CREDIT CARD FRUD DETECTION

In this project the challenge is to recognize fraudulent credit card transactions so that the customers of credit card companies are not charged for items that they did not purchase.

The dataset for this project is downloaded from https://www.kaggle.com/mlg-ulb/creditcardfraud

While going through the data we will find that the data is extremely unbalanced and we will use machine laerning algorithms and deal with the unbalanced

IMPORTING THE LIBRARIES

In [146]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

LOAD THE DATA INTO GOOGLE DRIVE

In [2]:

```
from google.colab import files
uploaded=files.upload()
```

Choose File **No file selected**

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

LOADING THE DATA

In [147]:

```
import io
data=pd.read_csv("/content/creditcard.csv")
```

EDA

View the data set given

In [148]:

```
data
```

Out[148]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	0.55
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.61
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.62

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	0.22
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	0.82
...	...	...	...	...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	4.356170	1.59
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	-0.975926	0.15
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	-0.484782	0.41
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	-0.399126	1.93
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	-0.915427	1.04

284807 rows × 31 columns

Shape of the data set

In [149]:

```
data.shape
```

Out[149]:

```
(284807, 31)
```

View the columns of the dataset

In [150]:

```
data.columns
```

Out[150]:

```
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')
```

View the top 6 rows

In [151]:

```
data.head()
```

Out[151]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	0.6178
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.0652
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.0660
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.1782
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538

View the last 6 rows

In [152]:

```
data.tail()
```

Out[152]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	4.356170	1.59
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	-0.975926	0.15
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	-0.484782	0.41
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	-0.399126	1.93
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	-0.915427	1.04

calculating some statistical data like percentile, mean and std of the numerical values

In [153]:

```
data.describe()
```

Out[153]:

	Time	V1	V2	V3	V4	V5	V6	V7	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2
mean	94813.859575	3.919560e-15	5.688174e-16	-8.769071e-15	2.782312e-15	-1.552563e-15	2.010663e-15	-1.694249e-15	
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	7
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2

Getting information about the data types

In [154]:

```
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
```

Checking for null values

In [155]:

```
data.isnull().sum()
```

Out[155]:

```
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
```

no missing data

Checking the number of fraud and non fraud cases

In [156]:

```
count_classes=data['Class'].value_counts()
count_classes
```

Out[156]:

```
0    284315
1       492
Name: Class, dtype: int64
```

In [157]:

```
not_fraud=data[data['Class']==0]
fraud=data[data['Class']==1]
```

In [158]:

```
not_fraud.shape,fraud.shape
```

Out[158]:

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

In [159]:

```
percentage_fraud=fraud.shape[0]/(not_fraud.shape[0]+fraud.shape[0])
```

In [160]:

```
percentage_fraud
```

Out[160]:

```
0.001727485630620034
```

We find that the number of frauds is very less when comapred to the non fraud cases so this data set is extermly unbalanced

PLOTTING FEW GRAPHS

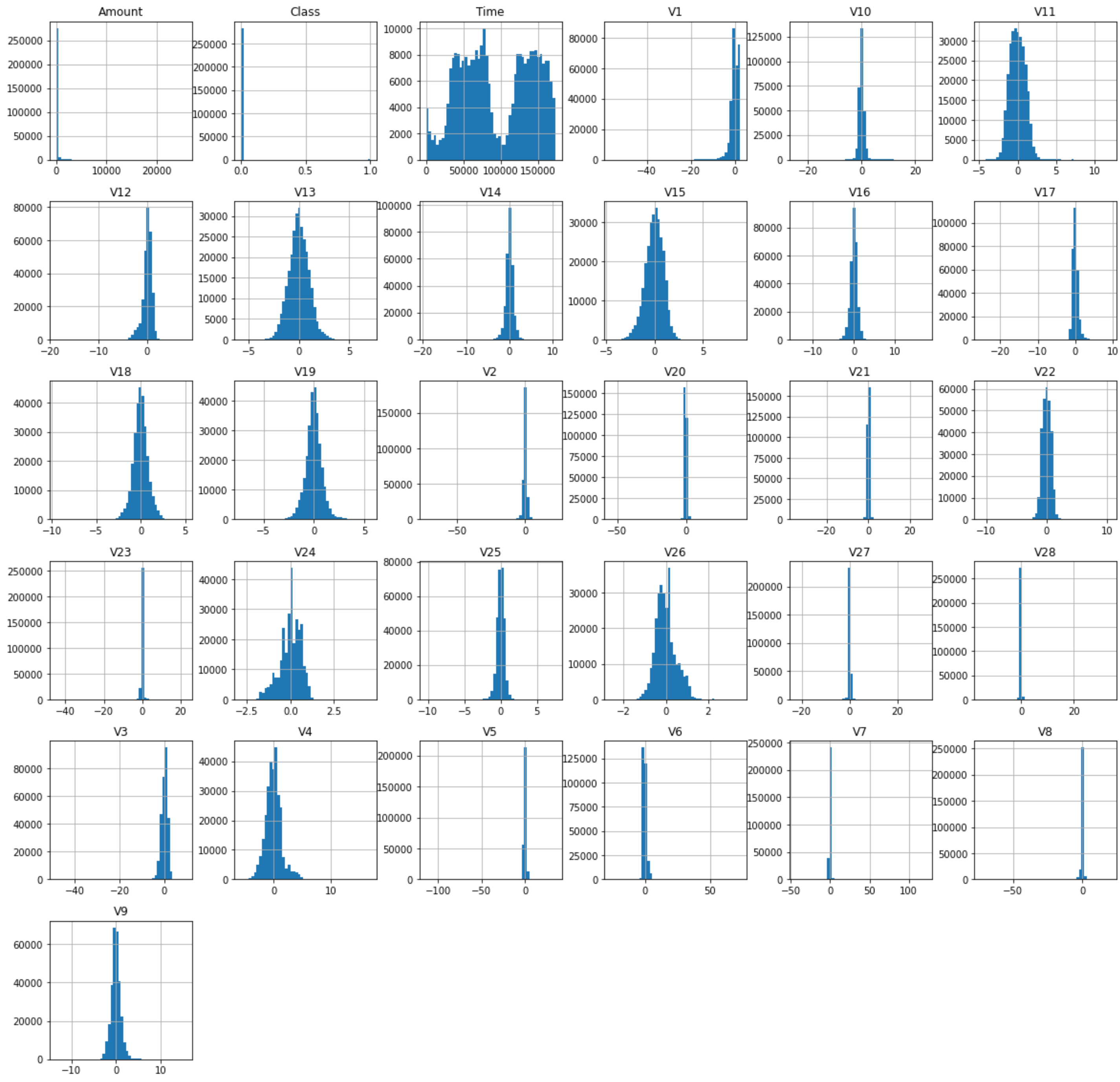
In [161]:

```
data.hist(bins=50,figsize=(20,20))
```

Out[161]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab5658cf8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab560fdd8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab55cc080>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab556edd8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab55a9080>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab555b2e8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab5504fd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab54be358>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab54be3c8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab549ba90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab544be10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab54091d0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab53bb550>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab53ed8d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab539ec50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab5352fd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab5310390>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab52c1710>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab5273a90>,
```

```
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab52a7e10>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab52671d0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab5216550>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab51ca8d0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab517bc50>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab5131fd0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab50ef390>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab511f710>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab50d1a90>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab5082e10>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab50421d0>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab4ff2550>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab50258d0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab4fd6c50>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab4f8cfd0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab4f4b390>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f5ab4efb710>]],
     dtype=object)
```



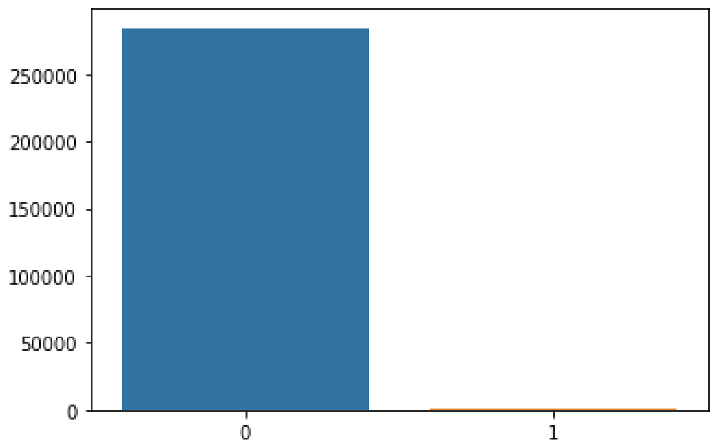
PLOTTING THE FRAUD AND NON FRAUD

In [162]:

```
sns.barplot(count_classes.index,count_classes.values)
```

Out[162]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f5ab3f22ac8>



PLOTTING THE CORRELATION MATRIX

In [163]:

```
corr = data.corr()
corr.style.background_gradient(cmap='coolwarm').set_precision(2)
```

Out[163]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20
Time	1.00	0.12	-0.01	-0.42	-0.11	0.17	-0.06	0.08	-0.04	-0.01	0.03	-0.25	0.12	-0.07	-0.10	-0.18	0.01	-0.07	0.09	0.03	-0.05
V1	0.12	1.00	0.00	-0.00	0.00	0.00	0.00	0.00	-0.00	0.00	0.00	0.00	0.00	-0.00	0.00	-0.00	0.00	-0.00	0.00	0.00	0.00
V2	-0.01	0.00	1.00	0.00	-0.00	-0.00	0.00	0.00	-0.00	-0.00	-0.00	0.00	-0.00	0.00	-0.00	0.00	0.00	-0.00	0.00	0.00	-0.00
V3	-0.42	-0.00	0.00	1.00	-0.00	-0.00	0.00	0.00	0.00	-0.00	0.00	-0.00	0.00	-0.00	0.00	0.00	0.00	0.00	0.00	0.00	-0.00
V4	-0.11	0.00	-0.00	-0.00	1.00	-0.00	-0.00	0.00	0.00	0.00	0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	0.00	-0.00	-0.00
V5	0.17	0.00	-0.00	-0.00	-0.00	1.00	0.00	-0.00	0.00	0.00	-0.00	0.00	0.00	-0.00	-0.00	-0.00	-0.00	0.00	0.00	-0.00	0.00
V6	-0.06	0.00	0.00	0.00	-0.00	0.00	1.00	0.00	-0.00	0.00	0.00	0.00	0.00	-0.00	0.00	-0.00	-0.00	0.00	0.00	0.00	0.00
V7	0.08	0.00	0.00	0.00	0.00	-0.00	0.00	1.00	-0.00	0.00	0.00	-0.00	0.00	-0.00	-0.00	0.00	0.00	0.00	-0.00	-0.00	0.00
V8	-0.04	-0.00	-0.00	0.00	0.00	0.00	-0.00	-0.00	1.00	0.00	0.00	0.00	-0.00	-0.00	-0.00	0.00	0.00	-0.00	-0.00	-0.00	-0.00
V9	-0.01	0.00	-0.00	-0.00	0.00	0.00	0.00	0.00	0.00	1.00	-0.00	0.00	-0.00	-0.00	0.00	-0.00	-0.00	0.00	0.00	0.00	-0.00
V10	0.03	0.00	-0.00	0.00	0.00	-0.00	0.00	0.00	0.00	-0.00	1.00	0.00	0.00	-0.00	0.00	0.00	-0.00	0.00	0.00	0.00	-0.00
V11	-0.25	0.00	0.00	-0.00	-0.00	0.00	0.00	-0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	-0.00	0.00	0.00	0.00	-0.00
V12	0.12	0.00	-0.00	0.00	-0.00	0.00	0.00	0.00	-0.00	-0.00	0.00	0.00	1.00	-0.00	0.00	0.00	0.00	-0.00	-0.00	0.00	0.00
V13	-0.07	-0.00	0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	0.00	-0.00	1.00	0.00	-0.00	0.00	-0.00	0.00	-0.00	0.00
V14	-0.10	0.00	-0.00	0.00	-0.00	-0.00	0.00	-0.00	-0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	-0.00	-0.00
V15	-0.18	-0.00	0.00	0.00	-0.00	-0.00	-0.00	0.00	0.00	-0.00	0.00	0.00	0.00	-0.00	0.00	1.00	0.00	0.00	0.00	-0.00	0.00

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20
V16	0.01	0.00	0.00	0.00	-0.00	-0.00	-0.00	0.00	0.00	-0.00	-0.00	-0.00	0.00	0.00	0.00	0.00	1.00	0.00	-0.00	0.00	0.00
V17	-0.07	-0.00	-0.00	0.00	-0.00	0.00	0.00	0.00	-0.00	0.00	0.00	0.00	-0.00	-0.00	0.00	0.00	0.00	1.00	-0.00	-0.00	-0.00
V18	0.09	0.00	0.00	0.00	0.00	0.00	0.00	-0.00	-0.00	0.00	0.00	0.00	-0.00	0.00	0.00	0.00	-0.00	-0.00	1.00	-0.00	-0.00
V19	0.03	0.00	0.00	0.00	-0.00	-0.00	0.00	-0.00	-0.00	0.00	0.00	0.00	0.00	-0.00	-0.00	-0.00	0.00	-0.00	-0.00	1.00	0.00
V20	-0.05	0.00	-0.00	-0.00	-0.00	0.00	0.00	0.00	-0.00	-0.00	-0.00	-0.00	0.00	0.00	-0.00	0.00	0.00	-0.00	-0.00	0.00	1.00
V21	0.04	-0.00	0.00	-0.00	-0.00	-0.00	-0.00	0.00	-0.00	0.00	0.00	-0.00	0.00	0.00	0.00	0.00	-0.00	-0.00	-0.00	0.00	-0.00
V22	0.14	0.00	0.00	0.00	0.00	0.00	-0.00	-0.00	0.00	0.00	-0.00	-0.00	-0.00	-0.00	0.00	-0.00	0.00	-0.00	-0.00	-0.00	0.00
V23	0.05	0.00	0.00	0.00	0.00	0.00	-0.00	0.00	0.00	0.00	0.00	0.00	0.00	-0.00	0.00	0.00	0.00	0.00	-0.00	0.00	0.00
V24	-0.02	0.00	-0.00	-0.00	0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	0.00	0.00	-0.00	0.00	-0.00	-0.00	-0.00	-0.00	0.00	0.00
V25	-0.23	-0.00	-0.00	0.00	0.00	0.00	0.00	0.00	-0.00	0.00	-0.00	-0.00	0.00	0.00	-0.00	0.00	-0.00	0.00	-0.00	0.00	-0.00
V26	-0.04	-0.00	0.00	0.00	-0.00	0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	0.00	-0.00	0.00	0.00	0.00	-0.00
V27	-0.01	0.00	-0.00	0.00	-0.00	0.00	-0.00	-0.00	0.00	-0.00	-0.00	-0.00	-0.00	-0.00	0.00	-0.00	0.00	0.00	0.00	-0.00	-0.00
V28	-0.01	0.00	-0.00	0.00	-0.00	-0.00	0.00	-0.00	-0.00	0.00	-0.00	-0.00	0.00	0.00	0.00	-0.00	0.00	-0.00	0.00	-0.00	-0.00
Amount	-0.01	-0.23	-0.53	-0.21	0.10	-0.39	0.22	0.40	-0.10	-0.04	-0.10	0.00	-0.01	0.01	0.03	-0.00	-0.00	0.01	0.04	-0.06	0.34
Class	-0.01	-0.10	0.09	-0.19	0.13	-0.09	-0.04	-0.19	0.02	-0.10	-0.22	0.15	-0.26	-0.00	-0.30	-0.00	-0.20	-0.33	-0.11	0.03	0.02

The percentage of the fraud and non farud values in the dataset

In [164]:

```
print('NO Frauds',percentage_fraud,'% of the dataset')
print('Frauds',1-percentage_fraud,'% of the dataset')
```

```
NO Frauds 0.001727485630620034 % of the dataset
Frauds 0.9982725143693799 % of the dataset
```

SPLITTING THE DATASET INTO TRAINING AND TESTING DATASET

We will split the dataset into training and testing datasets and using stratified k flod

IMPORTING THE LIBRARIES

In [165]:

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.model_selection import StratifiedKFold
```

We split the dataset into features and labels

In [166]:

```
X=data.drop('Class',axis=1)
```

```
y=data['Class']
```

Using the startified k flod we split the dataset into traning and testing data

In [167]:

```
sk=StratifiedKFold(n_splits=5,random_state=None,shuffle=False)
for train,test in sk.split(X,y):
  print("Train:",train,"Test:",test)
  X_train,X_test=X.iloc[train],X.iloc[test]
  y_train,y_test=y.iloc[train],y.iloc[test]
```

```
Train: [ 30473  30496  31002 ... 284804 284805 284806] Test: [     0      1      2 ... 5701
7 57018 57019]
Train: [      0       1       2 ... 284804 284805 284806] Test: [ 30473  30496  31002 ... 1
13964 113965 113966]
Train: [      0       1       2 ... 284804 284805 284806] Test: [ 81609  82400  83053 ... 1
70946 170947 170948]
Train: [      0       1       2 ... 284804 284805 284806] Test: [150654 150660 150661 ... 2
27866 227867 227868]
Train: [      0       1       2 ... 227866 227867 227868] Test: [212516 212644 213092 ... 2
84804 284805 284806]
```

In [168]:

```
X_train = X_train.values
X_test = X_test.values
y_train =y_train.values
y_test = y_test.values
```

The shape of the training and testing data

In [169]:

```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[169]:

```
((227846, 30), (56961, 30), (227846,), (56961,))
```

MODEL

As the dataset is very unbalanced we would be using IsolationForest and LocalOutlierFactor machine learning algorithms

In [170]:

```
from sklearn.metrics import classification_report,accuracy_score
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.svm import OneClassSVM
from pylab import rcParams
```

In [171]:

```
import scipy
state = np.random.RandomState(42)
```

Isolation Forest

In [173]:

```
IF=IsolationForest(n_estimators=100, max_samples=len(X_train),
                                       contamination=percentage_fraud,random_state=state
, verbose=0)
```

In [174]:

```
IF.fit(X_train,y_train)
```

Out[174]:

```
IsolationForest(behaviour='deprecated', bootstrap=False,
                contamination=0.0017274856306200344, max_features=1.0,
                max_samples=227846, n_estimators=100, n_jobs=None,
                random_state=RandomState(MT19937) at 0x7F5AB99C7DB0, verbose=0,
                warm_start=False)
```

Predicting the values for testing data

In [178]:

```
y_if_pred=IF.predict(X_test)
```

In [179]:

```
y_if_pred[y_if_pred==1]=0
y_if_pred[y_if_pred==-1]=1
```

Counting the values of values which were predicted wrongly

In [180]:

```
n_if_errors = (y_if_pred != y_test).sum()
```

SUMMARY OF THE MODEL

In [181]:

```
print("The IsolationForest has {} errors".format(n_errors))
print("Accuracy Score :")
print(accuracy_score(y_test,ys_pred))
print("Classification Report :")
print(classification_report(y_test,y_if_pred))
```

```
The IsolationForest has 152 errors
Accuracy Score :
0.9973315075222691
Classification Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56863
           1       0.14      0.11      0.13        98

    accuracy                           1.00     56961
   macro avg       0.57      0.56      0.56     56961
weighted avg       1.00      1.00      1.00     56961
```

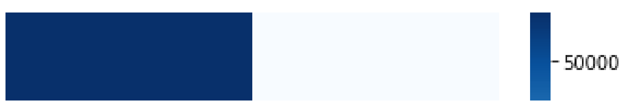
CONFUSION MATRIX

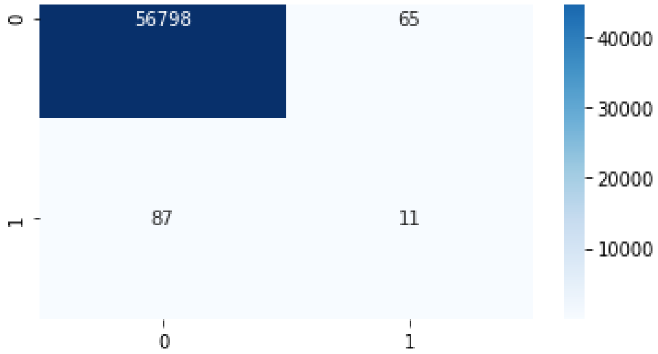
In [182]:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, ys_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
```

Out[182]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5ab3b36630>
```





Local Outlier Factor

In [183]:

```
LOF=LocalOutlierFactor(n_neighbors=20, algorithm='auto',
                                             leaf_size=30, metric='minkowski',
                                             p=2, metric_params=None, contamination=per
centage_fraud)
```

In [184]:

```
LOF.fit(X_train,y_train)
```

Out[184]:

```
LocalOutlierFactor(algorithm='auto', contamination=0.0017274856306200034,
                   leaf_size=30, metric='minkowski', metric_params=None,
                   n_jobs=None, n_neighbors=20, novelty=False, p=2)
```

Predicting the values for testing data

In [186]:

```
y_lof_pred=LOF.fit_predict(X_test)
```

In [187]:

```
y_lof_pred[y_lof_pred==1]=0
y_lof_pred[y_lof_pred==-1]=1
```

Counting the values of values which were predicted wrongly

In [188]:

```
na_errors = (y_lof_pred != y_test).sum()
```

SUMMARY OF THE MODEL

In [189]:

```
print("The LocalOutlierFactor has {} errors".format(na_errors))
print("Accuracy Score :")
print(accuracy_score(y_test,ya_pred))
print("Classification Report :")
print(classification_report(y_test,y_lof_pred))
```

```
The LocalOutlierFactor has 163 errors
Accuracy Score :
0.9971383929355173
Classification Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56863
           1       0.17      0.17      0.17        98
```

```
    accuracy                           1.00     56961
   macro avg       0.59      0.59      0.59     56961
weighted avg       1.00      1.00      1.00     56961
```

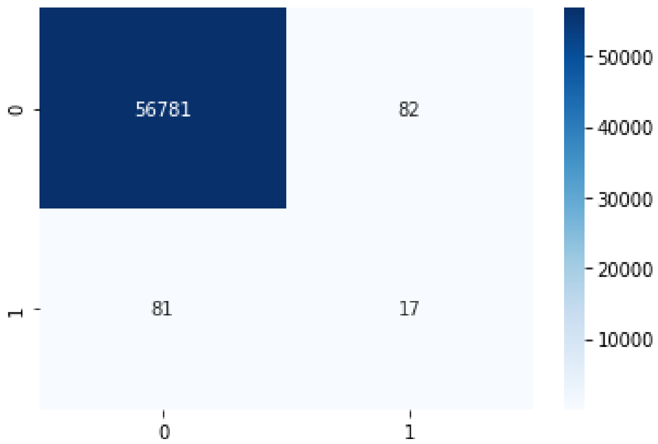
CONFUSION MATRIX

In [143]:

```python
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, ya_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
```

Out[143]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5ab5766f60>
```



Isolation forest detected 152 errors versus Lcal outlier Factor 163 errors

Isolation forest has a accuracy of 0.997331507522691 and Lcal outlier Factor has 0.9971383929355173

In [ ]:

```python

```