In [61]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
```

In [62]:
```
df=pd.read_csv(r"D:\codsoft\credit\creditcard.csv")
```

In [63]:
```
df.head()
```

Out[63]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.01
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.22
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.24
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.10
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.00

5 rows × 31 columns

In [64]:
```
df.shape
```

Out[64]:
```
(284807, 31)
```

Data Cleaning and Analyzing

In [65]:
```
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 [66]:
```
df.isnull().sum()
```

Out[66]:
```
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
```

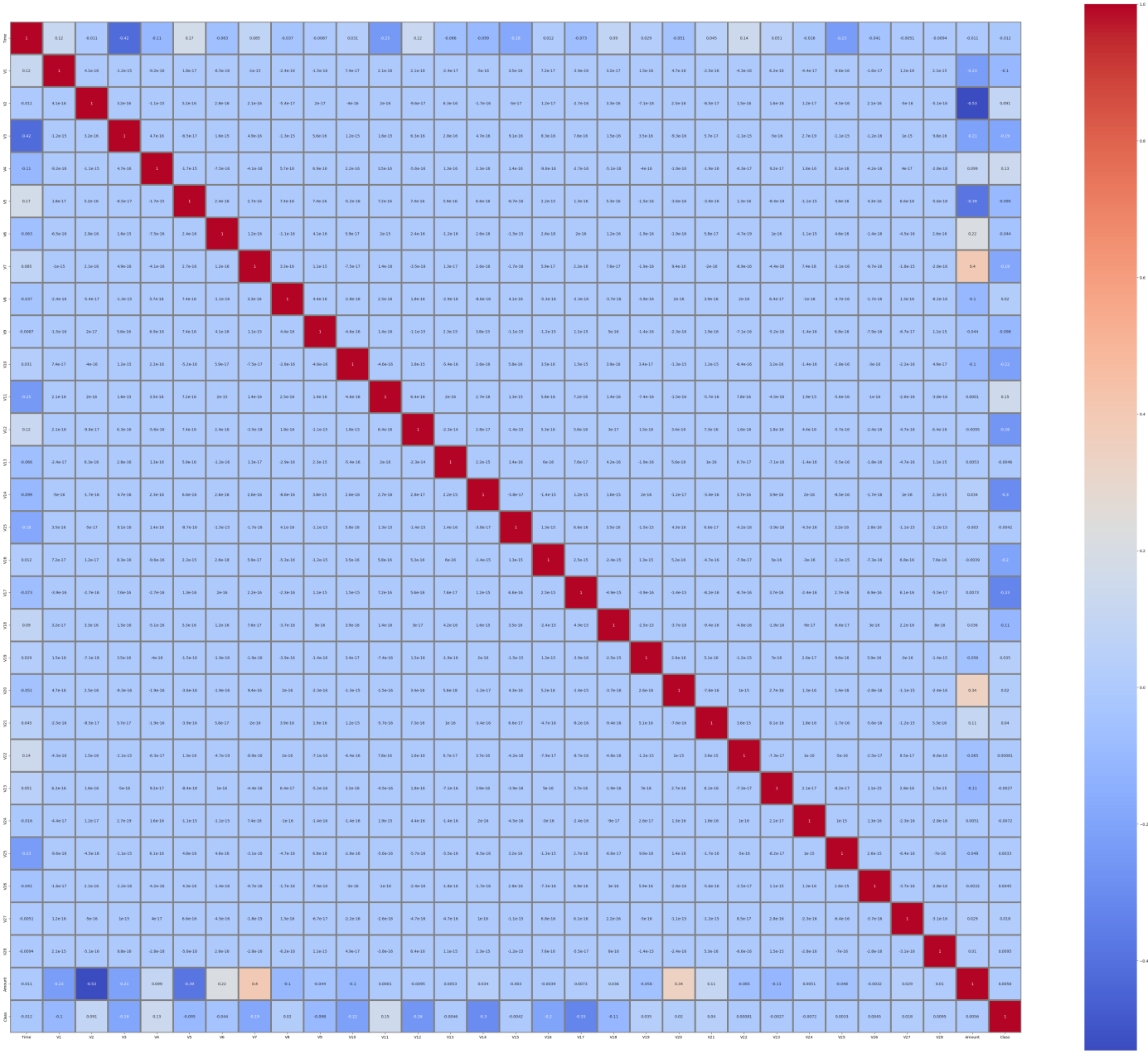
In [67]:
```
df.isnull().sum().sum()
```

Out[67]:
```
0
```

In [68]:
```
correlation=df.corr()
```

In [69]:
```
plt.figure(figsize=(60, 50))
sns.heatmap(correlation,annot=True,cmap="coolwarm",linewidths=3,square=True, linecolor="
```

Out[69]:
```
<Axes: >
```



In [70]:
```
df.nunique()
```

Out[70]:
```
Time      124592
V1        275663
V2        275663
V3        275663
V4        275663
V5        275663
V6        275663
V7        275663
V8        275663
V9        275663
V10       275663
V11       275663
V12       275663
V13       275663
V14       275663
V15       275663
V16       275663
V17       275663
V18       275663
V19       275663
V20       275663
V21       275663
V22       275663
```

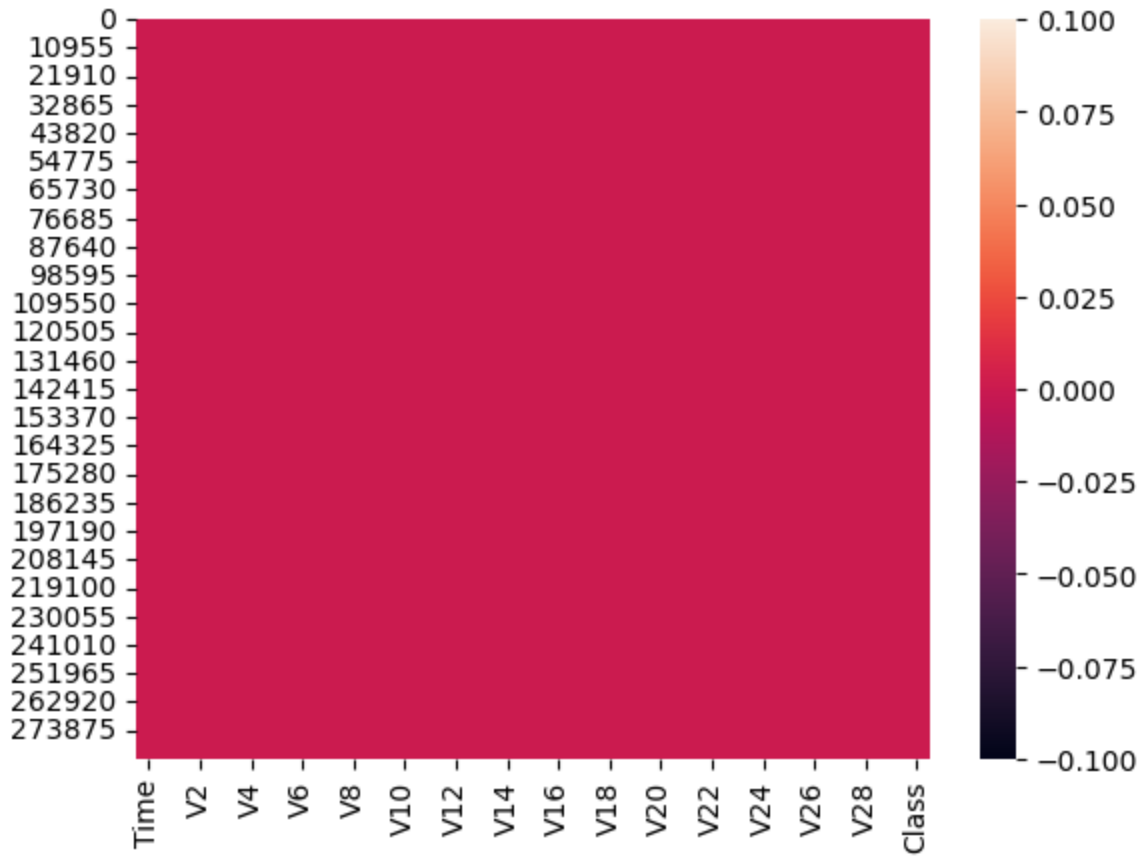
```
V23       275663
V24       275663
V25       275663
V26       275663
V27       275663
V28       275663
Amount     32767
Class          2
dtype: int64
```

In [71]:
```
100*df.Class.value_counts(normalize=True)
```

Out[71]:
```
Class
0    99.827251
1     0.172749
Name: proportion, dtype: float64
```

In [72]:
```
sns.heatmap(df.isnull())
```

Out[72]:
```
<Axes: >
```

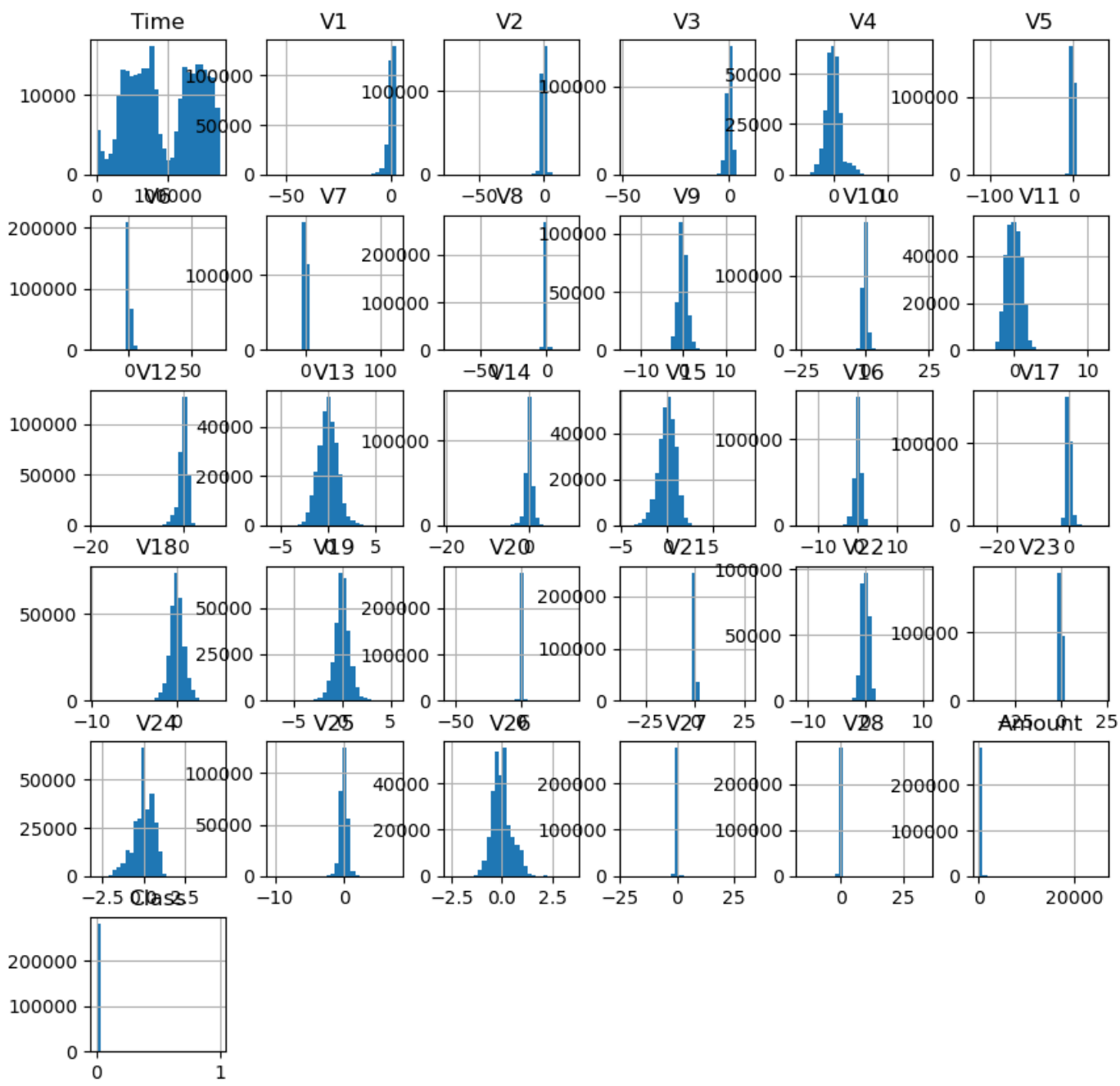


In [73]:
```
fig = px.histogram(df,
                   x='Class',
                   title='Class')

fig.update_layout(bargap=0.4)
fig.show()
```

In [75]:
```
df.hist(bins=30, figsize=(10,10))
```

Out[75]:
```
array([[<Axes: title={'center': 'Time'}>, <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'}>, <Axes: >, <Axes: >, <Axes: >,
        <Axes: >, <Axes: >]], dtype=object)
```



Logistic Regression & Random Forest Classification

In [14]:
```
x = df.drop('Class', axis=1)  # Use parentheses, and specify the axis
y = df['Class']  # Access the 'Class' column directly
```

In [15]:
```
x.head()
```

Out[15]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	0.25
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.06
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.52
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.20
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	0.40

5 rows × 30 columns

In [16]:
```
y.head()
```

Out[16]:
```
0    0
1    0
2    0
3    0
4    0
Name: Class, dtype: int64
```

In [17]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x,y, test_size=0.3,random_state=50)
```

In [18]:
```
print('Train.shape :', x.shape)
print('Test.shape :', y.shape)
```

```
Train.shape : (284807, 30)
Test.shape : (284807,)
```

In [19]:
```
from sklearn.linear_model import LogisticRegression
```

In [20]:
```
model = LogisticRegression()
model.fit(X_train,y_train)
```

```
D:\anaconda\Lib\site-packages\sklearn\linear_model\_logistic.py:460: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

Out[20]:
```
▾ LogisticRegression
LogisticRegression()
```

In [21]:
```
predict = model.predict(X_test)
predict
```

Out[21]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

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

In [23]:
```
accuracy1=accuracy_score(y_test,predict)*100
accuracy1
```

Out[23]:
```
99.89817773252344
```

In [24]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [25]:
```
clf = RandomForestClassifier(criterion = 'entropy', max_features = 4, n_estimators = 20,
```

In [29]:
```
clf.fit(X_train, y_train)
print(x)
```

```
            Time         V1         V2        V3        V4        V5  \
0            0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321
1            0.0   1.191857   0.266151  0.166480  0.448154  0.060018
2            1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198
3            1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309
```

```
4            2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193
...          ...        ...        ...       ...       ...       ...
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515
284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546

               V6        V7        V8        V9  ...       V20       V21  \
0        0.462388  0.239599  0.098698  0.363787  ...  0.251412 -0.018307
1       -0.082361 -0.078803  0.085102 -0.255425  ... -0.069083 -0.225775
2        1.800499  0.791461  0.247676 -1.514654  ...  0.524980  0.247998
3        1.247203  0.237609  0.377436 -1.387024  ... -0.208038 -0.108300
4        0.095921  0.592941 -0.270533  0.817739  ...  0.408542 -0.009431
...           ...       ...       ...       ...  ...       ...       ...
284802 -2.606837 -4.918215  7.305334  1.914428  ...  1.475829  0.213454
284803  1.058415  0.024330  0.294869  0.584800  ...  0.059616  0.214205
284804  3.031260 -0.296827  0.708417  0.432454  ...  0.001396  0.232045
284805  0.623708 -0.686180  0.679145  0.392087  ...  0.127434  0.265245
284806 -0.649617  1.577006 -0.414650  0.486180  ...  0.382948  0.261057

               V22       V23       V24       V25       V26       V27       V28  \
0        0.277838 -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053
1       -0.638672  0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724
2        0.771679  0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752
3        0.005274 -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458
4        0.798278 -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153
...           ...       ...       ...       ...       ...       ...       ...
284802   0.111864  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731
284803   0.924384  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527
284804   0.578229 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561
284805   0.800049 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533
284806   0.643078  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649

        Amount
0       149.62
1         2.69
2       378.66
3       123.50
4        69.99
...        ...
284802    0.77
284803   24.79
284804   67.88
284805   10.00
284806  217.00

[284807 rows x 30 columns]
```

In [30]:
```
predictions = clf.predict(X_test)
predictions
```

Out[30]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [33]:
```
accuracy1=accuracy_score(y_test,predictions)*100
accuracy1
```

Out[33]:
```
99.9531851643786
```

In [37]:
```
pd.DataFrame({ 'Classs': y_test,'Predictions': model.predict(X_test), 'predict':y_test})
```

Out[37]:

	Classs	Predictions	predict
40312	0	0	0

8353	0	0	0
100133	0	0	0
113766	0	0	0
31765	0	0	0
...	...	...	...
233973	0	0	0
196650	0	0	0
222150	0	0	0
255142	0	0	0
60299	0	0	0

85443 rows × 3 columns

In [41]:
```
H=model.score(X_test, y_test)*100
```

In [45]:
```
K=clf.score(X_test, y_test)*100
```

Out[45]:
```
99.89817773252344
```

In [55]:
```
data = {
    'Logistic Regression Accuracy': H,
    'Random Forest Accuracy': K
}


index = ['Score']
df = pd.DataFrame(data, index=index)
```

In [56]:
```
df
```

Out[56]:

	Logistic Regression Accuracy	Random Forest Accuracy
Score	99.898178	99.953185

In [ ]:
```

```