 0 | 0 | 275663
V2 | 0 | 0 | 275663
V3 | 0 | 0 | 275663
V4 | 0 | 0 | 275663
V5 | 0 | 0 | 275663
V6 | 0 | 0 | 275663
V7 | 0 | 0 | 275663
V8 | 0 | 0 | 275663
V9 | 0 | 0 | 275663
V10 | 0 | 0 | 275663
V11 | 0 | 0 | 275663
V12 | 0 | 0 | 275663
V13 | 0 | 0 | 275663
V14 | 0 | 0 | 275663
V15 | 0 | 0 | 275663
V16 | 0 | 0 | 275663
V17 | 0 | 0 | 275663
V18 | 0 | 0 | 275663
V19 | 0 | 0 | 275663
V20 | 0 | 0 | 275663
V21 | 0 | 0 | 275663
V22 | 0 | 0 | 275663
V23 | 0 | 0 | 275663
V24 | 0 | 0 | 275663
V25 | 0 | 0 | 275663
V26 | 0 | 0 | 275663
V27 | 0 | 0 | 275663
V28 | 0 | 0 | 275663
Amount | 1825 | 1825 | 32767
Class | 284315 | 284315 | 2
```

We note that the data looks normalized, this can be as a result of the desensitization process or through decomposition processes (i.e. PCA)

In [11]:

```
#Describing each field in the dataset *( Transpose the table)
df.describe(exclude="O").T
```

Out[11]:

	count	mean	std	min	25%	50%	75%	max
Time	284807.0	9.481386e+04	47488.145955	0.000000	54201.500000	84692.000000	139320.500000	172792.000000
V1	284807.0	1.168375e-15	1.958696	-56.407510	-0.920373	0.018109	1.315642	2.454930
V2	284807.0	3.416908e-16	1.651309	-72.715728	-0.598550	0.065486	0.803724	22.057729
V3	284807.0	-1.379537e-15	1.516255	-48.325589	-0.890365	0.179846	1.027196	9.382558
V4	284807.0	2.074095e-15	1.415869	-5.683171	-0.848640	-0.019847	0.743341	16.875344
V5	284807.0	9.604066e-16	1.380247	-113.743307	-0.691597	-0.054336	0.611926	34.801666
V6	284807.0	1.487313e-15	1.332271	-26.160506	-0.768296	-0.274187	0.398565	73.301626
V7	284807.0	-5.556467e-16	1.237094	-43.557242	-0.554076	0.040103	0.570436	120.589494
V8	284807.0	1.213481e-16	1.194353	-73.216718	-0.208630	0.022358	0.327346	20.007208
V9	284807.0	-2.406331e-15	1.098632	-13.434066	-0.643098	-0.051429	0.597139	15.594995
V10	284807.0	2.239053e-15	1.088850	-24.588262	-0.535426	-0.092917	0.453923	23.745136
V11	284807.0	1.673327e-15	1.020713	-4.797473	-0.762494	-0.032757	0.739593	12.018913
V12	284807.0	-1.247012e-15	0.999201	-18.683715	-0.405571	0.140033	0.618238	7.848392
V13	284807.0	8.190001e-16	0.995274	-5.791881	-0.648539	-0.013568	0.662505	7.126883
V14	284807.0	1.207294e-15	0.958596	-19.214325	-0.425574	0.050601	0.493150	10.526766
V15	284807.0	4.887456e-15	0.915316	-4.498945	-0.582884	0.048072	0.648821	8.877742
V16	284807.0	1.437716e-15	0.876253	-14.129855	-0.468037	0.066413	0.523296	17.315112
V17	284807.0	-3.772171e-16	0.849337	-25.162799	-0.483748	-0.065676	0.399675	9.253526
V18	284807.0	9.564149e-16	0.838176	-9.498746	-0.498850	-0.003636	0.500807	5.041069
V19	284807.0	1.039917e-15	0.814041	-7.213527	-0.456299	0.003735	0.458949	5.591971
V20	284807.0	6.406204e-16	0.770925	-54.497720	-0.211721	-0.062481	0.133041	39.420904
V21	284807.0	1.654067e-16	0.734524	-34.830382	-0.228395	-0.029450	0.186377	27.202839
V22	284807.0	-3.568593e-16	0.725702	-10.933144	-0.542350	0.006782	0.528554	10.503090
V23	284807.0	2.578648e-16	0.624460	-44.807735	-0.161846	-0.011193	0.147642	22.528412
V24	284807.0	4.473266e-15	0.605647	-2.836627	-0.354586	0.040976	0.439527	4.584549

	count	mean	std	min	25%	50%	75%	max
V25	284807.0	5.340915e-16	0.521278	-10.295397	-0.317145	0.016594	0.350716	7.519589
V26	284807.0	1.683437e-15	0.482227	-2.604551	-0.326984	-0.052139	0.240952	3.517346
V27	284807.0	-3.660091e-16	0.403632	-22.565679	-0.070840	0.001342	0.091045	31.612198
V28	284807.0	-1.227390e-16	0.330083	-15.430084	-0.052960	0.011244	0.078280	33.847808
Amount	284807.0	8.834962e+01	250.120109	0.000000	5.600000	22.000000	77.165000	25691.160000
Class	284807.0	1.727486e-03	0.041527	0.000000	0.000000	0.000000	0.000000	1.000000

In [12]:

```
#check if there any duplication
df.duplicated().sum()
```

Out[12]:

```
1081
```

Notes on basic EDA :

1. No null values. No need to use imputation
2. Categorical data ==> No Categorical data
3. Data types are all float values excluding the target (integer)
4. Data is very large with only 284807 datapoints
5. Duplicates: Dataset have duplicates and not good for the model training, so, removing these duplicates

Data Transformations

In [13]:

```
# drop duplication

df.drop_duplicates(df,inplace=True)
```

In [14]:

```
df.shape## Data Transformations
```

Out[14]:

```
(283726, 31)
```

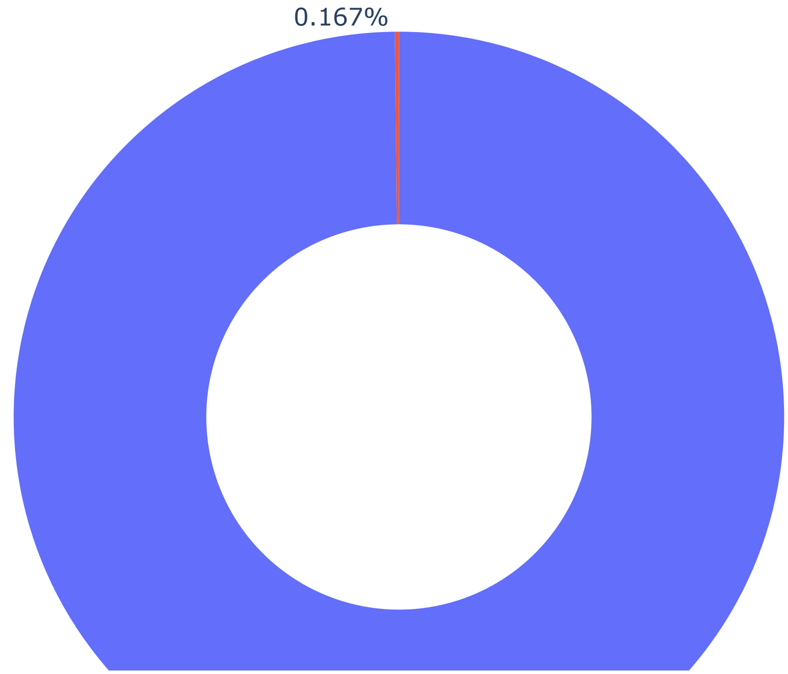
In [15]:

```
df.Class.value_counts()
```

Out[15]:
```
Class
0    283253
1       473
Name: count, dtype: int64
```

Charts

In [16]:
```
index = df.Class.value_counts().index
value = df.Class.value_counts().values
px.pie(data_frame = df,
      names = index,
      values = value,
      hole = .5)## Charts
```

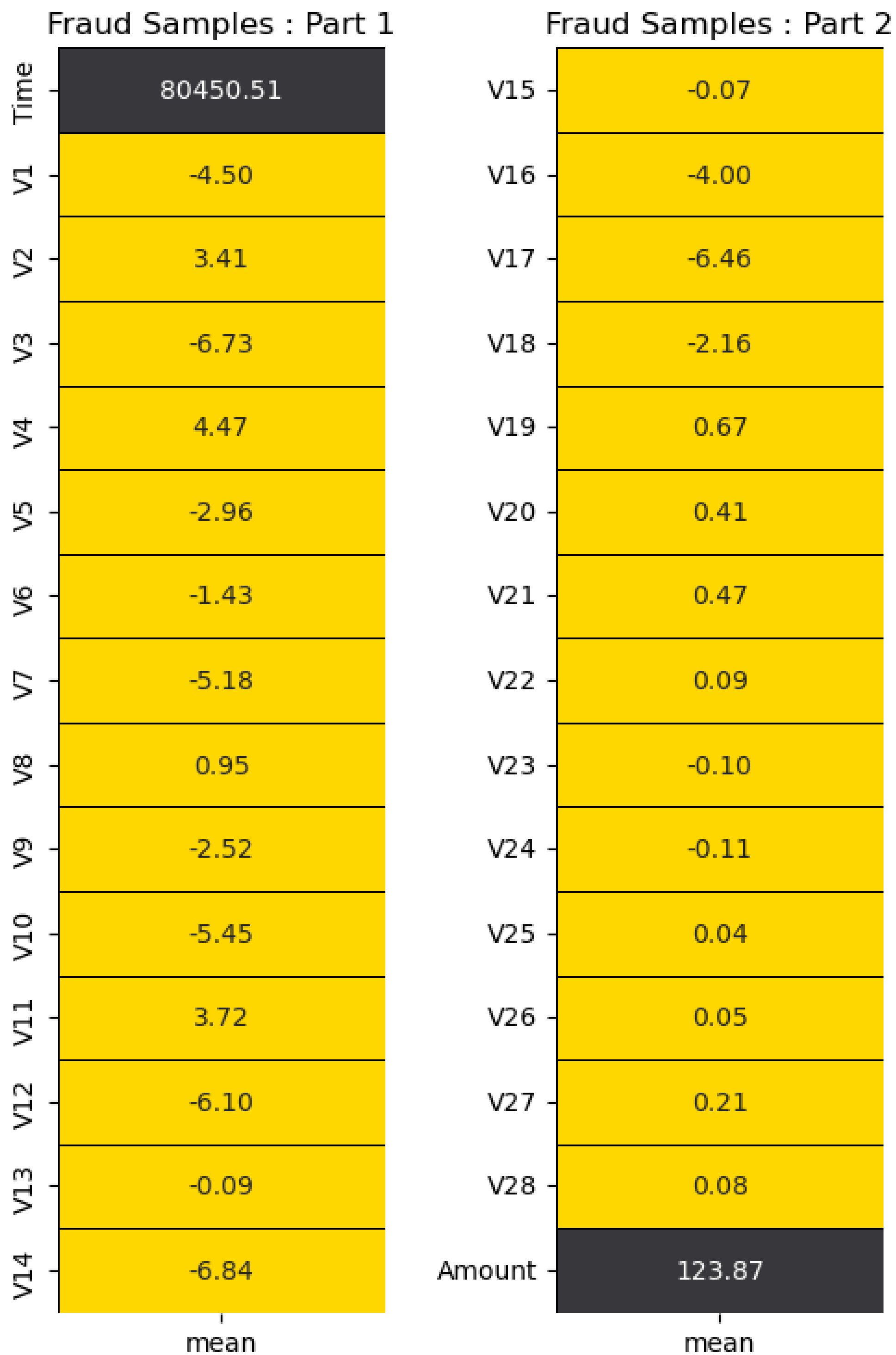


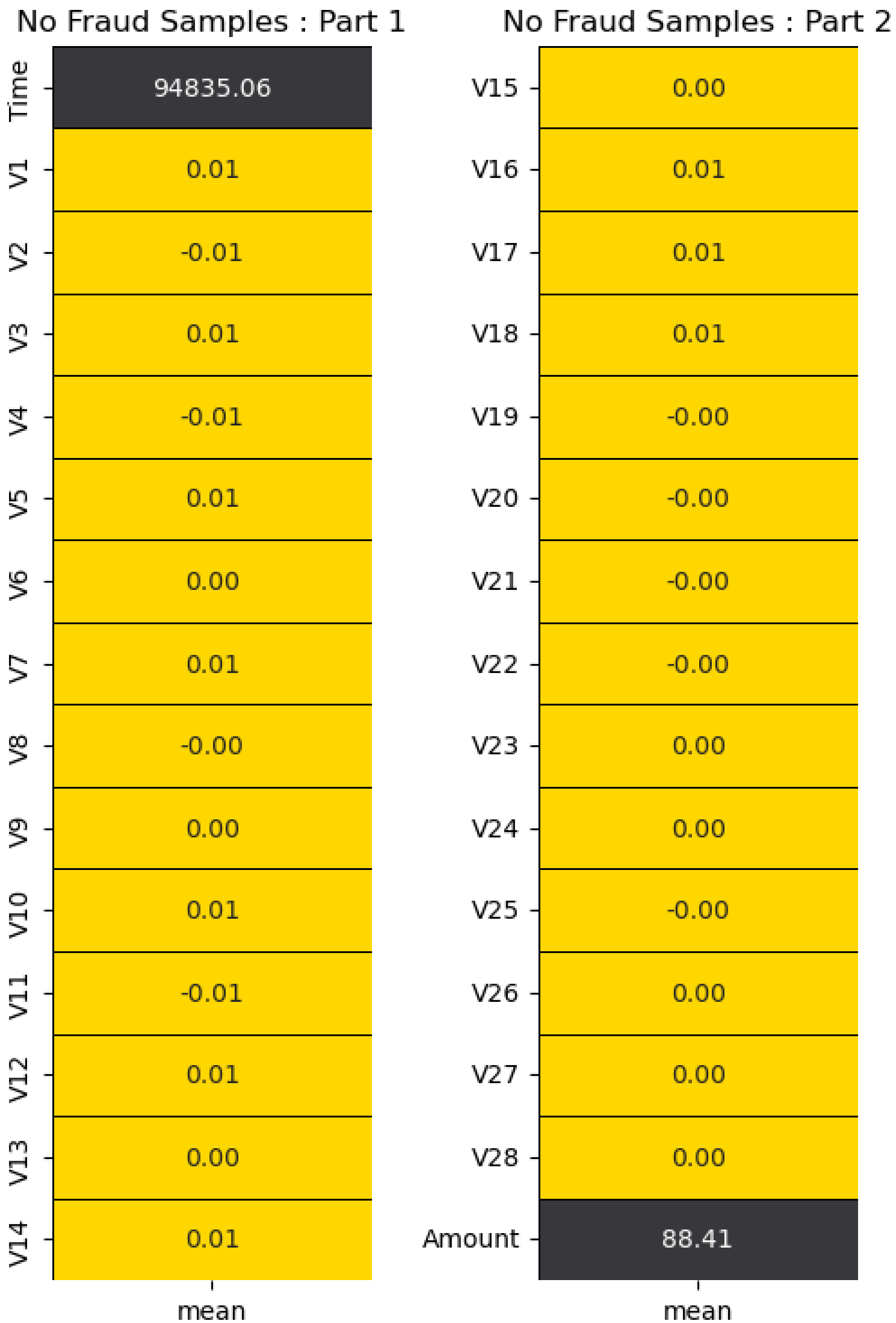
In [17]:
```
fraud = df[df.Class == 1]
fraud_desc = df[df.Class == 1].describe().T

nofraud = df[df.Class == 0]
nofraud_desc = df[df.Class == 0].describe().T
```

In [18]:
```
fraud.shape
```

Out[18]:
```
(473, 31)
```

In [19]:
```
nofraud.shape
```
Out[19]:
```
(283253, 31)
```
In [20]:
```
colors = ['#FFD700','#3B3B3C']

fig,ax = plt.subplots(nrows = 2,ncols = 2,figsize = (5,15))
plt.subplot(2,2,1)
sns.heatmap(fraud_desc[['mean']][:15],annot = True,cmap = colors,linewidths = 0.5,linecolor = 'black',cbar = False,fmt
plt.title('Fraud Samples : Part 1');

plt.subplot(2,2,2)
sns.heatmap(fraud_desc[['mean']][15:30],annot = True,cmap = colors,linewidths = 0.5,linecolor = 'black',cbar = False,fm
plt.title('Fraud Samples : Part 2');

plt.subplot(2,2,3)
sns.heatmap(nofraud_desc[['mean']][:15],annot = True,cmap = colors,linewidths = 0.5,linecolor = 'black',cbar = False,fm
plt.title('No Fraud Samples : Part 1');

plt.subplot(2,2,4)
sns.heatmap(nofraud_desc[['mean']][15:30],annot = True,cmap = colors,linewidths = 0.5,linecolor = 'black',cbar = False,
plt.title('No Fraud Samples : Part 2');

fig.tight_layout(w_pad = 2)
```


Fraud Samples : Part 1
Time
80450.51
V1
-4.50
V2
3.41
V3
-6.73
V4
4.47
V5
-2.96
V6
-1.43
V7
-5.18
V8
0.95
V9
-2.52
V10
-5.45
V11
3.72
V12
-6.10
V13
-0.09
V14
-6.84
mean
Fraud Samples : Part 2
V15
-0.07
V16
-4.00
V17
-6.46
V18
-2.16
V19
0.67
V20
0.41
V21
0.47
V22
0.09
V23
-0.10
V24
-0.11
V25
0.04
V26
0.05
V27
0.21
V28
0.08
Amount
123.87
mean


No Fraud Samples : Part 1
No Fraud Samples : Part 2
Time
94835.06
V1
0.01
V2
-0.01
V3
0.01
V4
-0.01
V5
0.01
V6
0.00
V7
0.01
V8
-0.00
V9
0.00
V10
0.01
V11
-0.01
V12
0.01
V13
0.00
V14
0.01
V15
0.00
V16
0.01
V17
0.01
V18
0.01
V19
-0.00
V20
-0.00
V21
-0.00
V22
-0.00
V23
0.00
V24
0.00
V25
-0.00
V26
0.00
V27
0.00
V28
0.00
Amount
88.41
mean
mean

Mean values of features for Fraud & No Fraud cases!

For No Fraud cases : V1 - V28 mean values are almost 0 for all the cases. And. Mean Amount, 88.29, is less than the mean transaction amount, 122.21, of the Fraud cases.

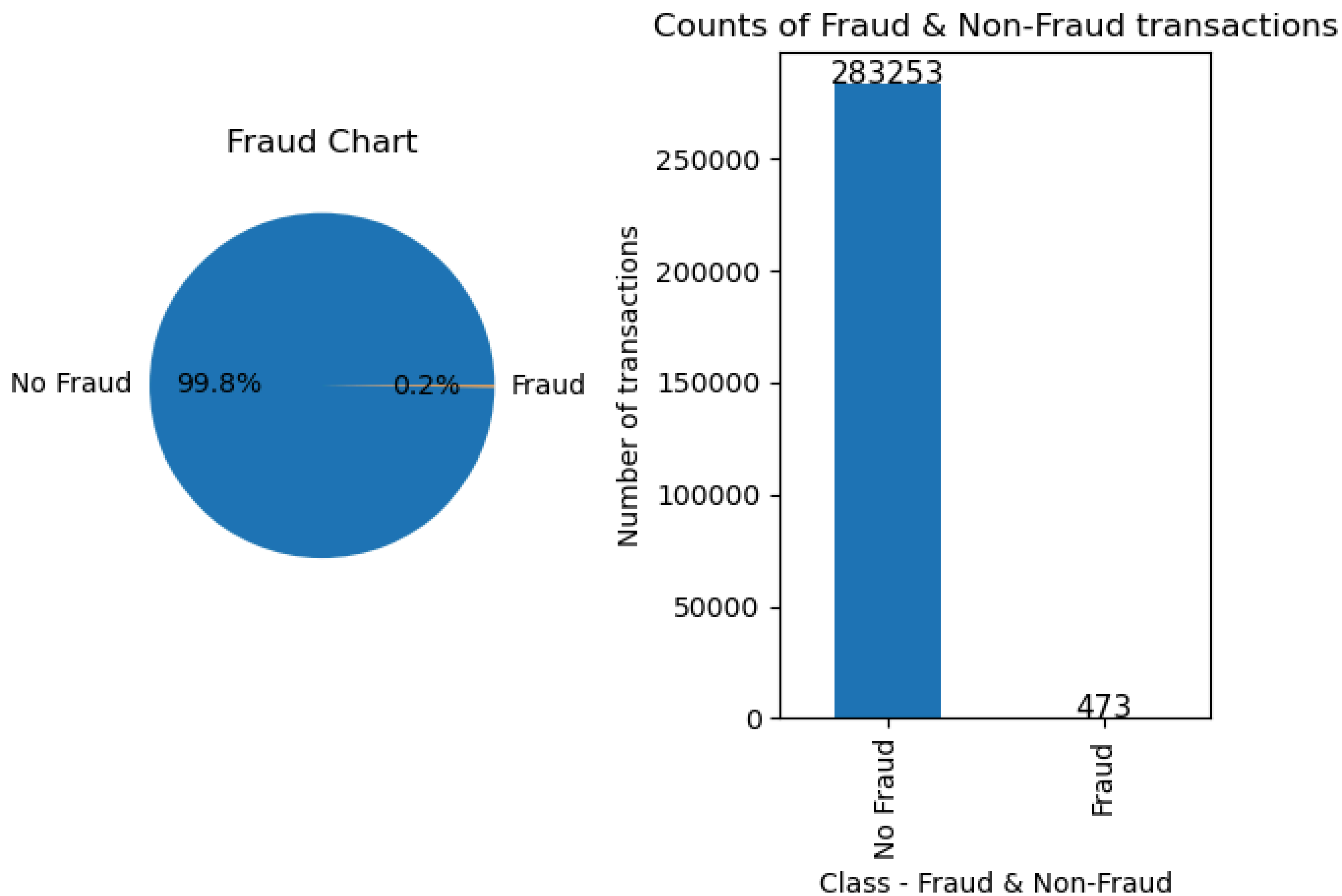
Time taken for No Fraud transactions is more than those for Fraud transactions.

These could be some of the indicators for detecting fraud transactions.

Target Variable Visualization (Class)

In [21]:

```
#Visualization to show Total Employees by Businees Travel.
plt.subplot(1,2,1)
value_1 = df['Class'].value_counts()
plt.title("Fraud Chart")
plt.pie(value_1.values, labels = ['No Fraud','Fraud'], autopct="%.1f%%")

#Visualization to show Employee Attrition by Gender.
plt.subplot(1,2,2)
new_df_fraud=df[df['Class']==1]
ax=df['Class'].value_counts().plot(kind='bar')
plt.title("Counts of Fraud & Non-Fraud transactions")
plt.xlabel('Class - Fraud & Non-Fraud')
plt.ylabel('Number of transactions')
ax.set_xticklabels(['No Fraud','Fraud'])
# Add annotation to bars
for rect in ax.patches:
    ax.text(rect.get_x() + rect.get_width() / 2, rect.get_height() + 2, rect.get_height(), horizontalalignment='center'
plt.tight_layout()
plt.show()
```



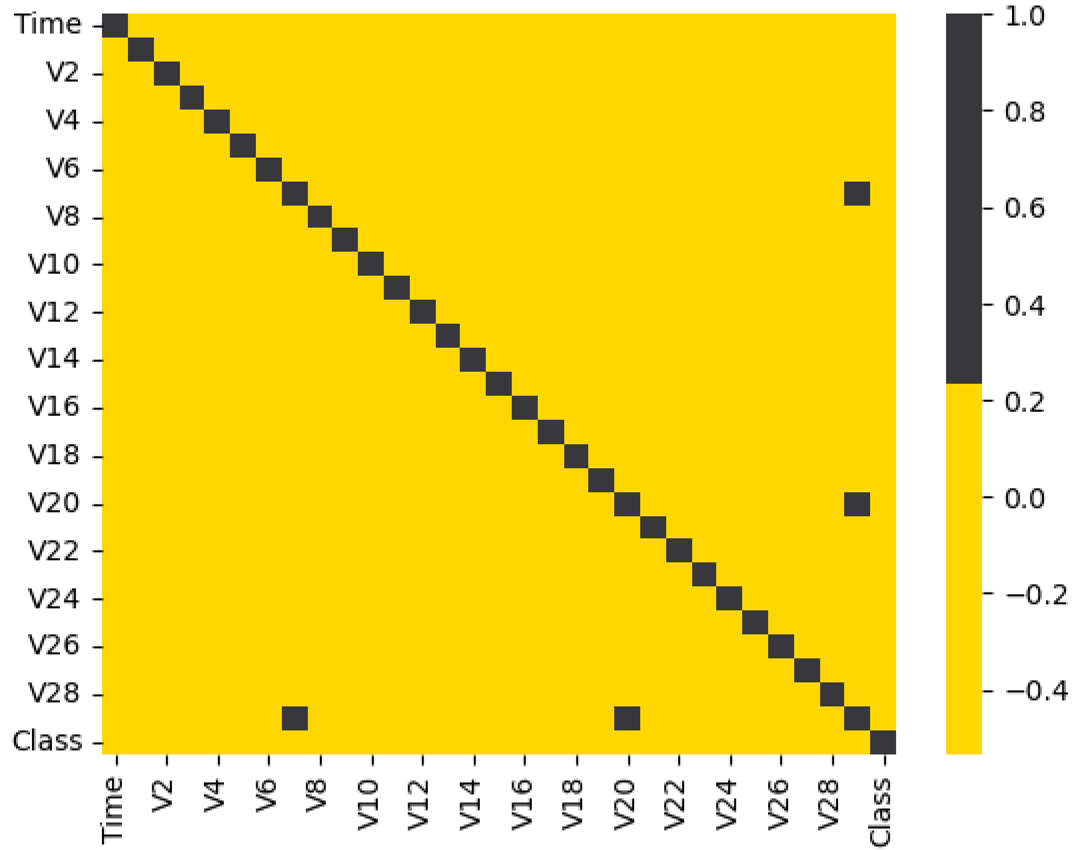
The data is clearly highly unbalanced with majority of the transactions being No Fraud. And the classification model will bias its prediction towards the majority class, No Fraud.

Hence, data balancing is required for building a robust model.

Correlation Matrix :

In [22]:
```
sns.heatmap(df.corr(),cmap = colors,cbar = True)
```

Out[22]:
```
<Axes: >
```

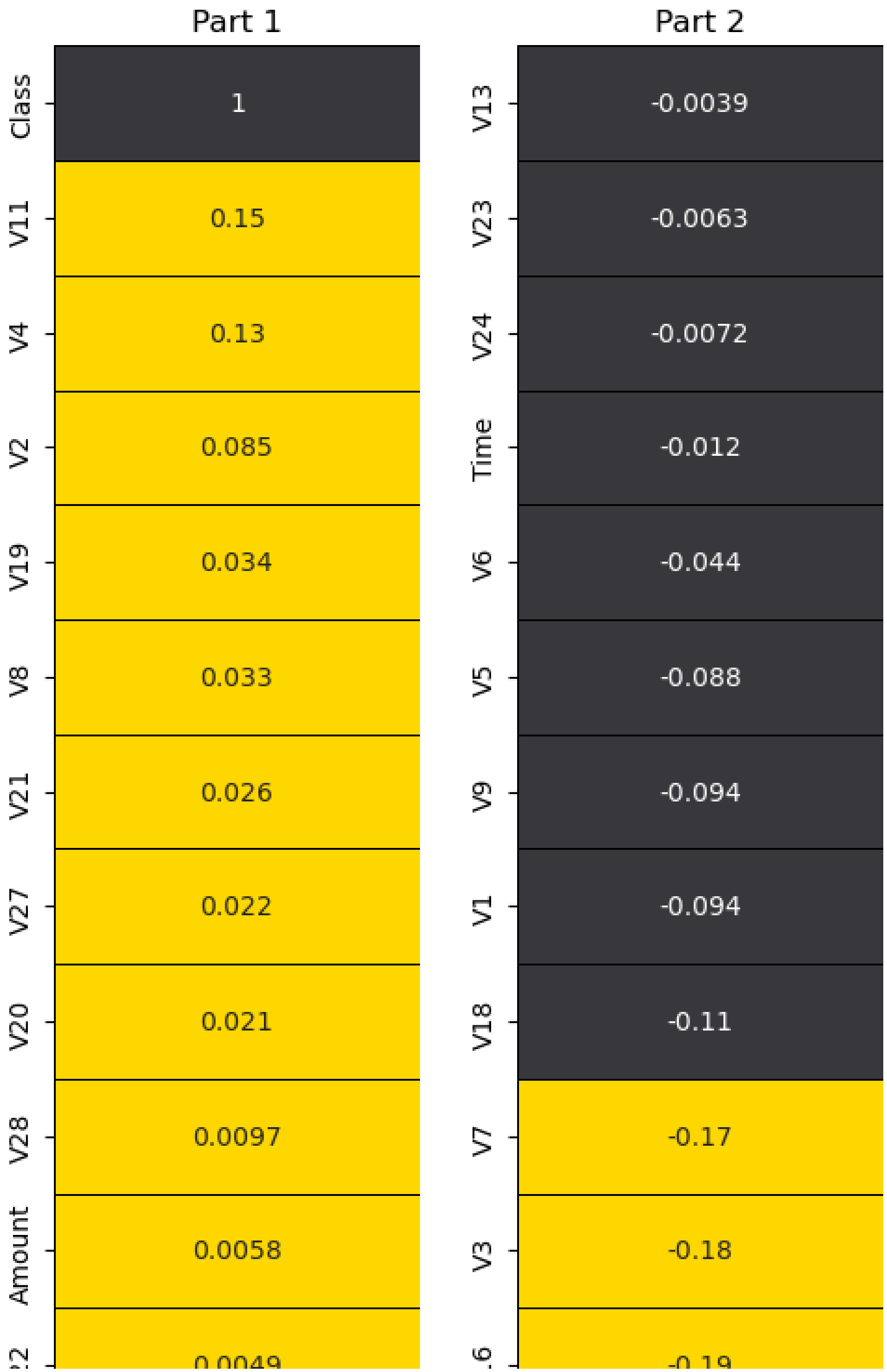


There are too many features in the dataset and it is difficult to understand anything.

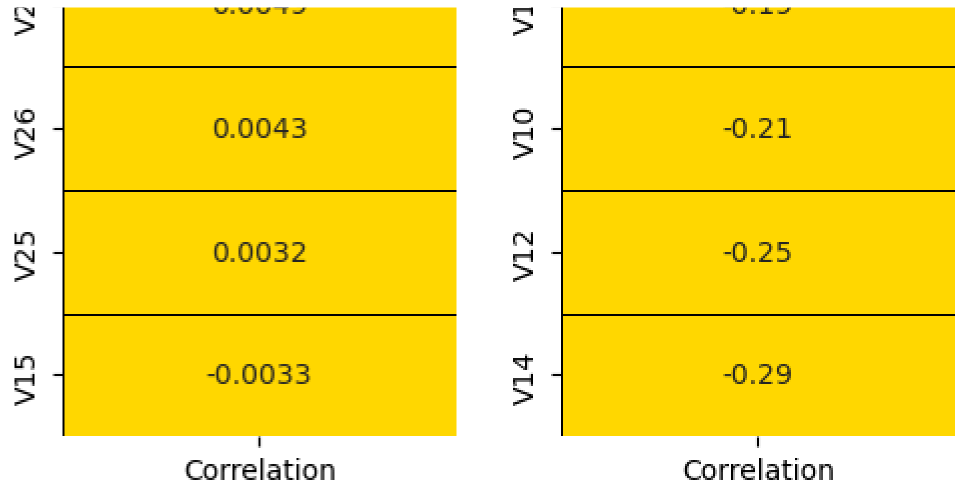
Hence, we will plot the correlation map only with the target variable.

In [23]:
```
corr = df.corrwith(df['Class']).sort_values(ascending = False).to_frame()
corr.columns = ['Correlation']
fig,ax = plt.subplots(nrows = 1,ncols = 2,figsize = (5,10))

plt.subplot(1,2,1)
sns.heatmap(corr.iloc[:15,:],annot = True,cmap = colors,linewidths = 0.4,linecolor = 'black',cbar = False)
plt.title('Part 1')

plt.subplot(1,2,2)
sns.heatmap(corr.iloc[15:30],annot = True,cmap = colors,linewidths = 0.4,linecolor = 'black',cbar = False)
plt.title('Part 2')
```

```
fig.tight_layout(w_pad = 2)
```



Part 1
Part 2
Class
1
V11
0.15
V4
0.13
V2
0.085
V19
0.034
V8
0.033
V21
0.026
V27
0.022
V20
0.021
V28
0.0097
Amount
0.0058
V13
-0.0039
V23
-0.0063
V24
-0.0072
Time
-0.012
V6
-0.044
V5
-0.088
V9
-0.094
V1
-0.094
V18
-0.11
V7
-0.17
V3
-0.18



For feature selection, we will exclude the features having correlation values between [-0.1,0.1].

V4, V11 are positively correlated and V7, V3, V16, V10, V12, V14, V17 are negatively correlated with the Class feature.

ANOVA Test

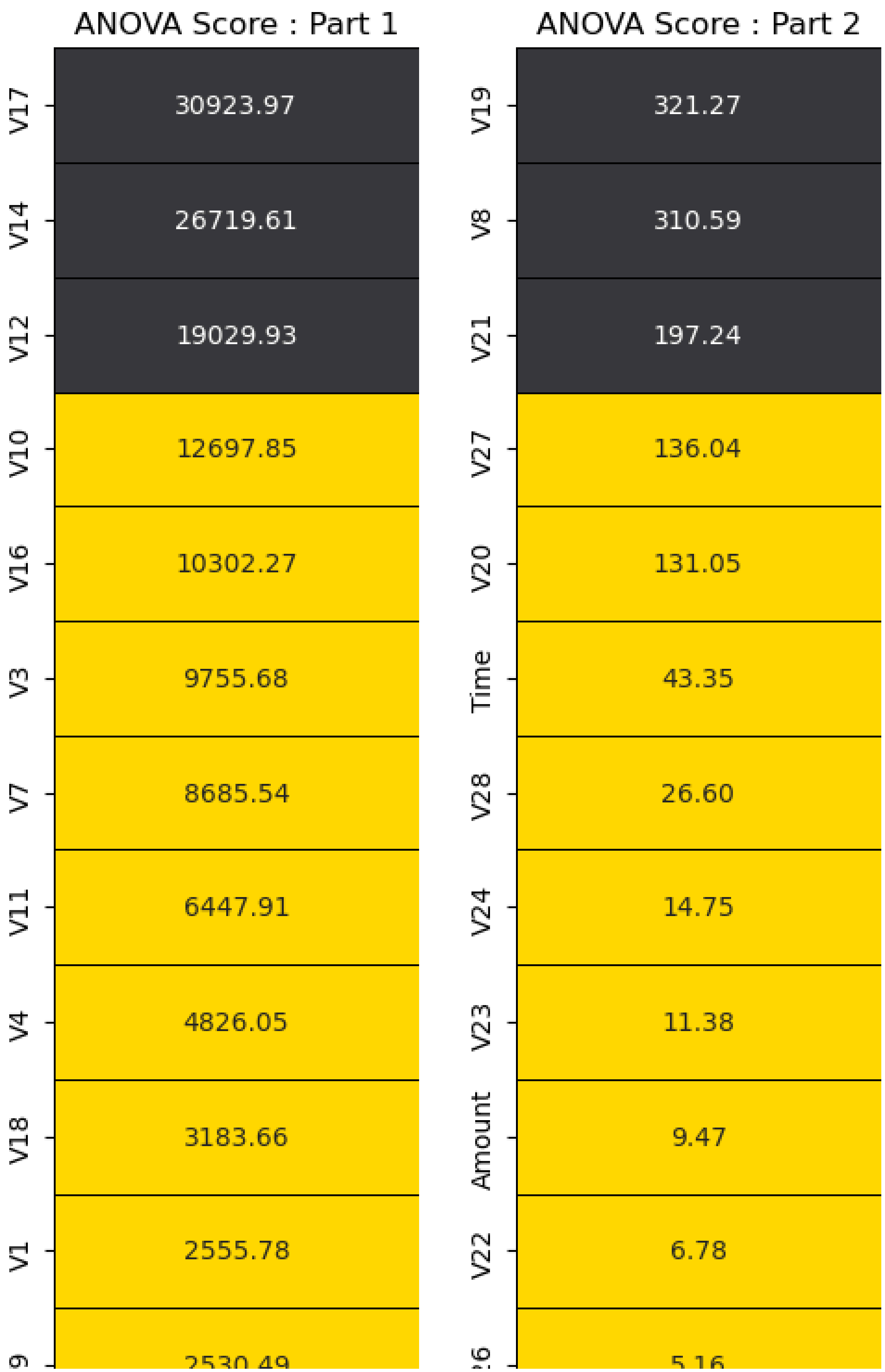
In [24]:

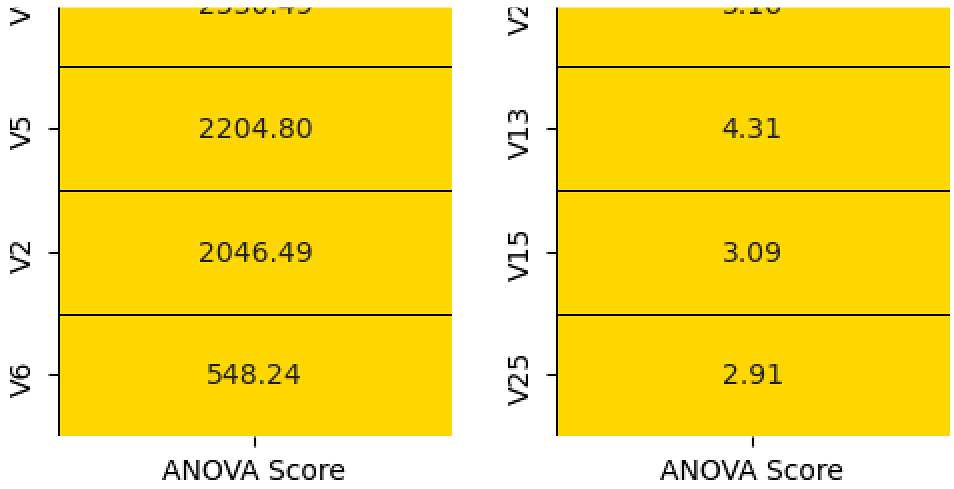
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
```

In [25]:

```
features = df.loc[:,:'Amount']
target = df.loc[:,'Class']

best_features = SelectKBest(score_func = f_classif,k = 'all')
fit = best_features.fit(features,target)

featureScores = pd.DataFrame(data = fit.scores_,index = list(features.columns),columns = ['ANOVA Score'])
featureScores = featureScores.sort_values(ascending = False,by = 'ANOVA Score')

fig,ax = plt.subplots(nrows = 1,ncols = 2,figsize = (5,10))

plt.subplot(1,2,1)
sns.heatmap(featureScores.iloc[:15,:],annot = True,cmap = colors,linewidths = 0.4,linecolor = 'black',cbar = False, fmt
plt.title('ANOVA Score : Part 1')

plt.subplot(1,2,2)
sns.heatmap(featureScores.iloc[15:30],annot = True,cmap = colors,linewidths = 0.4,linecolor = 'black',cbar = False, fmt
```

```
plt.title('ANOVA Score : Part 2')

fig.tight_layout(w_pad = 2)
```


ANOVA Score : Part 1
V17
30923.97
V14
26719.61
V12
19029.93
V10
12697.85
V16
10302.27
V3
9755.68
V7
8685.54
V11
6447.91
V4
4826.05
V18
3183.66
V1
2555.78
2530.49
ANOVA Score : Part 2
V19
321.27
V8
310.59
V21
197.24
V27
136.04
V20
131.05
Time
43.35
V28
26.60
V24
14.75
V23
11.38
Amount
9.47
V22
6.78
5.16



Note : Higher the value of the ANOVA score, higher the importance of that feature with the target variable.

From the above plot, we will reject features with values less than 50.

Also, 2 Datasets are created based on Correlation plot and other based on ANOVA Scores.

In [26]:

```
# Dataset for Model based on Correlation Plot
df1 = df[['V3','V4','V7','V10','V11','V12','V14','V16','V17','Class']]
df1.head()
```

Out[26]:

	V3	V4	V7	V10	V11	V12	V14	V16	V17	Class
0	2.536347	1.378155	0.239599	0.090794	-0.551600	-0.617801	-0.311169	-0.470401	0.207971	0
1	0.166480	0.448154	-0.078803	-0.166974	1.612727	1.065235	-0.143772	0.463917	-0.114805	0
2	1.773209	0.379780	0.791461	0.207643	0.624501	0.066084	-0.165946	-2.890083	1.109969	0
3	1.792993	-0.863291	0.237609	-0.054952	-0.226487	0.178228	-0.287924	-1.059647	-0.684093	0
4	1.548718	0.403034	0.592941	0.753074	-0.822843	0.538196	-1.119670	-0.451449	-0.237033	0

In [27]:

```
# Dataset for Model based on Correlation Plot
df2 = df.drop(columns = list(featureScores.index[20:]))
df2.head()
```

Out[27]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V12	V14	V1
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.617801	-0.311169	-0.4704
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	1.065235	-0.143772	0.4639
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.066084	-0.165946	-2.8900
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	0.178228	-0.287924	-1.0596
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	0.538196	-1.119670	-0.4514

5 rows × 21 columns

Data Balancing

In order to cope with unbalanced data, there are 2 options :

Undersampling : Trim down the majority samples of the target variable.

Oversampling : Increase the minority samples of the target variable to the majority samples.

For best performances, I will be using oversampling technique known as SMOTE to treat the imbalance.

Building a model and evaluating

Having performed some exploratory data analysis and simple feature engineering as well as having ensured that all categorical values are encoded, we are now ready to proceed onto building our models.

Will evaluate using different learning models.

In [28]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import RocCurveDisplay
#from sklearn.metrics import plot_roc_curve
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
```

```python
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.metrics import precision_recall_curve
from imblearn.over_sampling import SMOTE
```

In [29]:

```python
# matrices of features
# Based on Correlation Plot
x1 = df1.drop(labels=['Class'],axis=1)
y1 = df1['Class']
col=x1.columns
x1.info()

# Based on ANOVA Score
x2 = df2.drop(labels=['Class'],axis=1)
y2 = df2['Class']
col=x2.columns
x2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 283726 entries, 0 to 284806
Data columns (total 9 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   V3      283726 non-null  float64
 1   V4      283726 non-null  float64
 2   V7      283726 non-null  float64
 3   V10     283726 non-null  float64
 4   V11     283726 non-null  float64
 5   V12     283726 non-null  float64
 6   V14     283726 non-null  float64
 7   V16     283726 non-null  float64
 8   V17     283726 non-null  float64
dtypes: float64(9)
memory usage: 21.6 MB
<class 'pandas.core.frame.DataFrame'>
Index: 283726 entries, 0 to 284806
Data columns (total 20 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   V1      283726 non-null  float64
 1   V2      283726 non-null  float64
 2   V3      283726 non-null  float64
 3   V4      283726 non-null  float64
 4   V5      283726 non-null  float64
 5   V6      283726 non-null  float64
 6   V7      283726 non-null  float64
 7   V8      283726 non-null  float64
 8   V9      283726 non-null  float64
 9   V10     283726 non-null  float64
 10  V11     283726 non-null  float64
 11  V12     283726 non-null  float64
 12  V14     283726 non-null  float64
 13  V16     283726 non-null  float64
 14  V17     283726 non-null  float64
 15  V18     283726 non-null  float64
 16  V19     283726 non-null  float64
 17  V20     283726 non-null  float64
 18  V21     283726 non-null  float64
 19  V27     283726 non-null  float64
dtypes: float64(20)
memory usage: 45.5 MB
```

In [30]:
```
# Import the train_test_split method
# Split data into train and test sets as well as for validation and testing ( Splitting the data into 70 - 30 train - te

sm = SMOTE()
x_train1, x_test1, y_train1, y_test1 = train_test_split(x1, y1, train_size= 0.60, random_state=10);
x_train1_smote, y_train1_smote = sm.fit_resample(x_train1,y_train1)

x_train2, x_test2, y_train2, y_test2 = train_test_split(x2, y2, train_size = 0.60, random_state=10);
x_train2_smote, y_train2_smote = sm.fit_resample(x_train2,y_train2)
```

In [31]:
```
# summarize the new class distribution (Based on Correlation Plot)
from collections import Counter
counter = Counter(y_train1)
print(counter)
counter = Counter(y_test1)
print(counter)
counter = Counter(y_train1_smote)
print(counter)
```

```
Counter({0: 169935, 1: 300})
Counter({0: 113318, 1: 173})
Counter({0: 169935, 1: 169935})
```

In [32]:
```
# summarize the new class distribution (Based on ANOVA Score)
from collections import Counter
counter = Counter(y_train2)
print(counter)
counter = Counter(y_test2)
print(counter)
counter = Counter(y_train2_smote)
print(counter)
```

```
Counter({0: 169935, 1: 300})
Counter({0: 113318, 1: 173})
Counter({0: 169935, 1: 169935})
```

In [33]:
```
# Sklearn regression algorithms
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler
```

```python
from sklearn.preprocessing import OneHotEncoder

# Sklearn regression model evaluation function
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_report, roc_curve, precisio
from sklearn.metrics import f1_score
from sklearn.metrics import balanced_accuracy_score
import warnings

from sklearn.datasets import make_classification
from numpy import mean


from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
from sklearn import metrics
from math import sqrt
from sklearn.metrics import mean_squared_error
```

In [34]:

```python
# Confusion Matrix
def plot_confusion_matrix() :
    x_axis_labels = ['nofraud','fraud']# labels for x-axis
    y_axis_labels = ['nofraud','fraud'] # labels for y-axis
    names = ['True Neg','False Pos','False Neg','True Pos']
    counts = [value for value in cm.flatten()]
    percentages = ['{0:.2%}'.format(value) for value in cm.flatten()/np.sum(cm)]
    labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in zip(names,counts,percentages)]
    labels = np.asarray(labels).reshape(2,2)
    sns.heatmap(cm,annot=labels,fmt='',cmap=plt.cm.Blues,xticklabels=x_axis_labels, yticklabels=y_axis_labels)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show
```

In [35]:

```python
# Precision, Recall, F1 Score
def show_metrics(y_test,y_predict):
    print('RMSE is                  :',sqrt(mean_squared_error(y_test, y_predict)))
    print("Accuracy of model        :",accuracy_score(y_test, y_predict))
    print("Precision Score          :",precision_score(y_test, y_predict))
    print("F1 Score                 :",f1_score(y_test, y_predict))
    print("Recall Score             :",recall_score(y_test, y_predict))
```

```
    print("AUC Score               :",metrics.roc_auc_score(y_test,y_predict))
    print("Balanced Accuracy Score :",balanced_accuracy_score(y_test, y_predict))
```

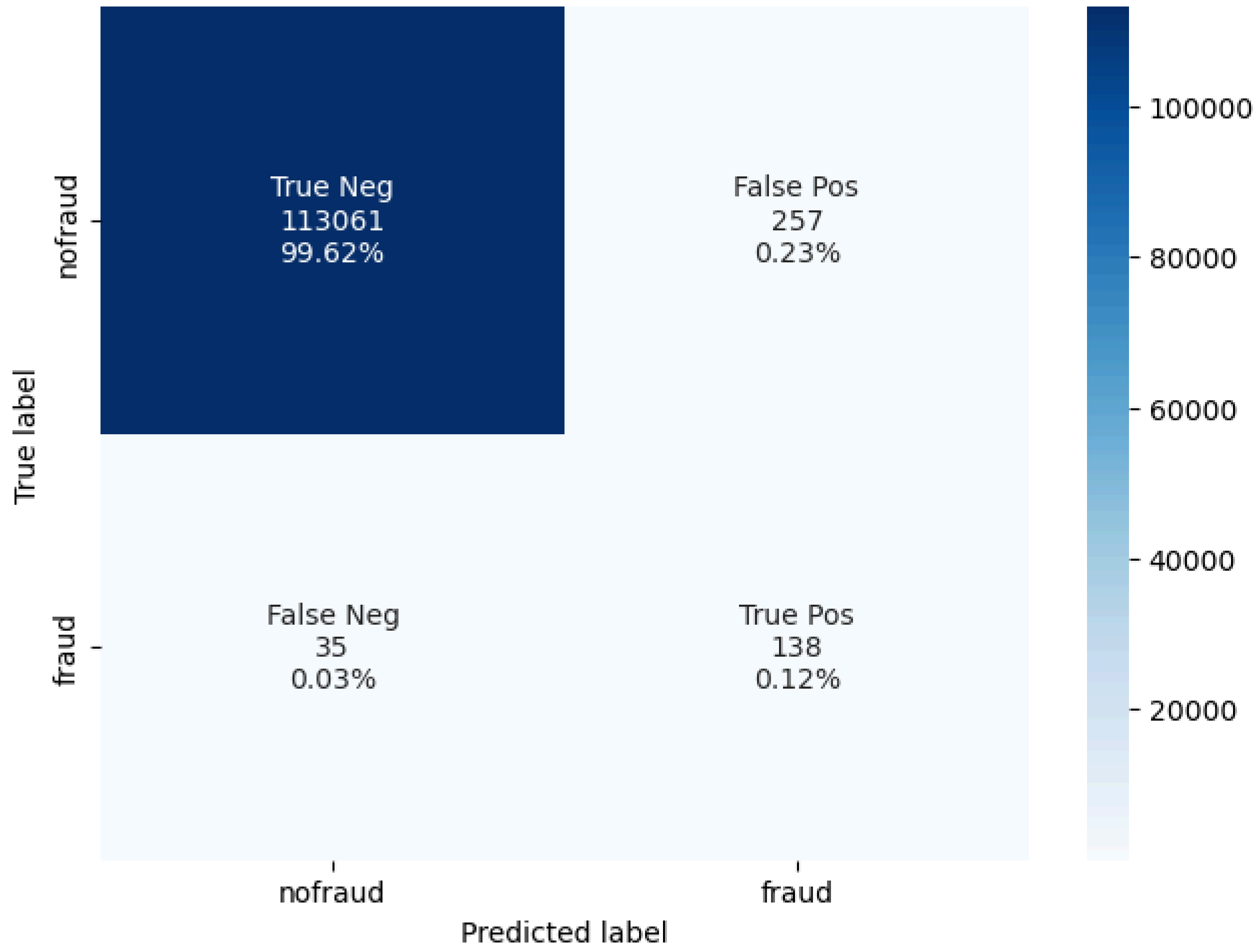
In [36]:
```
# ROC curve
def plot_roc(y_test, logpred):
    roc_auc = metrics.roc_auc_score(y_test, logpred)
    fpr, tpr, thresholds = metrics.roc_curve(y_test, logpred)
    plt.figure()

    plt.plot(fpr, tpr, label = 'ROC curve', color ='orange', linewidth = 2)
    plt.plot([0,1],[0,1], 'k--', linewidth = 2)
    plt.xlim([0.0,1.0])
    plt.ylim([0.0,1.0])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.show();
```

K-Nearest Neighbor

In [38]:
```
# Model based on Correlation Plot :
# Use the KNN classifier to fit data:
K_value=3
classifier = KNeighborsClassifier(n_neighbors = K_value, leaf_size = 1 , algorithm = 'auto')
classifier.fit(x_train1_smote, y_train1_smote)
# Predict y data with classifier:
y_predict1 = classifier.predict(x_test1)
show_metrics(y_test1,y_predict1)
```

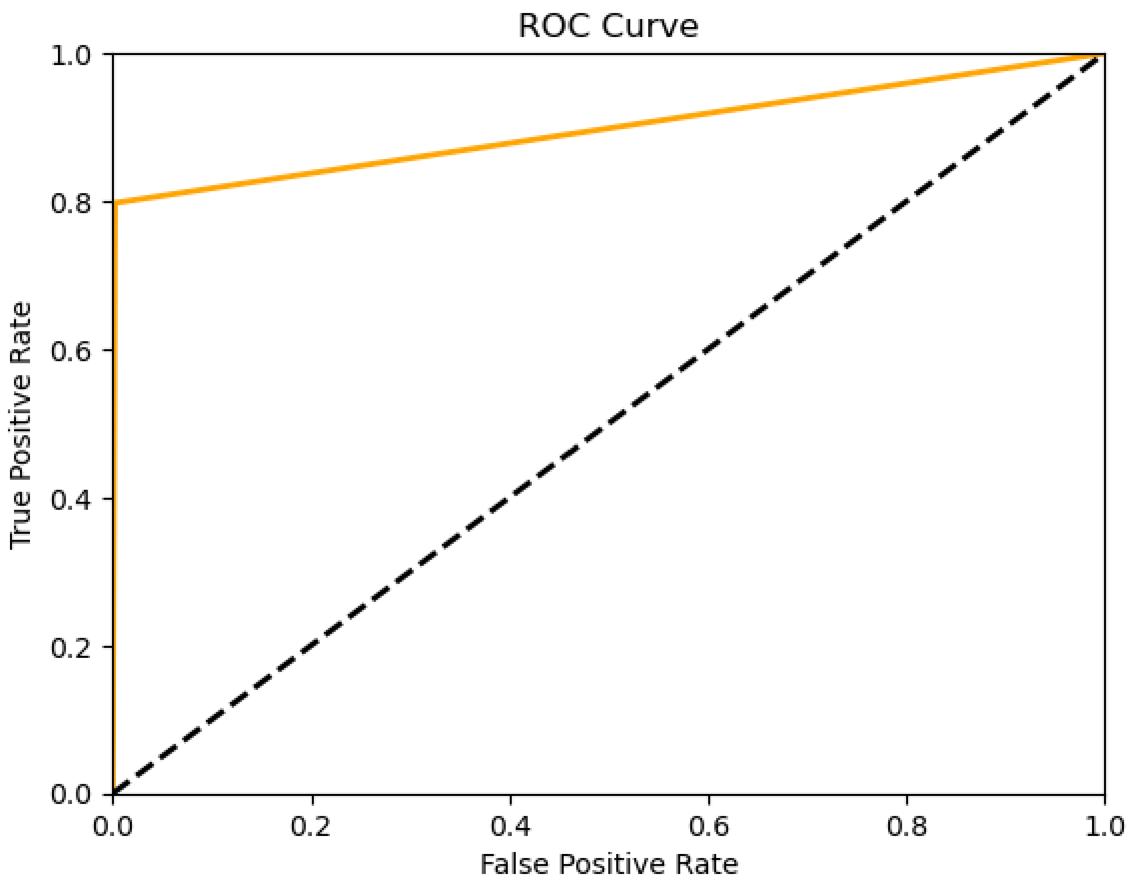
```
RMSE is                 : 0.05072367536444276
Accuracy of model       : 0.9974271087575226
Precision Score         : 0.3493670886075949
F1 Score                : 0.48591549295774644
Recall Score            : 0.7976878612716763
AUC Score               : 0.8977099536860156
Balanced Accuracy Score : 0.8977099536860156
```

In [39]:
```
# Print results:
cm=metrics.confusion_matrix(y_test1, y_predict1)
plot_confusion_matrix()
```



In [40]:
```
print(classification_report(y_test1, y_predict1))
plot_roc(y_test1, y_predict1)
```

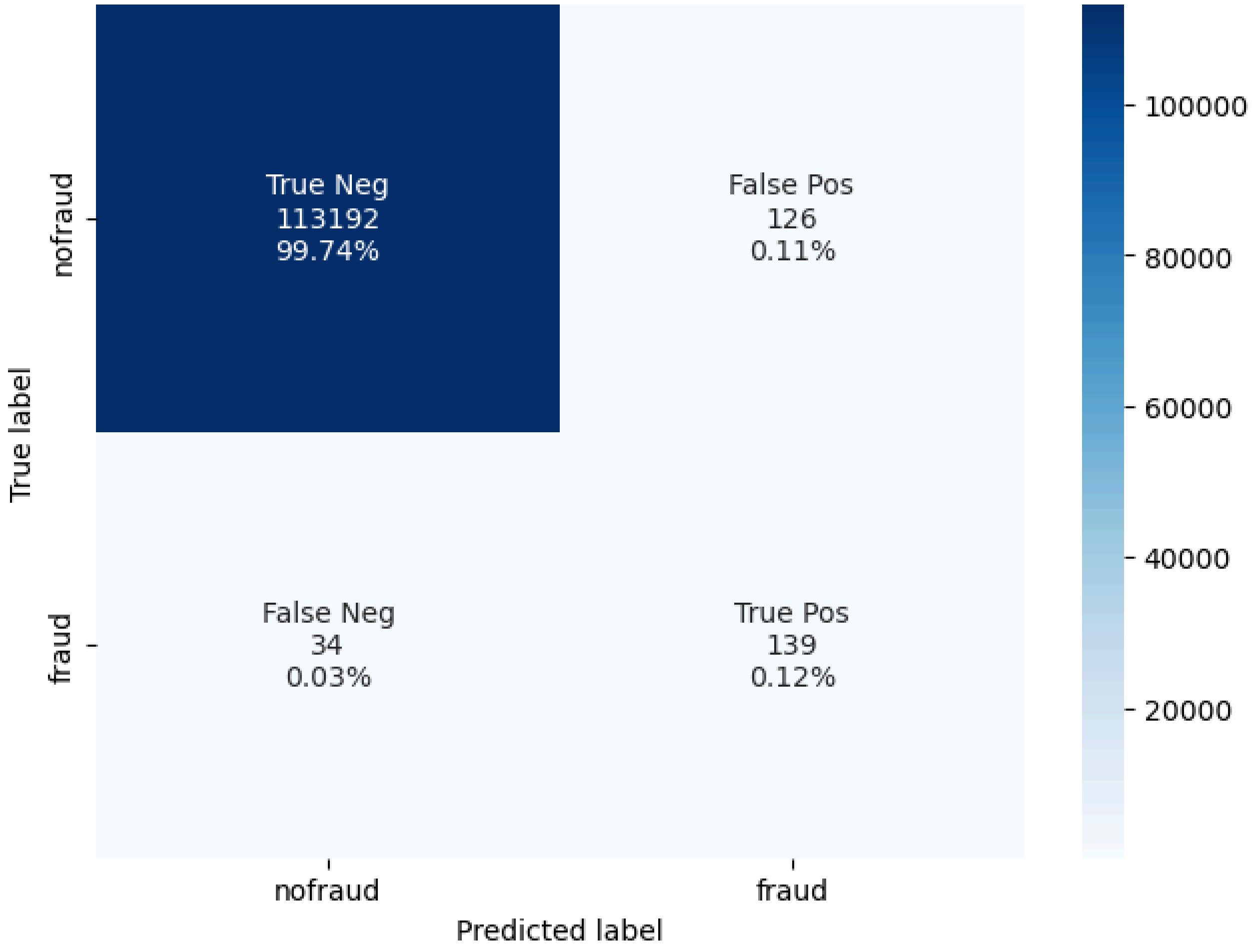
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    113318
           1       0.35      0.80      0.49       173

    accuracy                           1.00    113491
   macro avg       0.67      0.90      0.74    113491
weighted avg       1.00      1.00      1.00    113491
```



In [41]:
```
# Model based on ANOVA Score :
# Use the KNN classifier to fit data:
K_value=3
classifier = KNeighborsClassifier(n_neighbors = K_value, leaf_size = 1 , algorithm = 'auto')
classifier.fit(x_train2_smote, y_train2_smote)
# Predict y data with classifier:
y_predict2 = classifier.predict(x_test2)
show_metrics(y_test2,y_predict2)
```

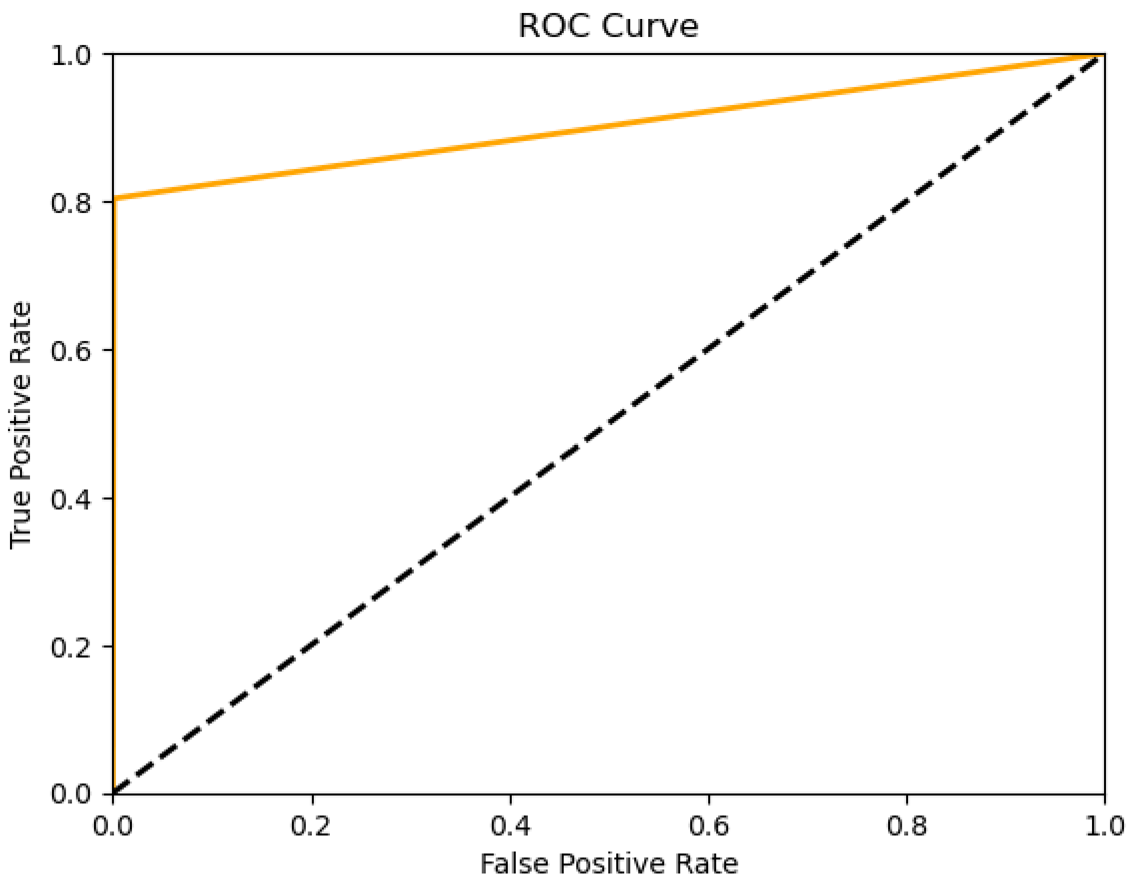
```
RMSE is                  : 0.037547349048O4265
Accuracy of model        : 0.9985901965794645
Precision Score          : 0.5245283018867924
F1 Score                 : 0.6347031963470319
Recall Score             : 0.8034682080924855
AUC Score                : 0.9011781464755126
Balanced Accuracy Score  : 0.9011781464755126
```

In [42]:
```
# Print results:
cm=metrics.confusion_matrix(y_test2, y_predict2)
plot_confusion_matrix()
```



In [43]:
```
print(classification_report(y_test2, y_predict2))
plot_roc(y_test2, y_predict2)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    113318
           1       0.52      0.80      0.63       173

    accuracy                           1.00    113491
   macro avg       0.76      0.90      0.82    113491
weighted avg       1.00      1.00      1.00    113491
```



Random Forest Classifier

In [44]:
```
seed = 0   # We set our random seed to zero for reproducibility
# Random Forest parameters
rf_params = {
    'n_jobs': -1,
```

```
    'n_estimators': 1000,
#     'warm_start': True,
    'max_features': 0.3,
    'max_depth': 4,
    'min_samples_leaf': 2,
    'max_features' : 'sqrt',
    'random_state' : seed,
    'verbose': 0
}
```

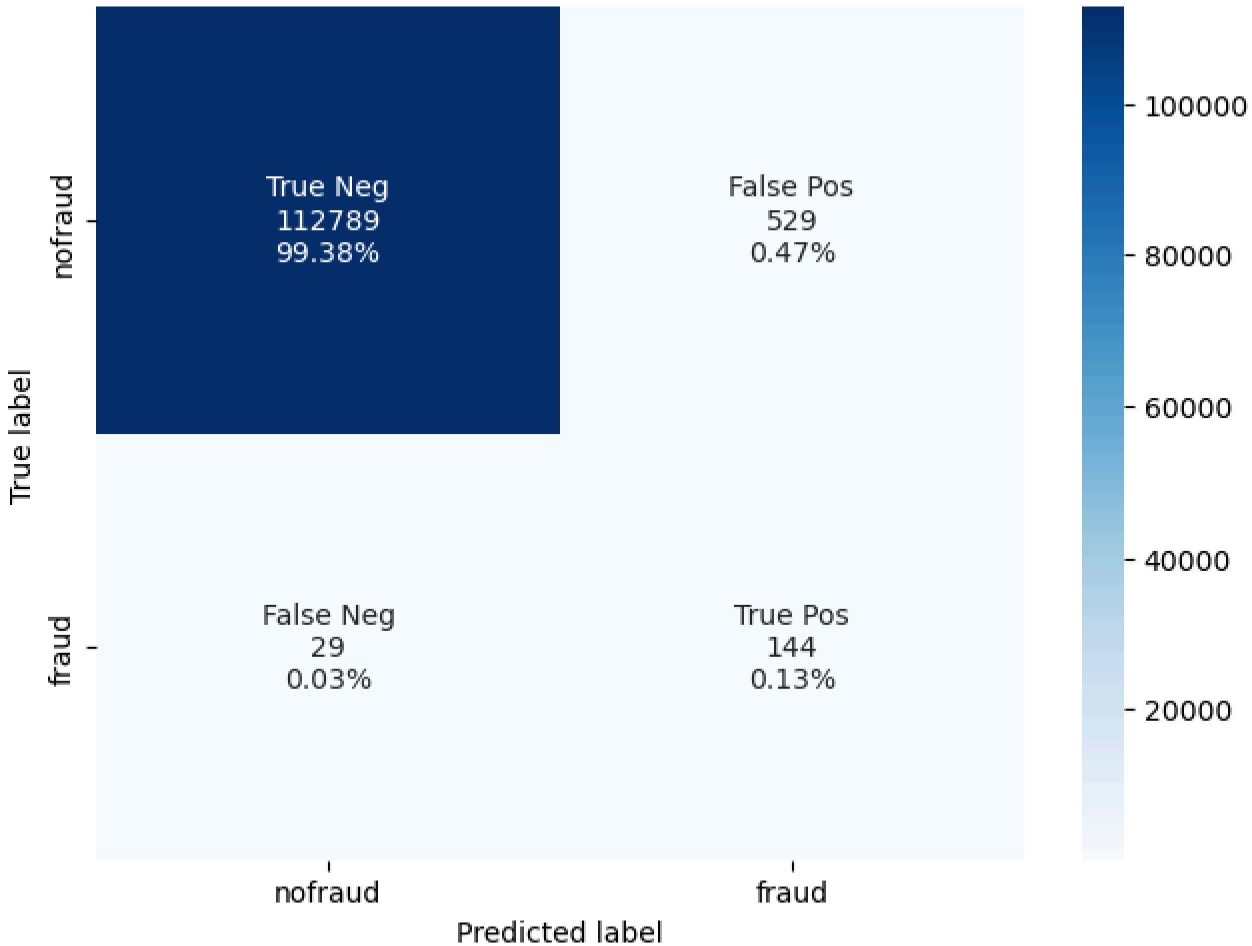
In [45]:

```
# Model based on Correlation Plot :
classifier = RandomForestClassifier(**rf_params)
classifier.fit(x_train1_smote, y_train1_smote)
# Predict y data with classifier:
y_predict1 = classifier.predict(x_test1)
show_metrics(y_test1,y_predict1)
```

```
RMSE is                  : 0.07011910887281532
Accuracy of model        : 0.9950833105708823
Precision Score          : 0.2139673105497771
F1 Score                 : 0.3404255319148936
Recall Score             : 0.8323699421965318
AUC Score                : 0.9138508317735338
Balanced Accuracy Score : 0.9138508317735338
```

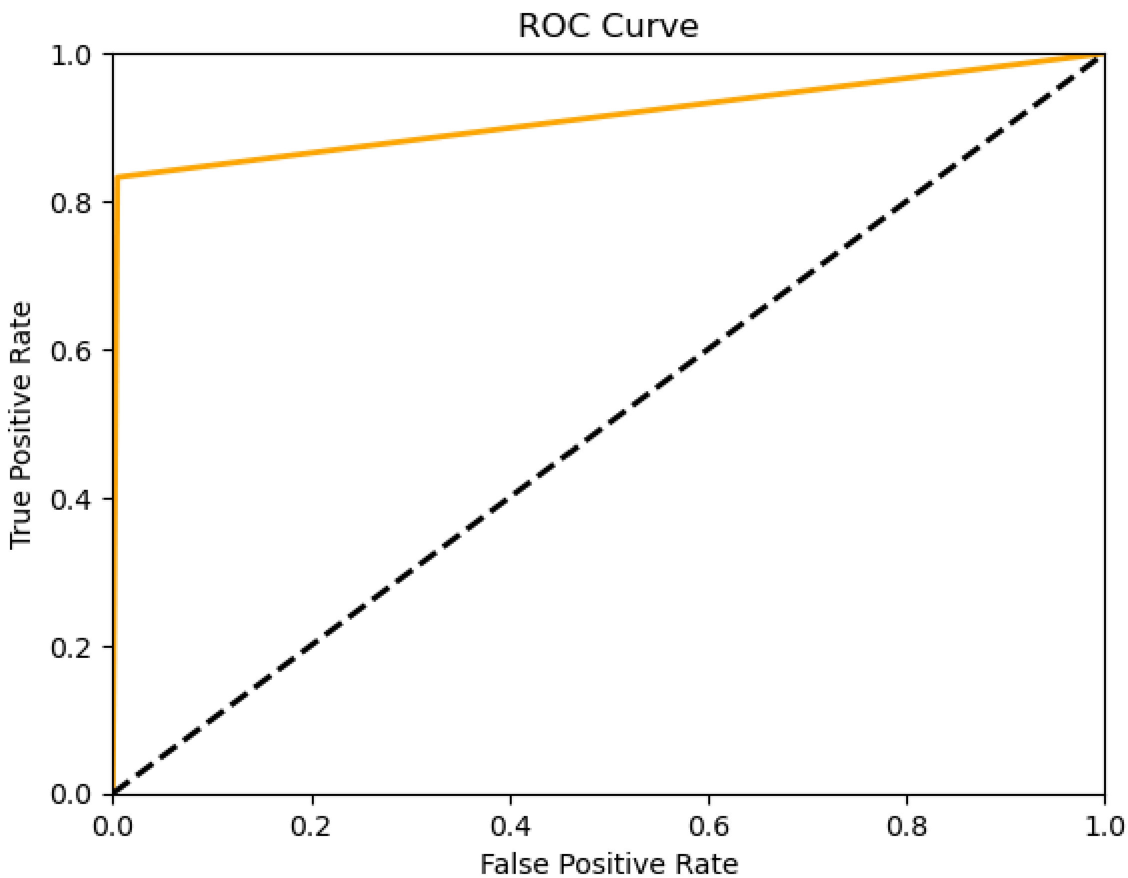
In [46]:

```
# Print results:
cm=metrics.confusion_matrix(y_test1, y_predict1)
plot_confusion_matrix()
```



In [47]:
```
print(classification_report(y_test1, y_predict1))
plot_roc(y_test1, y_predict1)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    113318
           1       0.21      0.83      0.34       173

    accuracy                           1.00    113491
   macro avg       0.61      0.91      0.67    113491
weighted avg       1.00      1.00      1.00    113491
```

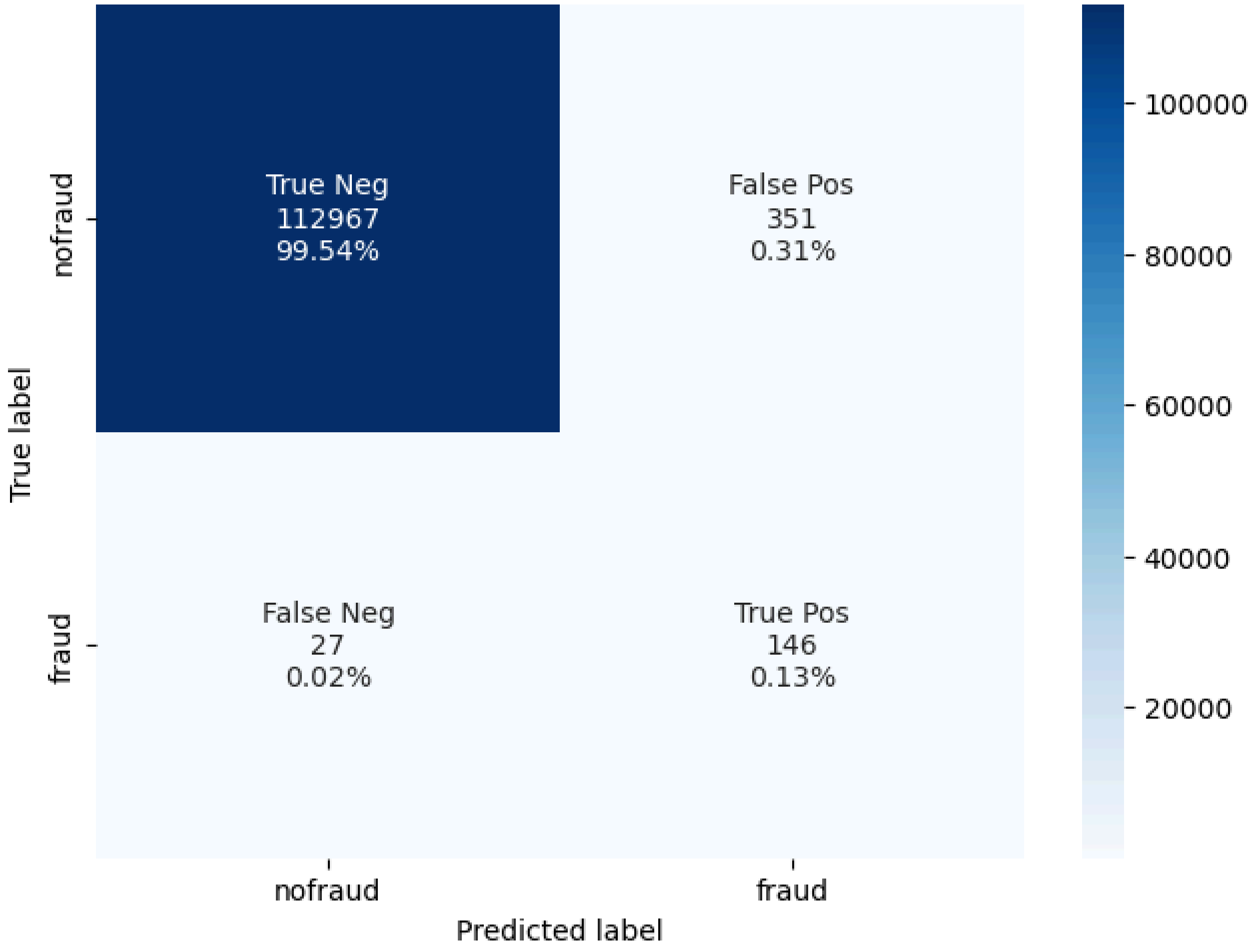


In [48]:

```
# Model based on ANOVA Score :
classifier = RandomForestClassifier(**rf_params)
classifier.fit(x_train2_smote, y_train2_smote)
# Predict y data with classifier:
y_predict2 = classifier.predict(x_test2)
show_metrics(y_test2,y_predict2)
```

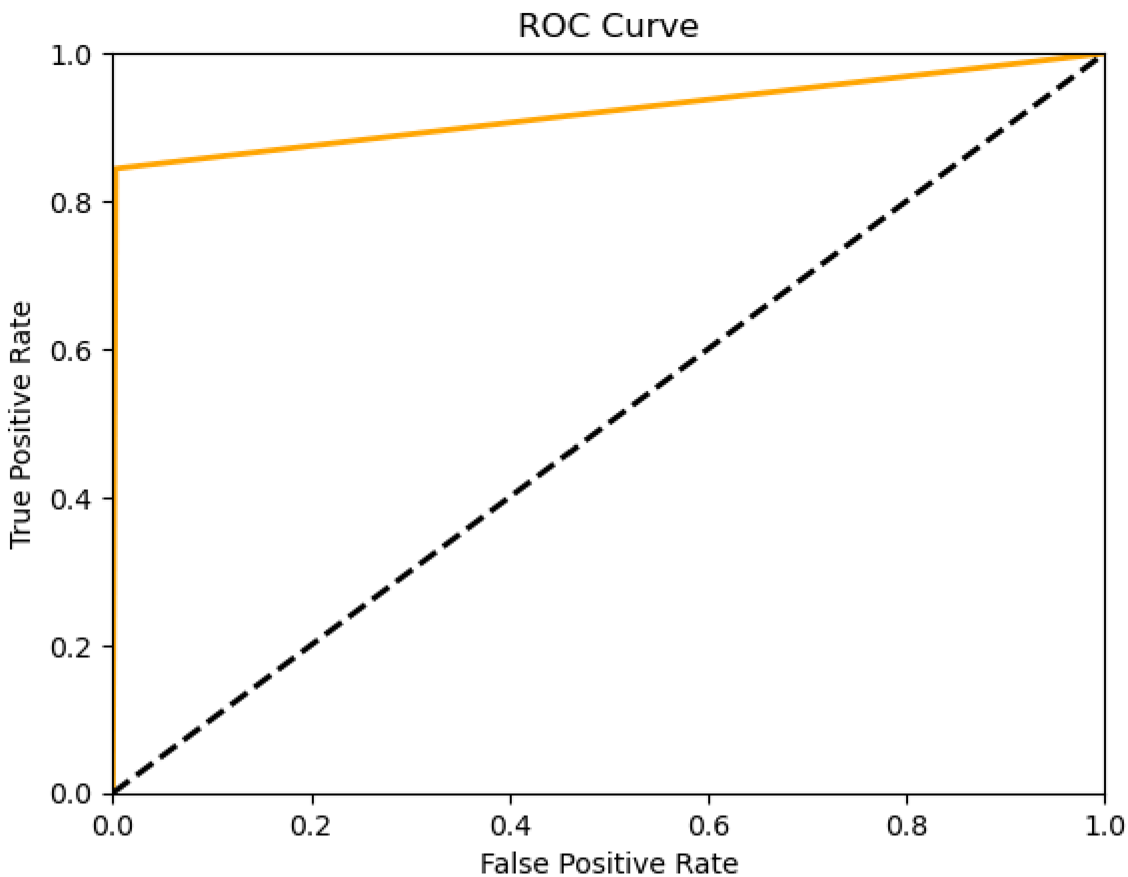
```
RMSE is                 : 0.05771187556313895
Accuracy of model       : 0.9966693394189847
Precision Score         : 0.2937625754527163
F1 Score                : 0.435820895522388
Recall Score            : 0.8439306358381503
AUC Score               : 0.9204165789720412
Balanced Accuracy Score : 0.9204165789720411
```

In [49]:
```
# Print results:
cm=metrics.confusion_matrix(y_test2, y_predict2)
plot_confusion_matrix()
```



In [50]:
```
print(classification_report(y_test2, y_predict2))
plot_roc(y_test2, y_predict2)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    113318
           1       0.29      0.84      0.44       173

    accuracy                           1.00    113491
   macro avg       0.65      0.92      0.72    113491
weighted avg       1.00      1.00      1.00    113491
```



Decision Tree Classifier

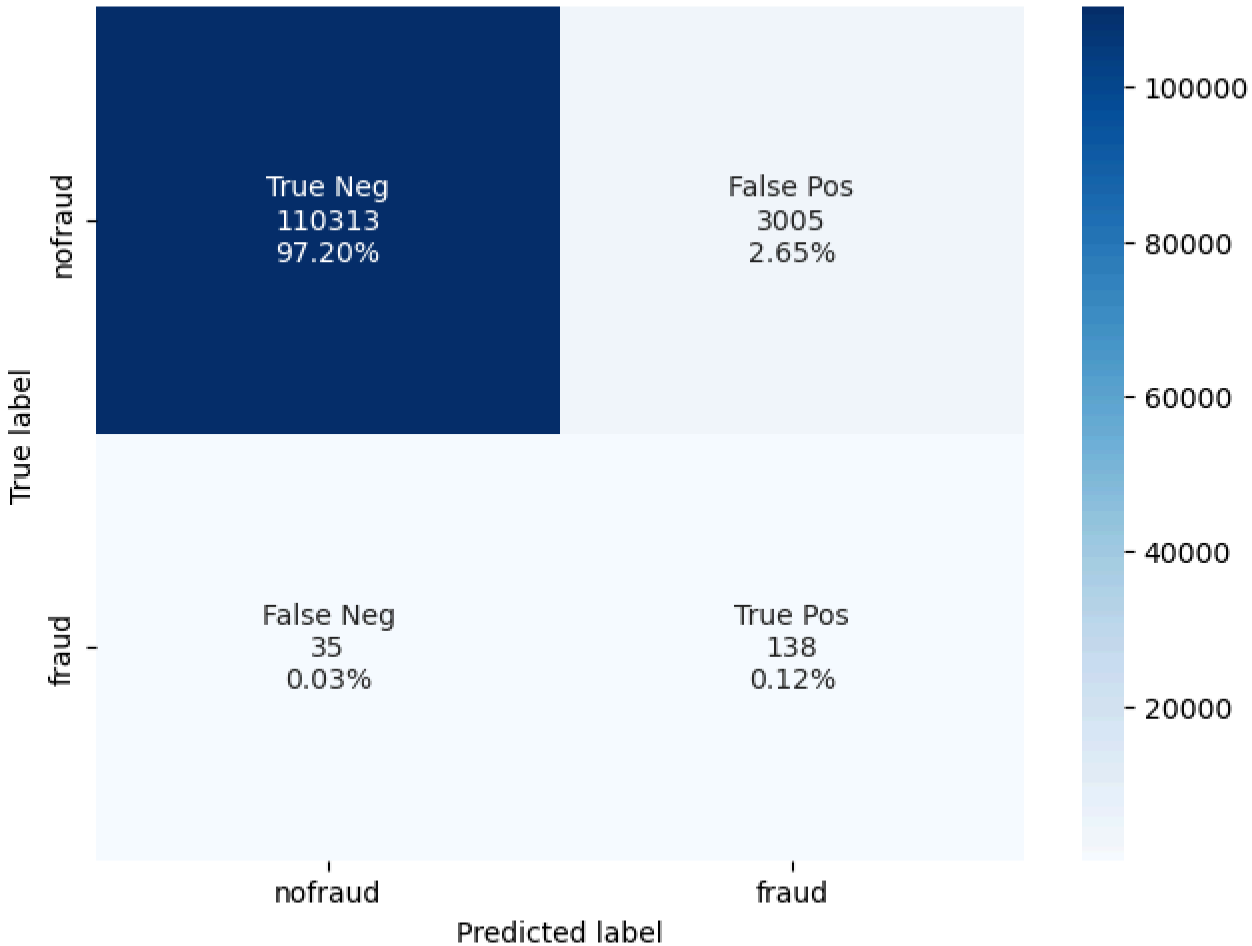
In [51]:
```
# Model based on Correlation Plot :
from sklearn.tree import DecisionTreeClassifier
# train model
classifier = DecisionTreeClassifier(max_depth = 10, random_state= 101, max_features =None , min_samples_leaf = 30)
```

```
classifier.fit(x_train1_smote, y_train1_smote)
# Predict y data with classifier:
y_predict1 = classifier.predict(x_test1)
show_metrics(y_test1,y_predict1)
```

```
RMSE is                  : 0.16366510009826601
Accuracy of model        : 0.9732137350098246
Precision Score          : 0.043907095132039456
F1 Score                 : 0.08323281061519903
Recall Score             : 0.7976878612716763
AUC Score                : 0.8855847838100912
Balanced Accuracy Score : 0.8855847838100912
```

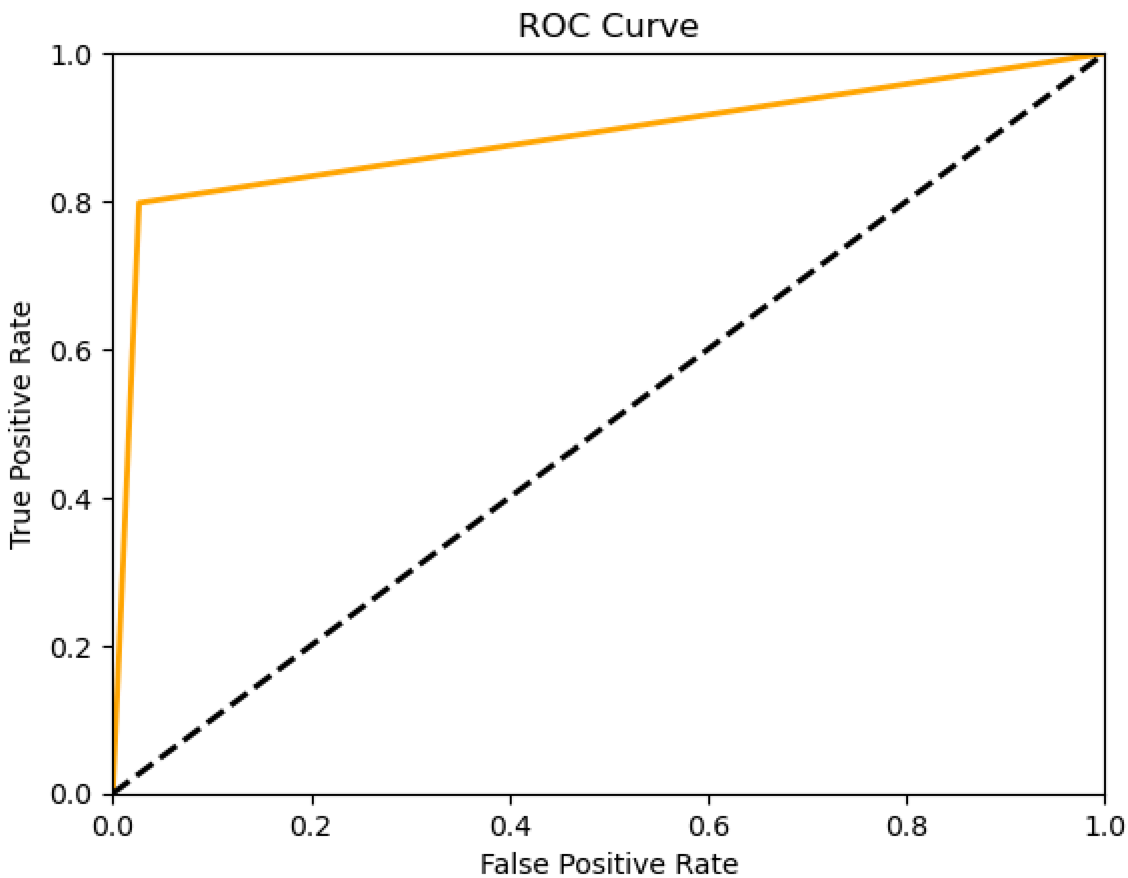
In [52]:

```
# Print results:
cm=metrics.confusion_matrix(y_test1, y_predict1)
plot_confusion_matrix()
```



In [53]:
```
print(classification_report(y_test1, y_predict1))
plot_roc(y_test1, y_predict1)
```

```
              precision    recall  f1-score   support

           0       1.00      0.97      0.99    113318
           1       0.04      0.80      0.08       173

    accuracy                           0.97    113491
   macro avg       0.52      0.89      0.53    113491
weighted avg       1.00      0.97      0.99    113491
```

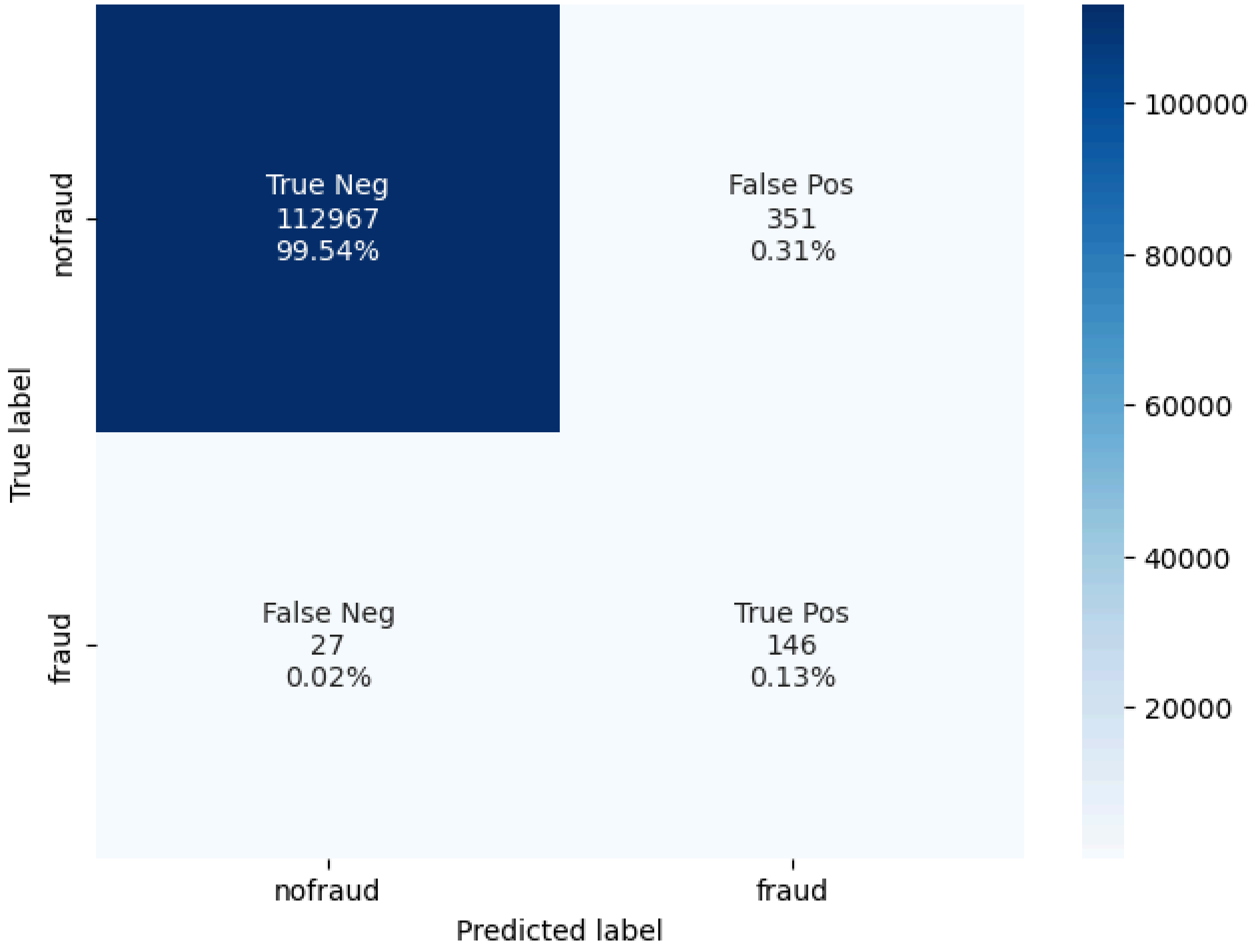


In [73]:

```
# Model based on ANOVA Score :
from sklearn.tree import DecisionTreeClassifier
# train model
classifier = DecisionTreeClassifier(max_depth = 10, random_state= 101, max_features =None , min_samples_leaf = 30)
classifier.fit(x_train2_smote, y_train2_smote)
# Predict y data with classifier:
y_predict2 = classifier.predict(x_test2)
show_metrics(y_test2,y_predict2)
```

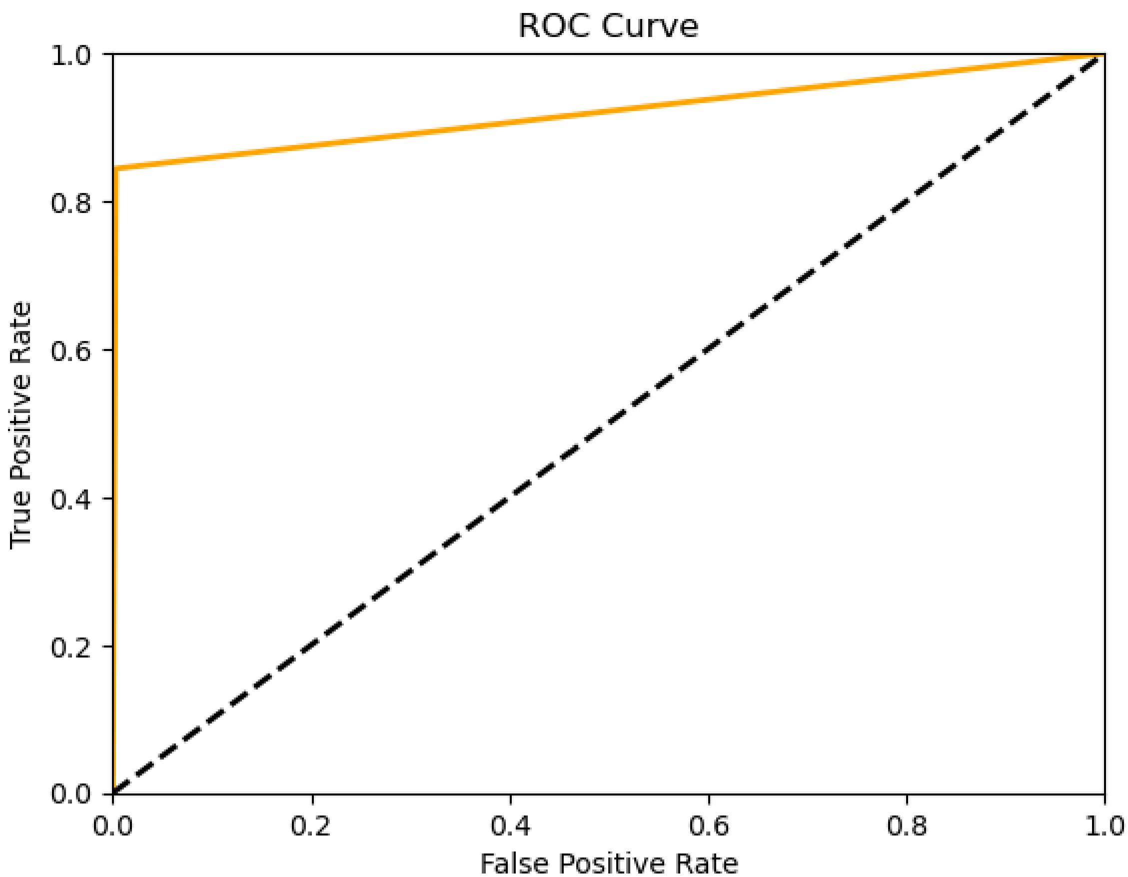
```
RMSE is                  : 0.15512440857023807
Accuracy of model        : 0.9759364178657338
Precision Score          : 0.04901269393511989
F1 Score                 : 0.09238949817215023
Recall Score             : 0.8034682080924855
AUC Score                : 0.8898339646156139
Balanced Accuracy Score : 0.8898339646156139
```

In [55]:
```
# Print results:
cm=metrics.confusion_matrix(y_test2, y_predict2)
plot_confusion_matrix()
```



In [56]:
```
print(classification_report(y_test2, y_predict2))
plot_roc(y_test2, y_predict2)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    113318
           1       0.29      0.84      0.44       173

    accuracy                           1.00    113491
   macro avg       0.65      0.92      0.72    113491
weighted avg       1.00      1.00      1.00    113491
```



XGBoost Model

In [57]:

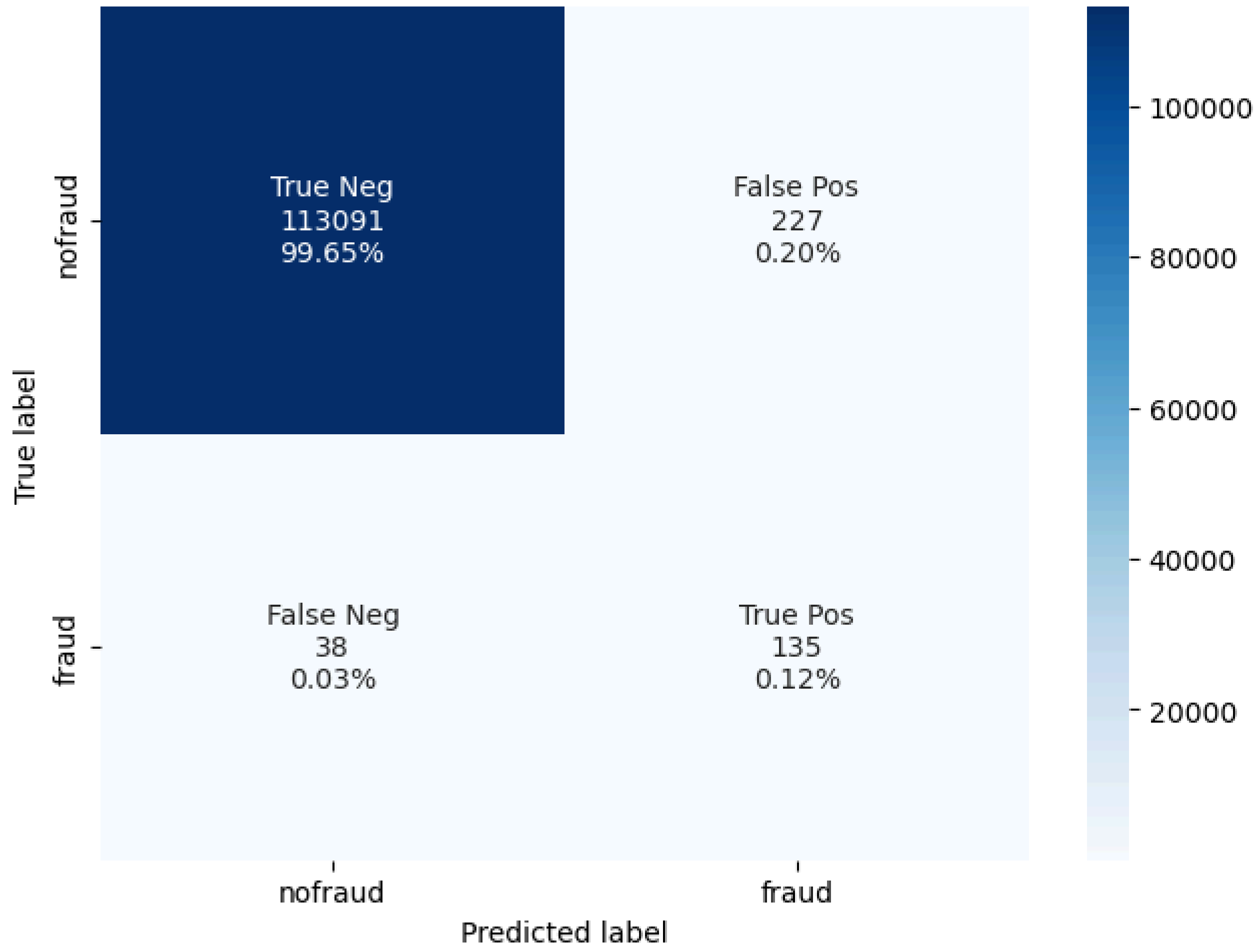
```
# Model based on Correlation Plot :
from xgboost import XGBClassifier

# train model
```

```python
classifier = XGBClassifier(objective = 'binary:logistic',eval_metric = 'logloss',seed = 42, use_label_encoder = False)
classifier.fit(x_train1_smote, y_train1_smote)
# Predict y data with classifier:
y_predict1 = classifier.predict(x_test1)
show_metrics(y_test1,y_predict1)
```

```
RMSE is                  : 0.04832170232164843
Accuracy of model        : 0.997665013084738
Precision Score          : 0.3729281767955801
F1 Score                 : 0.5046728971962616
Recall Score             : 0.7803468208092486
AUC Score                : 0.8891718043049754
Balanced Accuracy Score : 0.8891718043049754
```

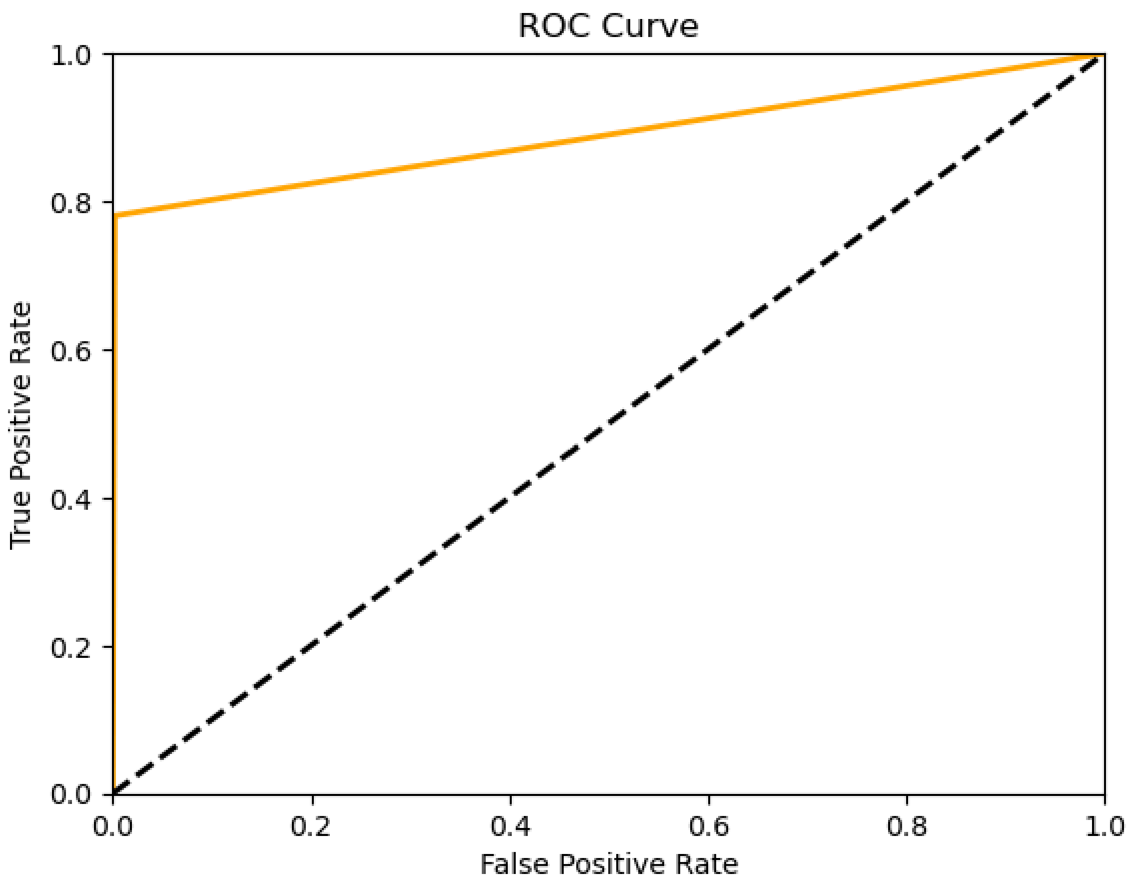
In [58]:

```python
# Print results:
cm=metrics.confusion_matrix(y_test1, y_predict1)
plot_confusion_matrix()
```



In [59]:
```
print(classification_report(y_test1, y_predict1))
plot_roc(y_test1, y_predict1)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    113318
           1       0.37      0.78      0.50       173

    accuracy                           1.00    113491
   macro avg       0.69      0.89      0.75    113491
weighted avg       1.00      1.00      1.00    113491
```



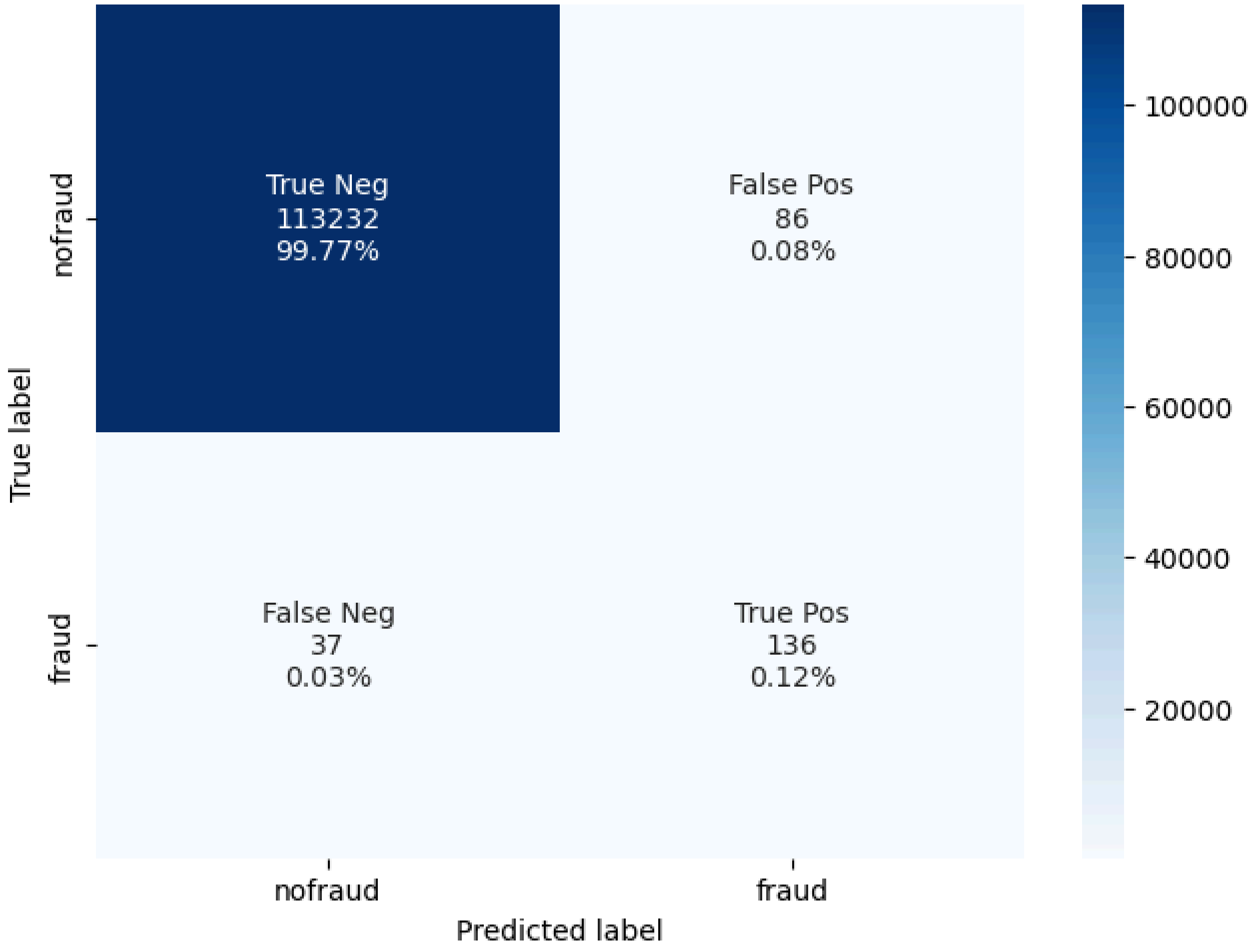
In [74]:

```
# Model based on ANOVA Score :

classifier = XGBClassifier(objective = 'binary:logistic',eval_metric = 'logloss',seed = 42, use_label_encoder = False)
classifier.fit(x_train2_smote, y_train2_smote)
# Predict y data with classifier:
y_predict2 = classifier.predict(x_test2)
show_metrics(y_test2,y_predict2)
```

```
RMSE is                  : 0.03292091097671362
Accuracy of model        : 0.9989162136204633
Precision Score          : 0.6126126126126126
F1 Score                 : 0.6886075949367089
Recall Score             : 0.7861271676300579
AUC Score                : 0.8926841207111973
Balanced Accuracy Score : 0.8926841207111973
```

In [75]:

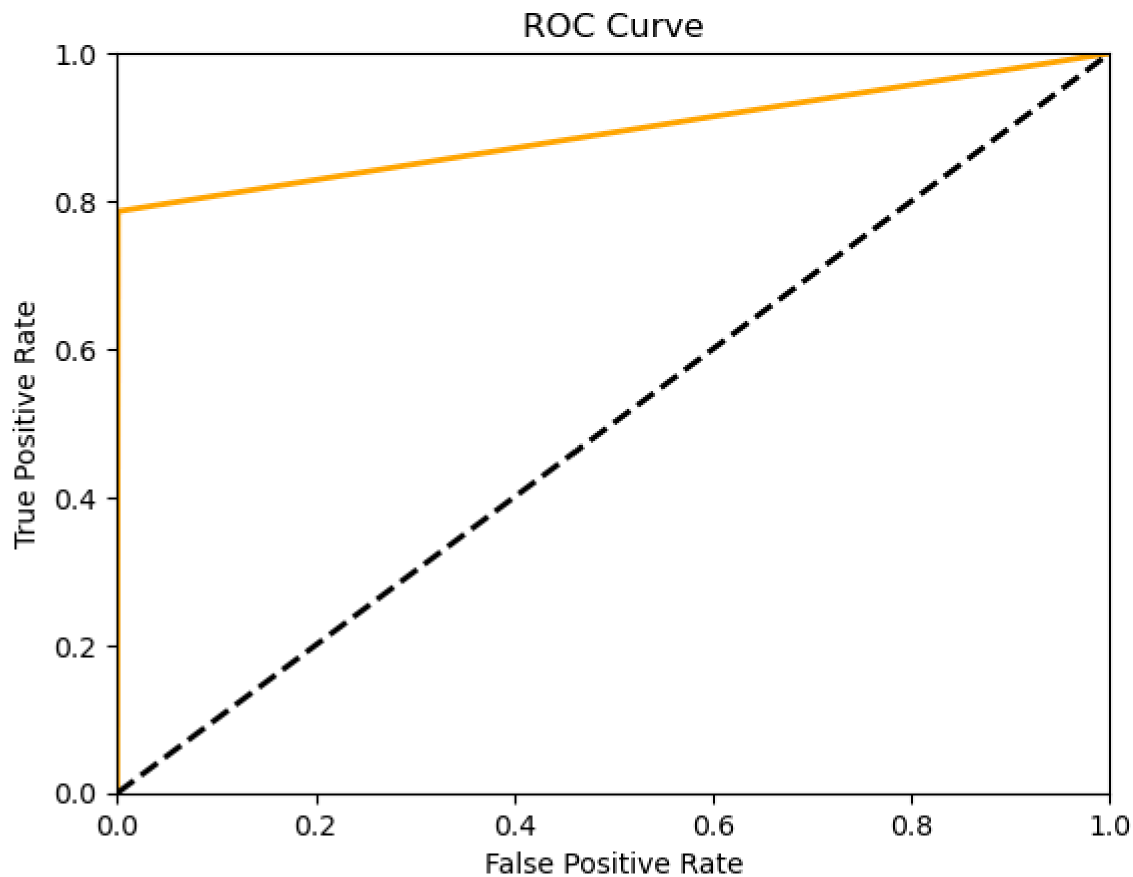
```
# Print results:
cm=metrics.confusion_matrix(y_test2, y_predict2)
plot_confusion_matrix()
```



In [76]:

```
print(classification_report(y_test2, y_predict2))
plot_roc(y_test2, y_predict2)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    113318
           1       0.61      0.79      0.69       173

    accuracy                           1.00    113491
   macro avg       0.81      0.89      0.84    113491
weighted avg       1.00      1.00      1.00    113491
```



Gradient Boosted Classifier

In [65]:
```
# Gradient Boosting Parameters
gb_params ={
    'n_estimators': 200,
    'max_features': 0.9,
```

```
    'learning_rate' : 0.25,
    'max_depth': 2,
    'min_samples_leaf': 2,
    'subsample': 1,
    'max_features' : 'sqrt',
    'random_state' : seed,
    'verbose': 0
}
```

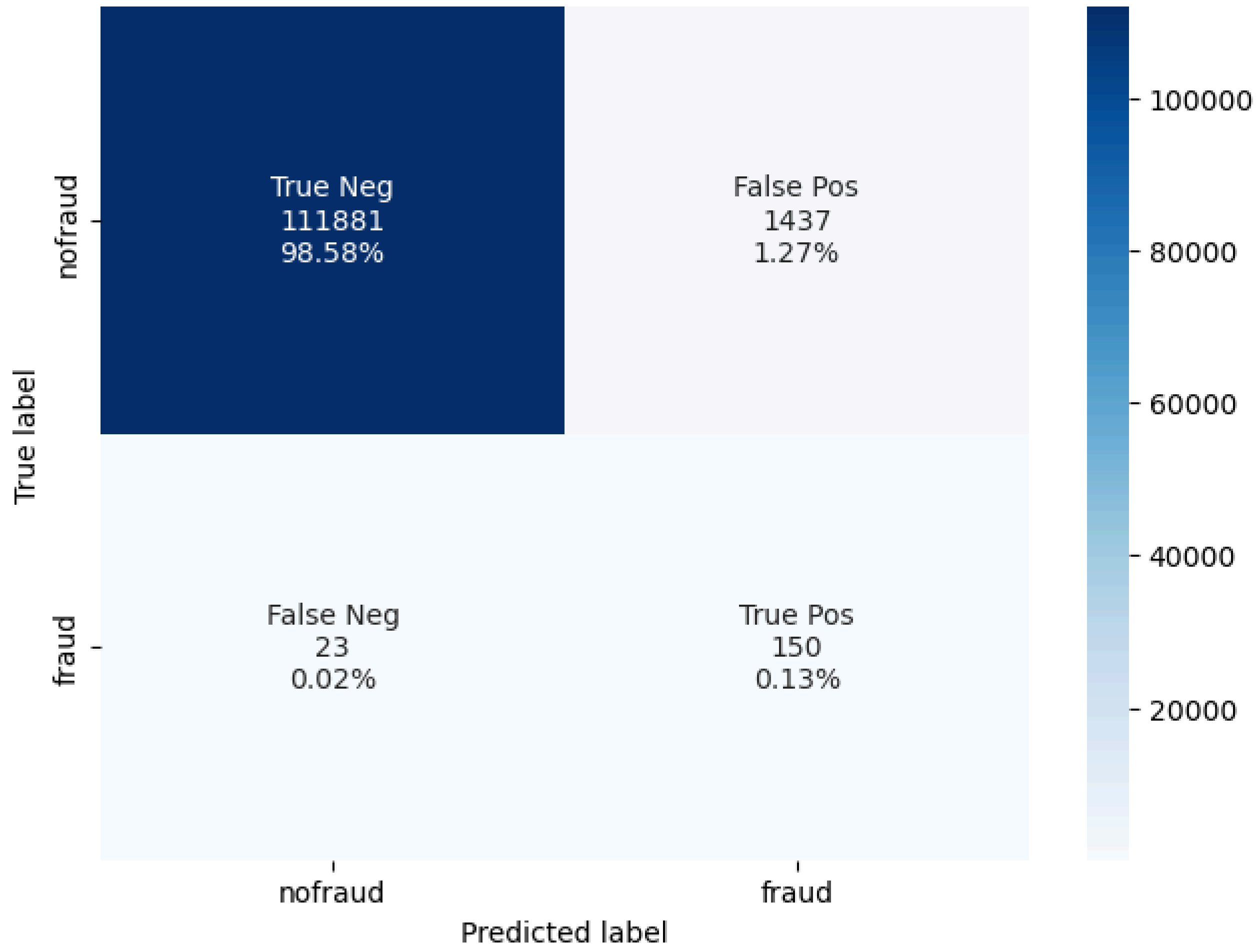
In [66]:

```
# Model based on Correlation Plot :
classifier =  GradientBoostingClassifier(**gb_params)
classifier.fit(x_train1_smote, y_train1_smote)
# Predict y data with classifier:
y_predict1 = classifier.predict(x_test1)
show_metrics(y_test1,y_predict1)
```

```
RMSE is                  : 0.11342158618352542
Accuracy of model        : 0.9871355437876131
Precision Score          : 0.0945179584120983
F1 Score                 : 0.17045454545454544
Recall Score             : 0.8670520231213873
AUC Score                : 0.9271854478373663
Balanced Accuracy Score : 0.9271854478373664
```

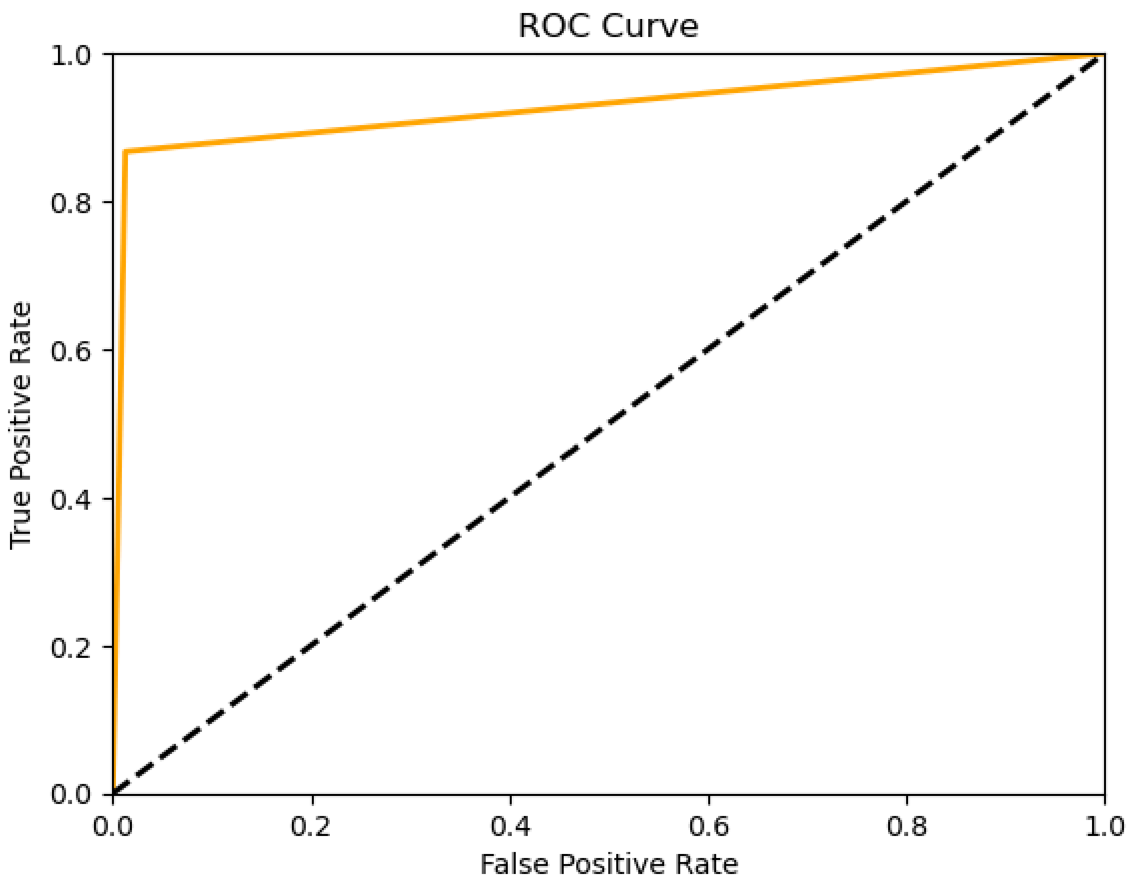
In [67]:

```
# Print results:
cm=metrics.confusion_matrix(y_test1, y_predict1)
plot_confusion_matrix()
```



In [68]:
```
print(classification_report(y_test1, y_predict1))
plot_roc(y_test1, y_predict1)
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99    113318
           1       0.09      0.87      0.17       173

    accuracy                           0.99    113491
   macro avg       0.55      0.93      0.58    113491
weighted avg       1.00      0.99      0.99    113491
```

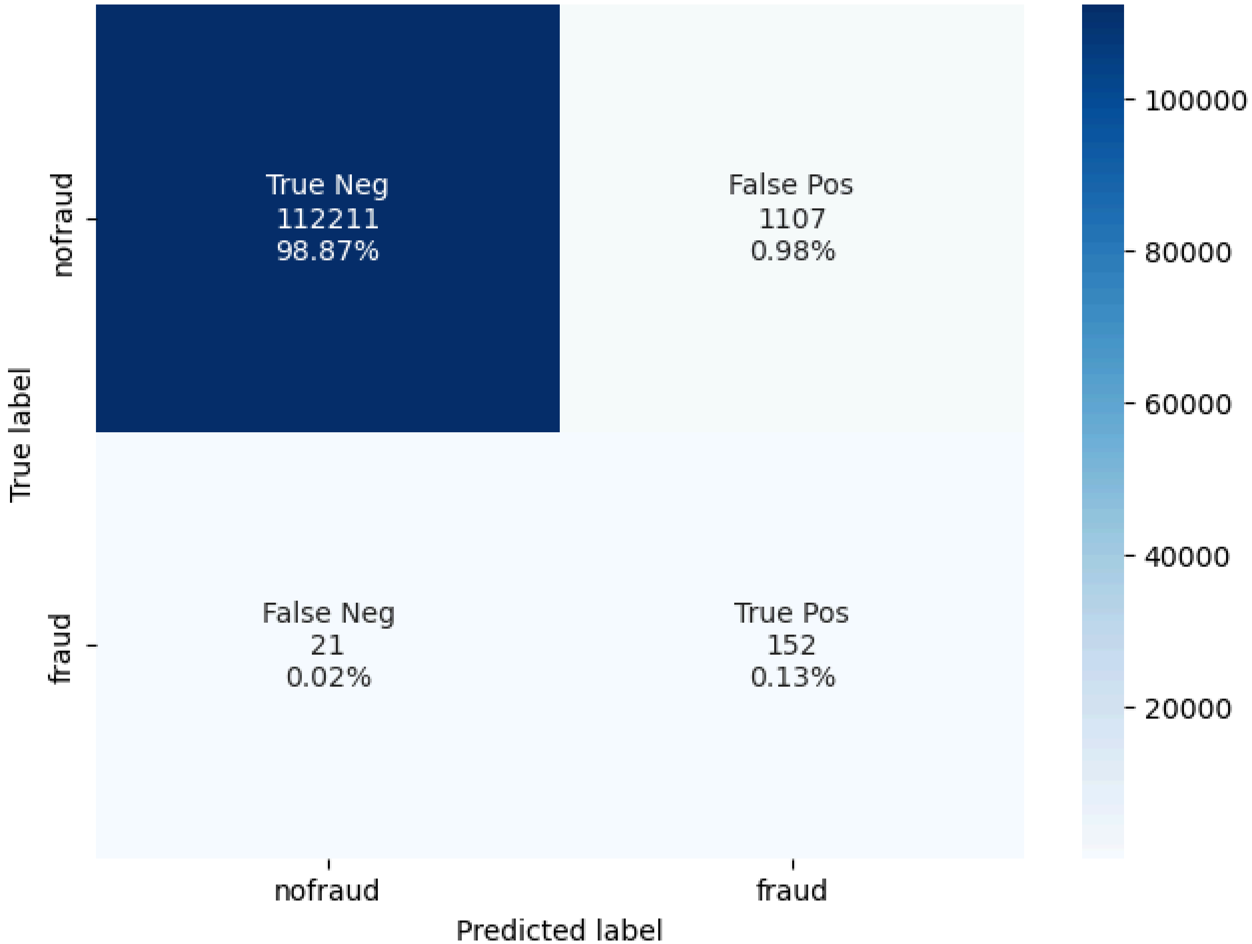


In [80]:

```
# Model based on ANOVA Score :
classifier =  GradientBoostingClassifier(**gb_params)
classifier.fit(x_train2_smote, y_train2_smote)
# Predict y data with classifier:
y_predict2 = classifier.predict(x_test2)
show_metrics(y_test2,y_predict2)
```

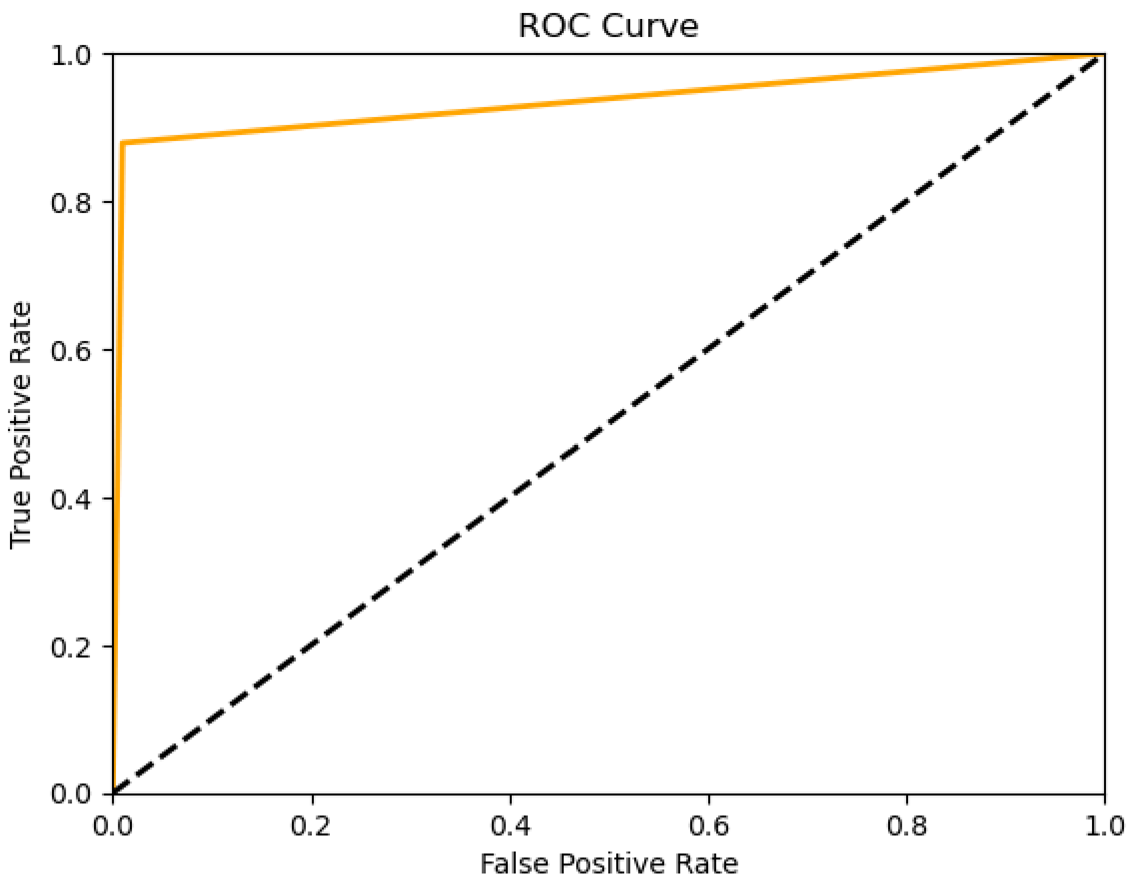
```
RMSE is                 : 0.09969510577142501
Accuracy of model       : 0.9900608858852243
Precision Score         : 0.12073073868149325
F1 Score                : 0.2122905027932961
Recall Score            : 0.8786127167630058
AUC Score               : 0.9344218740100878
Balanced Accuracy Score : 0.9344218740100879
```

In [81]:
```
# Print results:
cm=metrics.confusion_matrix(y_test2, y_predict2)
plot_confusion_matrix()
```



In [82]:
```
print(classification_report(y_test2, y_predict2))
plot_roc(y_test2, y_predict2)
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99    113318
           1       0.12      0.88      0.21       173

    accuracy                           0.99    113491
   macro avg       0.56      0.93      0.60    113491
weighted avg       1.00      0.99      0.99    113491
```



Comparing all the models based on Model Performance

In [78]:

```
# Model Results based on Correlation Plot :
comparison_frame1 = pd.DataFrame({'Model':['Test_accuracy                :',
                                          'Test_Precision_Score         :',
```

```
                                     'Test_F1_Score                :',
                                     'Test_Recall_Score            :',
                                     'Test_AUC_Score               :',
                                     'Test_Balanced_Accuracy_Score:'],
                                     'K-Nearest Neighbor(SMOTE)':[0.99742,0.34936,0.48591,0.79768,0.89770,0.89770]
                                     'Random Forest (SMOTE)':[0.99508,0.21396,0.34042,0.83236,0.91385,0.91385],
                                     'Decision Tree (SMOTE)':[0.97321,0.04390,0.08323,0.79768,0.88558,0.88558],
                                     'XGBoost (SMOTE)':[0.99766,0.37292,0.50467,0.78034,0.88917,0.88917],
                                     'Gradient Boosted Classifier (SMOTE)':[0.98713,0.09451,0.17045,0.86705,0.9271

comparison_frame1
```

Out[78]:

	Model	K-Nearest Neighbor(SMOTE)	Random Forest (SMOTE)	Decision Tree (SMOTE)	XGBoost (SMOTE)	Gradient Boosted Classifier (SMOTE)
0	Test_accuracy :	0.99742	0.99508	0.97321	0.99766	0.98713
1	Test_Precision_Score :	0.34936	0.21396	0.04390	0.37292	0.09451
2	Test_F1_Score :	0.48591	0.34042	0.08323	0.50467	0.17045
3	Test_Recall_Score :	0.79768	0.83236	0.79768	0.78034	0.86705
4	Test_AUC_Score :	0.89770	0.91385	0.88558	0.88917	0.92718
5	Test_Balanced_Accuracy_Score:	0.89770	0.91385	0.88558	0.88917	0.92718

In [83]:

```
# Model Results based on ANOVA Score :
comparison_frame2 = pd.DataFrame({'Model':['Test_accuracy                :',
                                     'Test_Precision_Score         :',
                                     'Test_F1_Score                :',
                                     'Test_Recall_Score            :',
                                     'Test_AUC_Score               :',
                                     'Test_Balanced_Accuracy_Score:'],
                                     'K-Nearest Neighbor(SMOTE)':[0.99859,0.52452,0.63470,0.80346,0.90117,0.90117]
                                     'Random Forest (SMOTE)':[0.99666,0.29376,0.43582,0.84393,0.92041,0.92041],
                                     'Decision Tree (SMOTE)':[0.97593,0.04901,0.09238,0.80346,0.88983,0.88983],
                                     'XGBoost (SMOTE)':[ 0.99891,0.61261,0.68860,0.78612,0.89268,0.89268],
                                     'Gradient Boosted Classifier (SMOTE)':[0.99006,0.12073,0.21229,0.87861,0.9344

comparison_frame2
```

Out[83]:

	Model	K-Nearest Neighbor(SMOTE)	Random Forest (SMOTE)	Decision Tree (SMOTE)	XGBoost (SMOTE)	Gradient Boosted Classifier (SMOTE)
0	Test_accuracy :	0.99859	0.99666	0.97593	0.99891	0.99006
1	Test_Precision_Score :	0.52452	0.29376	0.04901	0.61261	0.12073
2	Test_F1_Score :	0.63470	0.43582	0.09238	0.68860	0.21229
3	Test_Recall_Score :	0.80346	0.84393	0.80346	0.78612	0.87861
4	Test_AUC_Score :	0.90117	0.92041	0.88983	0.89268	0.93442
5	Test_Balanced_Accuracy_Score:	0.90117	0.92041	0.88983	0.89268	0.93442

Results interpretation :

SMOTE uses a nearest neighbors algorithm to generate new and synthetic data we used for training our model.

Decision Tree - Model accuracy and Precision is the lowest among the 5 Models.

K-Nearest Neighbor - Has good Accuracy Score, and 2nd highest F1 and Recall Scores. The False positive & False negative cases are higher than all models.

Random Forest - Accuracy score is good, however the model is not predicting the Fraud correctly (precision is low) .

XGBoost - Accuracy is highest as well as it is identifying the Fraud better with the Highest Precision Score). Also, it has high Recall & F1 Scores.

Gradient Boosted Classifier - Accuracy is high .While precision is low. It also has the highest Recall, F1 & AUC Scores.

Also, The Scores are higher with feature reduction done with ANOVA test than Correlation plot.

So, the best result is obtained byXGBoost Classifier after deploying SMOTE for the unbalanced dataset.

Conclusion

This is a great dataset to learn about binary classification problem with unbalanced data.

As the features are disguised, feature selection cannot be assisted based on the domain knowledge of the topic. Statistical tests hold the complete importance to select features for modeling.