In [1]:

```
import numpy as np # for linear algebra
import pandas as pd

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

In [2]:

```
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

In [3]:

```
df=pd.read_csv('creditcard.csv')
df.head(10)
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.3
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.2
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.5
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.3
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.8
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.5
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.4
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.6
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.3
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-0.7

10 rows × 31 columns

In [4]:

```
df.tail(10)
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	
284797	172782.0	-0.241923	0.712247	0.399806	-0.463406	0.244531	-1.343668	0.929369	-0.2
284798	172782.0	0.219529	0.881246	-0.635891	0.960928	-0.152971	-1.014307	0.427126	0.1
284799	172783.0	-1.775135	-0.004235	1.189786	0.331096	1.196063	5.519980	-1.518185	2.0
284800	172784.0	2.039560	-0.175233	-1.196825	0.234580	-0.008713	-0.726571	0.017050	-0.1
284801	172785.0	0.120316	0.931005	-0.546012	-0.745097	1.130314	-0.235973	0.812722	0.1
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.3
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.2
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.7
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.6
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.4

10 rows × 31 columns

In [5]:

```
df.shape
```

Out[5]:

```
(284807, 31)
```

In [6]:

```
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
```

In [7]:

```
df["Class"].value_counts()
```

Out[7]:

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

In [8]:

```
df = df.drop(['Time'],axis=1)
df.head()
```

Out[8]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739

5 rows × 30 columns

In [9]:

```
df.shape
```

Out[9]:

```
(284807, 30)
```

In [10]:

```
df.duplicated().any()
```

Out[10]:

```
True
```

In [11]:

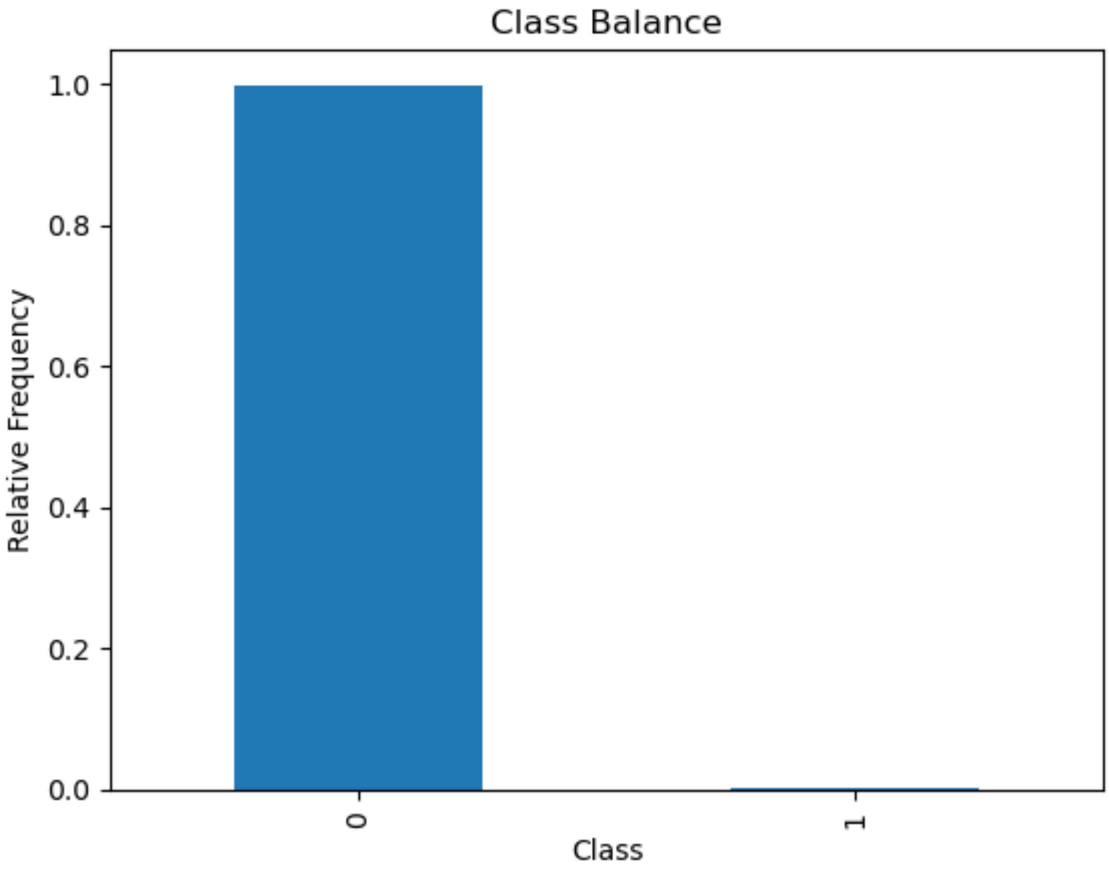
```
df = df.drop_duplicates()
df.shape
```

Out[11]:

```
(275663, 30)
```

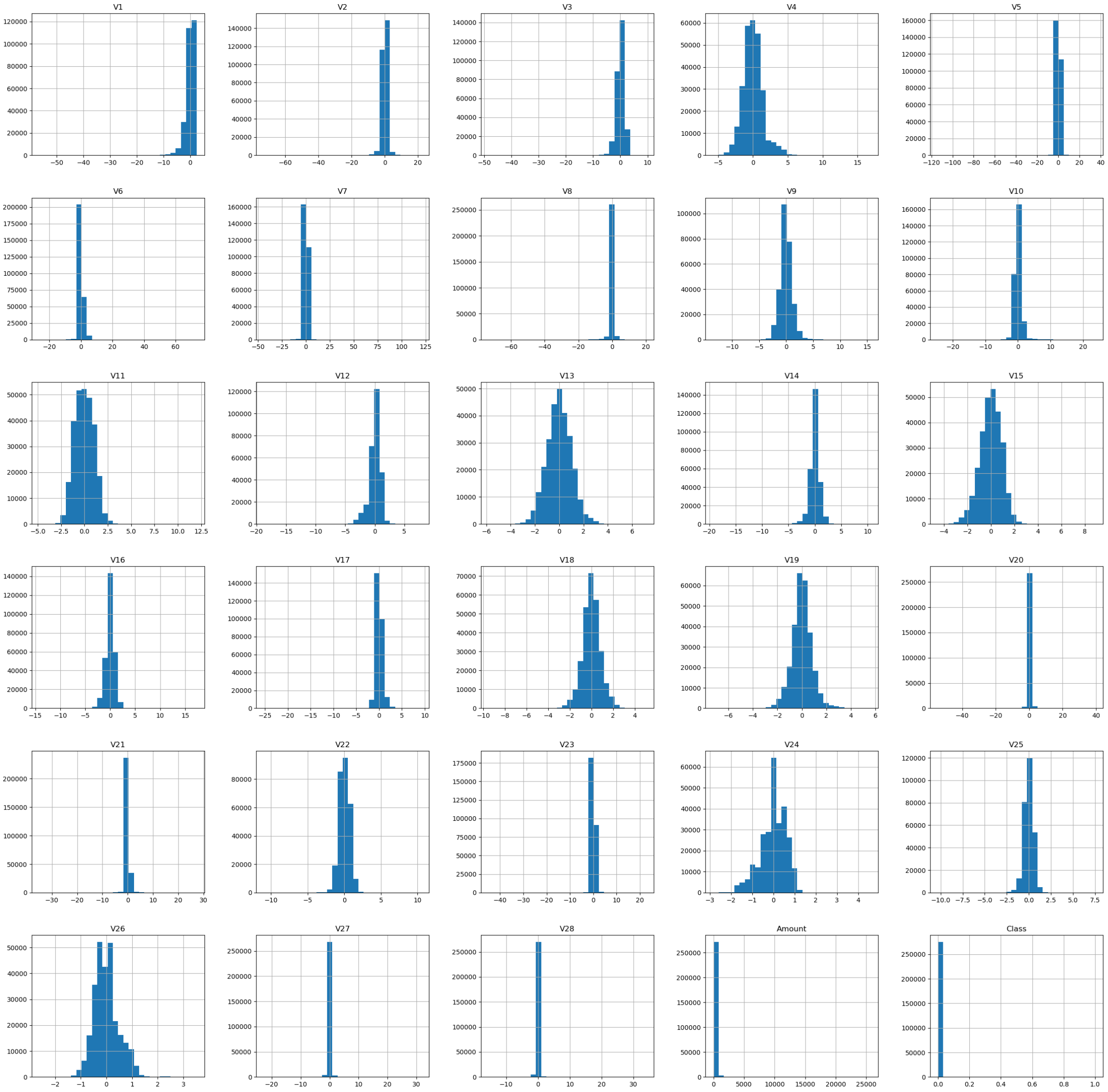
In [12]:

```
df["Class"].value_counts(normalize=True).plot(
    kind="bar", xlabel="Class", ylabel="Relative Frequency", title="Class Balance"
);
```



```
In [13]: df.hist(bins=30, figsize=(30, 30))
```

```
Out[13]: array([[<Axes: title={'center': 'V1'}>, <Axes: title={'center': 'V2'}>,
                 <Axes: title={'center': 'V3'}>, <Axes: title={'center': 'V4'}>,
                 <Axes: title={'center': 'V5'}>],
                [<Axes: title={'center': 'V6'}>, <Axes: title={'center': 'V7'}>,
                 <Axes: title={'center': 'V8'}>, <Axes: title={'center': 'V9'}>,
                 <Axes: title={'center': 'V10'}>],
                [<Axes: title={'center': 'V11'}>, <Axes: title={'center': 'V12'}>,
                 <Axes: title={'center': 'V13'}>, <Axes: title={'center': 'V14'}>,
                 <Axes: title={'center': 'V15'}>],
                [<Axes: title={'center': 'V16'}>, <Axes: title={'center': 'V17'}>,
                 <Axes: title={'center': 'V18'}>, <Axes: title={'center': 'V19'}>,
                 <Axes: title={'center': 'V20'}>],
                [<Axes: title={'center': 'V21'}>, <Axes: title={'center': 'V22'}>,
                 <Axes: title={'center': 'V23'}>, <Axes: title={'center': 'V24'}>,
                 <Axes: title={'center': 'V25'}>],
                [<Axes: title={'center': 'V26'}>, <Axes: title={'center': 'V27'}>,
                 <Axes: title={'center': 'V28'}>,
                 <Axes: title={'center': 'Amount'}>,
                 <Axes: title={'center': 'Class'}>]], dtype=object)
```



In [14]:
```
df.describe()
```

Out[14]:

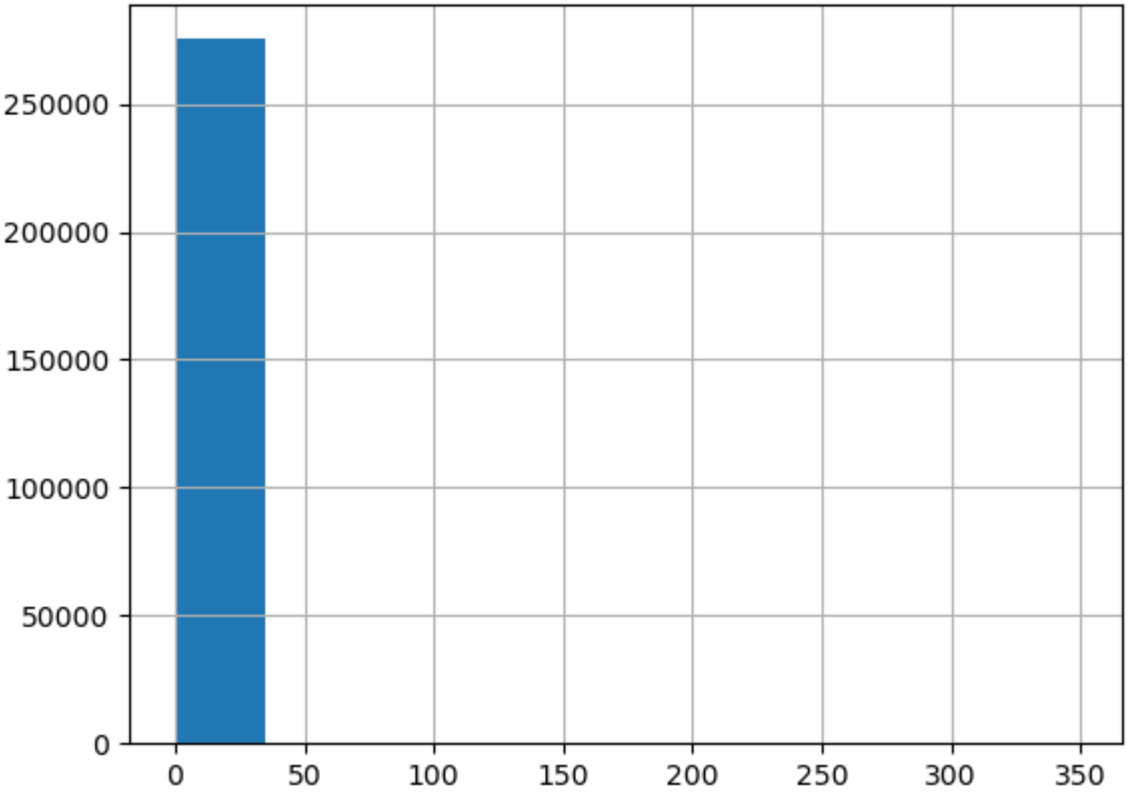
	V1	V2	V3	V4	V5	V6
count	275663.000000	275663.000000	275663.000000	275663.000000	275663.000000	275663.000000
mean	-0.037460	-0.002430	0.025520	-0.004359	-0.010660	-0.014206
std	1.952522	1.667260	1.507538	1.424323	1.378117	1.313213
min	-56.407510	-72.715728	-48.325589	-5.683171	-113.743307	-26.160506
25%	-0.941105	-0.614040	-0.843168	-0.862847	-0.700192	-0.765861
50%	-0.059659	0.070249	0.200736	-0.035098	-0.060556	-0.270931
75%	1.294471	0.819067	1.048461	0.753943	0.604521	0.387704
max	2.454930	22.057729	9.382558	16.875344	34.801666	73.301626

8 rows × 30 columns

Scaling the dataset

In [15]:

```
from sklearn.preprocessing import RobustScaler
new_df = df.copy()
new_df['Amount'] = RobustScaler().fit_transform(new_df['Amount'].to_numpy().reshape
new_df['Amount'].hist();
```



In [16]:

```
new_df['Amount'].describe()
```

Out[16]:

```
count    275663.000000
mean          0.908007
std           3.439940
min          -0.322511
25%          -0.236924
50%           0.000000
75%           0.763076
max         348.694743
Name: Amount, dtype: float64
```

Copying the contents of the data into new_df

In [17]:

```
new_df.head()
```

Out[17]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739

5 rows × 30 columns

Spliting the dataset into training and testing data

In [18]:
```
X = new_df.drop('Class',axis=1)
y = new_df['Class']
```

In [19]:
```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.20, random_state=4

print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (220530, 29)
y_train shape: (220530,)
X_test shape: (55133, 29)
y_test shape: (55133,)
```

Building Models with the unbalanced dataset

Logistic Regression

In [20]:
```
from sklearn.linear_model import LogisticRegression

logistic_model = LogisticRegression(max_iter=1000)
logistic_model.fit(X_train, y_train)
logistic_model.score(X_train, y_train)
```

Out[20]:
```
0.999183784519113
```

In [21]:
```
# Predicting the result
y_pred = logistic_model.predict(X_test)
```

In [22]:
```
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report

ac = accuracy_score(y_test,y_pred)*100
cm = confusion_matrix(y_test,y_pred)
cr= classification_report(y_test,y_pred, target_names=['Not Fraud', 'Fraud'])
print("accuracy score:",ac)
print("confusion matrix:",cm)
print("classification report:",cr)
```

```
accuracy score: 99.92200678359603
confusion matrix: [[55035     7]
 [   36    55]]
classification report:               precision    recall  f1-score   support

   Not Fraud       1.00      1.00      1.00     55042
       Fraud       0.89      0.60      0.72        91

    accuracy                           1.00     55133
   macro avg       0.94      0.80      0.86     55133
weighted avg       1.00      1.00      1.00     55133
```

Random Forest

In [23]:

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(max_depth=2, n_jobs=-1)
rf.fit(X_train, y_train)
```

Out[23]:

```
▾             RandomForestClassifier

RandomForestClassifier(max_depth=2, n_jobs=-1)
```

In [24]:

```
y_pred=rf.predict(X_test)
```

In [25]:

```
ac = accuracy_score(y_test,y_pred)*100
cm = confusion_matrix(y_test,y_pred)
cr= classification_report(y_test,y_pred, target_names=['Not Fraud', 'Fraud'])
print("accuracy score:",ac)
print("confusion matrix:",cm)
print("classification report:",cr)
```

```
accuracy score: 99.89842743910181
confusion matrix: [[55030    12]
 [   44    47]]
classification report:               precision    recall  f1-score   support

   Not Fraud       1.00      1.00      1.00     55042
       Fraud       0.80      0.52      0.63        91

    accuracy                           1.00     55133
   macro avg       0.90      0.76      0.81     55133
weighted avg       1.00      1.00      1.00     55133
```

Naive Bayes GaussianNB

In [26]:

```
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()
gnb.fit(X_train, y_train)

y_pred = gnb.predict(X_test)

ac = accuracy_score(y_test,y_pred)*100
cm = confusion_matrix(y_test,y_pred)
cr= classification_report(y_test,y_pred, target_names=['Not Fraud', 'Fraud'])
print("accuracy score:",ac)
print("confusion matrix:",cm)
print("classification report:",cr)
```

```
accuracy score: 97.81618994068889
confusion matrix: [[53857  1185]
 [   19    72]]
classification report:               precision    recall  f1-score   support

   Not Fraud       1.00      0.98      0.99     55042
       Fraud       0.06      0.79      0.11        91

    accuracy                           0.98     55133
   macro avg       0.53      0.88      0.55     55133
weighted avg       1.00      0.98      0.99     55133
```

Decision Tree

In [27]:

```
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier(random_state=42)
dtc.fit(X_train, y_train)

y_pred = dtc.predict(X_test)

ac = accuracy_score(y_test,y_pred)*100
cm = confusion_matrix(y_test,y_pred)
cr= classification_report(y_test,y_pred, target_names=['Not Fraud', 'Fraud'])
print("accuracy score:",ac)
print("confusion matrix:",cm)
print("classification report:",cr)
```

```
accuracy score: 99.89479984764115
confusion matrix: [[55007    35]
 [   23    68]]
classification report:               precision    recall  f1-score   support

   Not Fraud       1.00      1.00      1.00     55042
       Fraud       0.66      0.75      0.70        91

    accuracy                           1.00     55133
   macro avg       0.83      0.87      0.85     55133
weighted avg       1.00      1.00      1.00     55133
```

Balanced Dataset with OverSampling Technique

In [28]:
```
X = new_df.drop('Class',axis=1)
y = new_df['Class']

print("X.shape: ", X.shape)
print("y.shape: ", y.shape)
```

```
X.shape:  (275663, 29)
y.shape:  (275663,)
```

In [29]:
```
from imblearn.over_sampling import SMOTE

X_res,y_res = SMOTE().fit_resample(X,y)
y_res.value_counts()
```

Out[29]:
```
0    275190
1    275190
Name: Class, dtype: int64
```

Train Test Split on Balanced data

In [30]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X_res,y_res,test_size=0.20,random_

print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (440304, 29)
y_train shape: (440304,)
X_test shape: (110076, 29)
y_test shape: (110076,)
```

Logistic Regression on Balanced Data

In [31]:
```
from sklearn.linear_model import LogisticRegression

logistic_model = LogisticRegression(max_iter=1000)
logistic_model.fit(X_train, y_train)

y_pred = logistic_model.predict(X_test)

ac = accuracy_score(y_test,y_pred)*100
cm = confusion_matrix(y_test,y_pred)
cr= classification_report(y_test,y_pred, target_names=['Not Fraud', 'Fraud'])
print("accuracy score:",ac)
print("confusion matrix:",cm)
print("classification report:",cr)
```

```
accuracy score: 94.49925506014026
confusion matrix: [[53731  1342]
 [ 4713 50290]]
classification report:               precision    recall  f1-score   support

   Not Fraud       0.92      0.98      0.95     55073
       Fraud       0.97      0.91      0.94     55003

    accuracy                           0.94    110076
   macro avg       0.95      0.94      0.94    110076
weighted avg       0.95      0.94      0.94    110076
```

Random Forest on Balanced Data

In [32]:

```python
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(max_depth=2, n_jobs=-1)
rf.fit(X_train, y_train)

y_pred = rf.predict(X_test)

ac = accuracy_score(y_test,y_pred)*100
cm = confusion_matrix(y_test,y_pred)
cr= classification_report(y_test,y_pred, target_names=['Not Fraud', 'Fraud'])
print("accuracy score:",ac)
print("confusion matrix:",cm)
print("classification report:",cr)
```

```
accuracy score: 92.85947890548348
confusion matrix: [[54708   365]
 [ 7495 47508]]
classification report:               precision    recall  f1-score   support

   Not Fraud       0.88      0.99      0.93     55073
       Fraud       0.99      0.86      0.92     55003

    accuracy                           0.93    110076
   macro avg       0.94      0.93      0.93    110076
weighted avg       0.94      0.93      0.93    110076
```

GaussianNB on Balanced data

In [33]:

```python
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()
gnb.fit(X_train, y_train)

y_pred = gnb.predict(X_test)

ac = accuracy_score(y_test,y_pred)*100
cm = confusion_matrix(y_test,y_pred)
cr= classification_report(y_test,y_pred, target_names=['Not Fraud', 'Fraud'])
print("accuracy score:",ac)
print("confusion matrix:",cm)
print("classification report:",cr)
```

```
accuracy score: 91.18336422108362
confusion matrix: [[53698  1375]
 [ 8330 46673]]
classification report:               precision    recall  f1-score   support

   Not Fraud       0.87      0.98      0.92     55073
       Fraud       0.97      0.85      0.91     55003

    accuracy                           0.91    110076
   macro avg       0.92      0.91      0.91    110076
weighted avg       0.92      0.91      0.91    110076
```

Decision Tree on Balanced data

In [34]:

```python
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier(max_depth=6, random_state=42)
dtc.fit(X_train, y_train)

y_pred = dtc.predict(X_test)

ac = accuracy_score(y_test,y_pred)*100
cm = confusion_matrix(y_test,y_pred)
cr= classification_report(y_test,y_pred, target_names=['Not Fraud', 'Fraud'])
print("accuracy score:",ac)
print("confusion matrix:",cm)
print("classification report:",cr)
```

```
accuracy score: 96.14629892074566
confusion matrix: [[52862  2211]
 [ 2031 52972]]
classification report:               precision    recall  f1-score   support

   Not Fraud       0.96      0.96      0.96     55073
       Fraud       0.96      0.96      0.96     55003

    accuracy                           0.96    110076
   macro avg       0.96      0.96      0.96    110076
weighted avg       0.96      0.96      0.96    110076
```

In [ ]: