In [11]:
```
# Imports
import pandas as pd
import numpy as np
import plotly as px
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

In [12]:
```
df = pd.read_csv("/Users/jay/creditcardfraud/creditcard.csv")
```

In [65]:
```
df.head()
```

Out[65]:

	Time	V1	V2	V3	V4	V5	V6	V7	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.09
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.08
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.24
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.37
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.27

5 rows × 31 columns

In [66]:
```
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 [67]:

```
df.isnull().sum()
```

Out[67]:

```
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 [68]:

```
df.describe()
```

Out[68]:

	Time	V1	V2	V3	V4	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.8480
mean	94813.859575	3.918649e-15	5.682686e-16	-8.761736e-15	2.811118e-15	-1.5521
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.3802
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.1374
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.43
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.1192
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.4801

8 rows × 31 columns

In [20]:
```
# legit and fraud transaction data
fraud = df[df['Class']==1]
legit = df[df['Class']==0]
```

In [21]:
```
fraud['Amount'].describe()
```

Out[21]:
```
count      492.000000
mean       122.211321
std        256.683288
min          0.000000
25%          1.000000
50%          9.250000
75%        105.890000
max       2125.870000
Name: Amount, dtype: float64
```

In [22]:
```
legit['Amount'].describe()
```

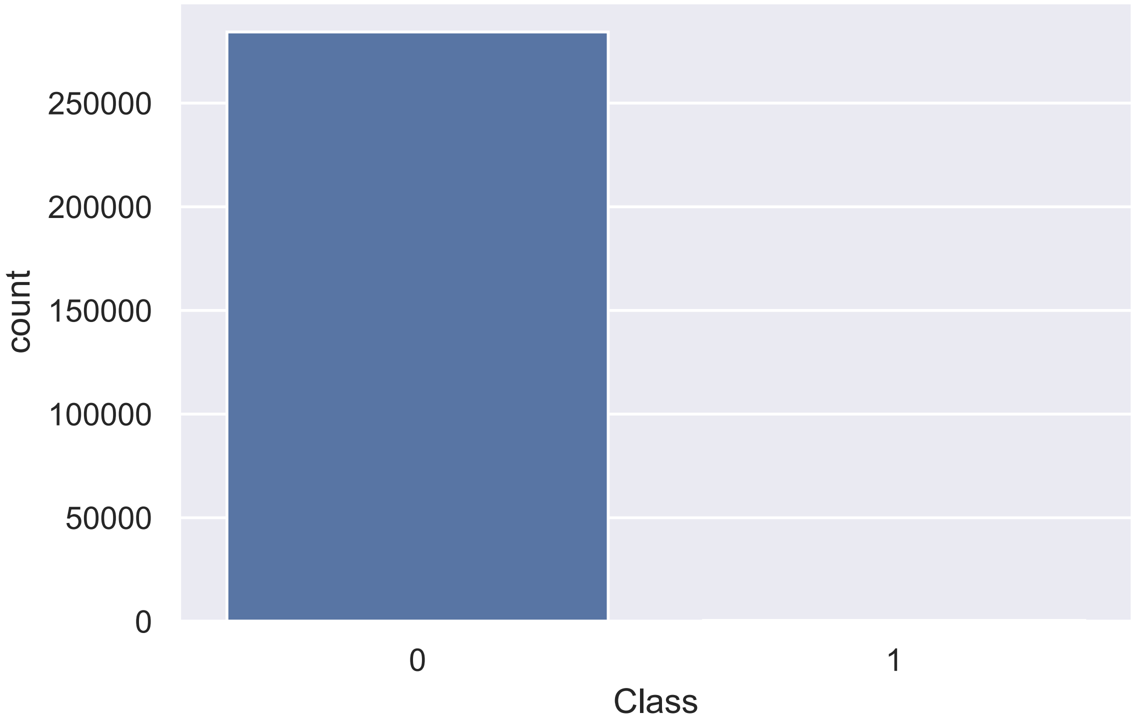
Out[22]:
```
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

Here the mean of legit transaction is $88 and the mean of the fradulent transaction is $122 which is much higher than than the legit transaction even if the max amount on legit transaction is $25691

In [23]:
```
sns.countplot('Class', data=df)
```

```
/Users/jay/opt/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py
:36: FutureWarning: Pass the following variable as a keyword arg: x. From v
ersion 0.12, the only valid positional argument will be `data`, and passing
other arguments without an explicit keyword will result in an error or misi
nterpretation.
  warnings.warn(
```

Out[23]: <AxesSubplot:xlabel='Class', ylabel='count'>



In [24]:

```
df.groupby('Class').mean()
```

Out[24]:

Class	Time	V1	V2	V3	V4	V5	V6	
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.0(
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.5

2 rows × 32 columns

As we can see there is a substantial difference betweeen the mean of legit and fraud transaction in almost all the features.

In [50]:

```
legit_sample = legit.sample(n=492)    # which is equivalent to the fradulen
```

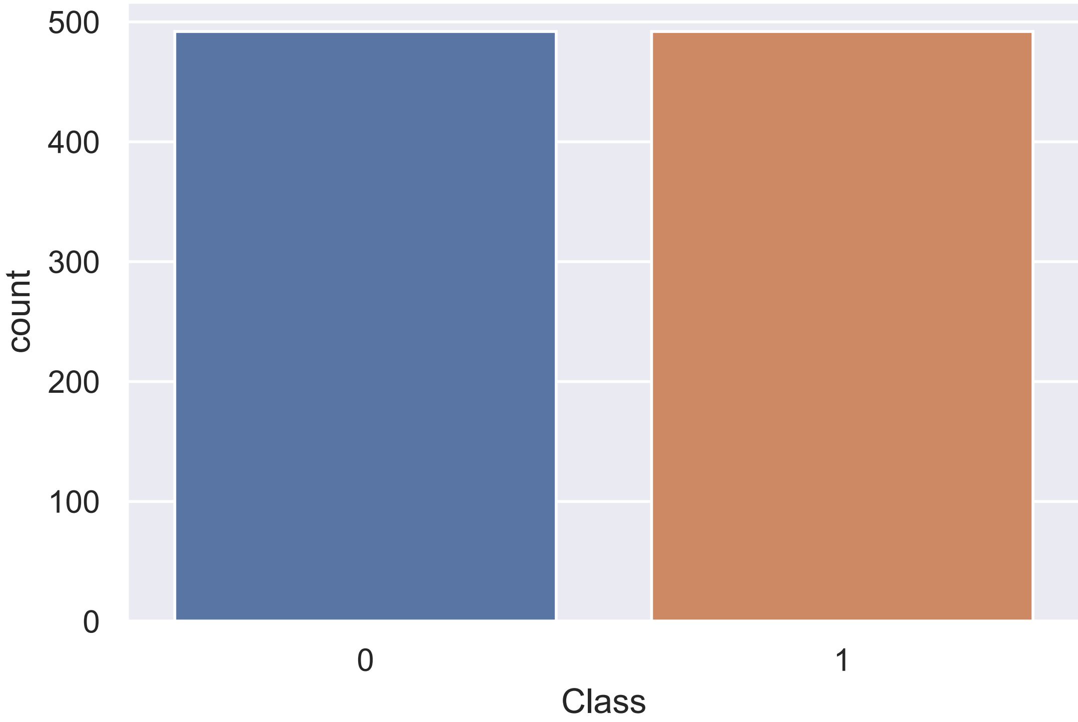
In [51]:
```
# Concatenate legit_sample and fraud

df_undersample = pd.concat([legit_sample, fraud], axis=0)
```

In [27]:
```
sns.countplot('Class', data=df_undersample)
```

```
/Users/jay/opt/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py
:36: FutureWarning: Pass the following variable as a keyword arg: x. From v
ersion 0.12, the only valid positional argument will be `data`, and passing
other arguments without an explicit keyword will result in an error or misi
nterpretation.
  warnings.warn(
```

Out[27]: <AxesSubplot:xlabel='Class', ylabel='count'>



Now as we have equal counts of the type of transaction, we can feed in the data for machine learning

In [28]:
```
# Let's recheck the mean again if there is still any difference in mean va

df_undersample.groupby('Class').mean()
```

Out[28]:

	Time	V1	V2	V3	V4	V5	V6	
Class								
0	97271.987805	0.077204	0.074780	-0.037623	0.106936	0.088720	-0.014933	0.026
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.56

2 rows × 32 columns

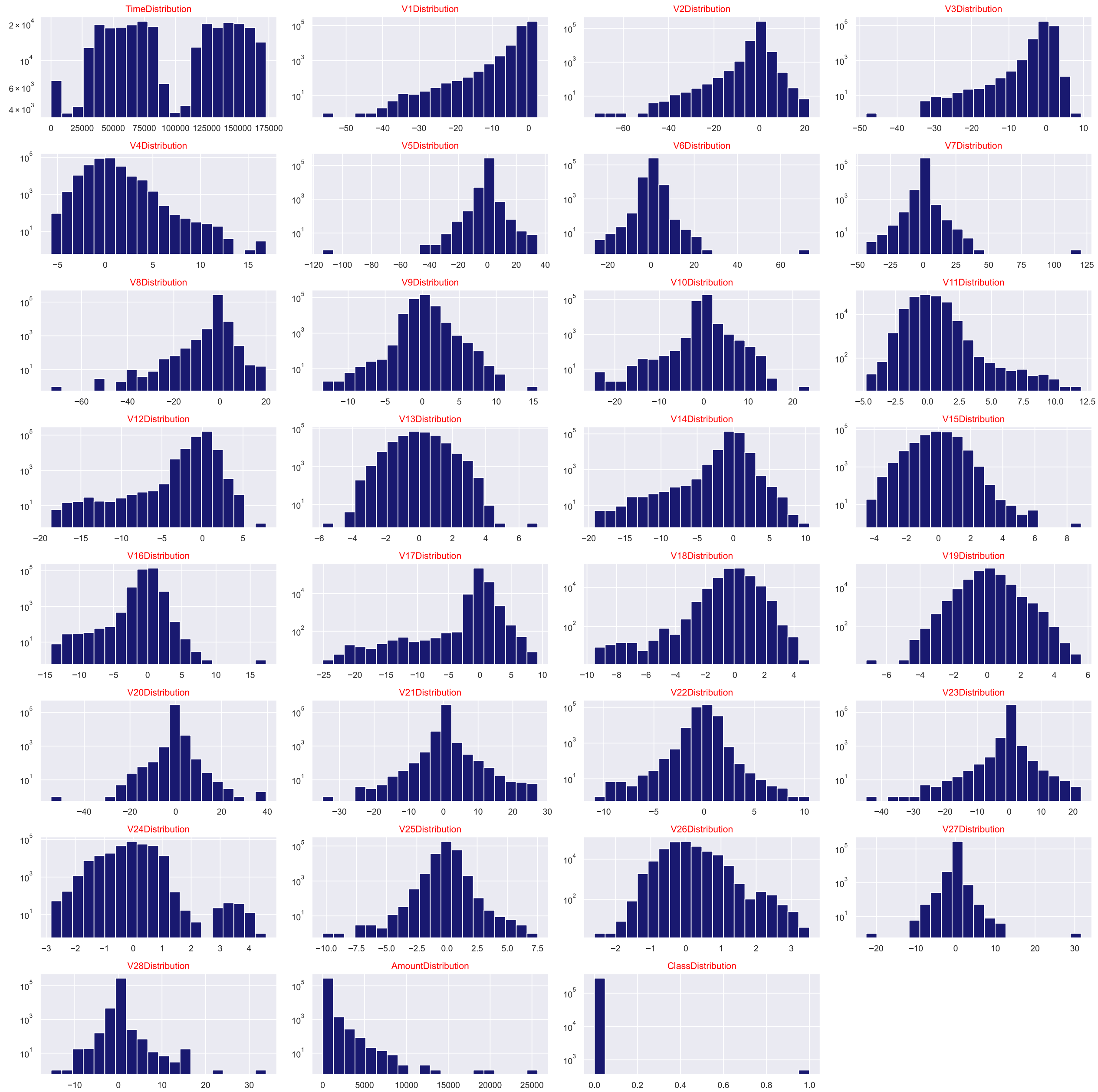
So even after discarding the major amount of data, there is still difference in mean between legit and fraud transaction

In [41]:

```
# Checking for the distribution

def multi_histogram(dataframe, features, rows, cols):
    fig = plt.figure(figsize=(20,20))
    for i, feature in enumerate(features):
        ax = fig.add_subplot(rows, cols, i+1)
        dataframe[feature].hist(bins=20, ax=ax, facecolor='midnightblue')
        ax.set_title(feature + "Distribution", color="red")
        ax.set_yscale('log')
    fig.tight_layout()
    plt.show()

multi_histogram(df, df.columns,8,4)
```



Since most of the data is already scaled, we need to scale the rest of the variables (Amount and time) and even the chart looks skewed to the right which means there are outlies or less bigger amount in the transaction history

In [29]:

```
from sklearn.preprocessing import StandardScaler, RobustScaler

# Robust scaler can handle outliers
std_scaler = StandardScaler()
rob_scaler = RobustScaler()

df['scaled_amount'] = rob_scaler.fit_transform(df['Amount'].values.reshape
df['scaled_time'] = rob_scaler.fit_transform(df['Time'].values.reshape(-1,1

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

In [30]:

```
df_scaled.head()
```

Out[30]:

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

5 rows × 31 columns

In [31]:

```
df_scaled = df_scaled[['scaled_time','V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V
       'V8', 'V9', 'V10', 'V11','V12', 'V13', 'V14', 'V15', 'V16', 'V17',
       'V19', 'V20', 'V21','V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28'
       'scaled_amount','Class'
        ]]
```

In [32]:

```
df_scaled.head()
```

Out[32]:

	scaled_time	V1	V2	V3	V4	V5	V6	V7
0	-0.994983	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.23959
1	-0.994983	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.07880
2	-0.994972	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.79146
3	-0.994972	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.23760
4	-0.994960	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.59294

5 rows × 31 columns

First we'll use under sampling method by taking the number of normal transaction equivalent to the fradulent transaction.

In [70]:
```
df_undersample = df_undersample.drop(['Amount', 'Time'], axis=1)
```

In [71]:
```
df_undersample_scaled = df_undersample[['scaled_time','V1', 'V2', 'V3', 'V4
       'V8', 'V9', 'V10', 'V11','V12', 'V13', 'V14', 'V15', 'V16', 'V17',
       'V19', 'V20', 'V21','V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28',
       'scaled_amount','Class'
        ]]
```

In [72]:
```
df_undersample_scaled.shape
```

Out[72]: (984, 31)

In [102…
```
X = df_undersample_scaled.drop('Class', axis=1)
y = df_undersample_scaled['Class']
```

In [98]:
```
X.shape, y.shape
```

Out[98]: ((984, 30), (984,))

In [99]:
```
# Splitting data into training and testing data

from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
```

In [103…
```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

(787, 30) (197, 30) (787,) (197,)

In [104…
```
# Binary classification

from sklearn.linear_model import LogisticRegression

reg_log = LogisticRegression()

reg_log.fit(X_train, y_train)
```

Out[104… LogisticRegression()

In [105…
```
y_train_predict = reg_log.predict(X_train)
```

In [109…
```
# Accuracy

from sklearn.metrics import accuracy_score, classification_report

train_accuracy = accuracy_score( y_train, y_train_predict)
```

In [107…
```
print(train_accuracy)
```

```
0.951715374841169
```

In [108…
```
# Now finding the accuracy on test set

y_predict = reg_log.predict(X_test)

reg_accuracy_score = accuracy_score(y_test, y_predict)
print(reg_accuracy_score)
```

```
0.949238578680203
```

In [110…
```
print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.91      0.99      0.95        95
           1       0.99      0.91      0.95       102

    accuracy                           0.95       197
   macro avg       0.95      0.95      0.95       197
weighted avg       0.95      0.95      0.95       197
```