creditcardfrauddetection

November 4, 2023

0.0.1 Problem Statement: To determine whether the credit card transaction done was fraudulent(1) or genuine(0).

[1]:
```
# !pip install -r requirements.txt
```

[2]:
```
# dataset source : https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud/
↪discussion/373669
```

[3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from ydata_profiling import ProfileReport # for auto EDA


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

[4]:
```
# !pip install ydata-profiling
```

[5]:
```
credit_data = pd.read_csv("creditcard.csv")
```

[6]:
```
credit_data.sample(5)
```

[6]:

	Time	V1	V2	V3	V4	V5	V6	\
57789	48076.0	-0.938551	0.369490	1.470454	0.608430	0.761627	0.615952	\
250751	155050.0	0.285412	0.837827	-0.421605	-0.568154	0.987699	-0.624011	
154260	100943.0	-1.336136	-0.540035	0.079651	0.517510	2.690960	-1.252076	
73733	55257.0	-0.183733	2.126842	-4.262701	0.519193	2.820698	2.253822	
213970	139472.0	-0.687378	0.877389	0.725428	-0.507154	0.277876	0.427389	

	V7	V8	V9	…	V21	V22	V23	
57789	1.544211	-0.531758	0.143019	…	-0.343632	-0.256711	-0.102440	\
250751	0.849450	-0.037080	-0.388924	…	-0.196628	-0.554900	0.174315	
154260	0.320931	0.044992	0.796500	…	0.170119	0.232353	0.079490	
73733	-0.370254	1.548207	-0.547463	…	-0.135336	-0.514711	0.213927	

```
213970 -0.099900  0.475242  0.241289  …  0.356397  1.119852 -0.253242

             V24       V25       V26       V27       V28  Amount  Class
57789  -0.415812 -0.114067 -0.641563 -0.546678 -0.462172  138.13      0
250751  0.667165 -0.870728  0.008427  0.031762  0.068723    0.99      0
154260  0.585394  0.073484 -0.721025  0.051023  0.192172   29.70      0
73733   0.648686 -0.203964 -0.387047 -0.012192 -0.065566    1.79      0
213970  0.112419 -0.089989 -0.126535 -0.038203  0.110873    1.55      0

[5 rows x 31 columns]
```

[7]:
```
print(f' We have {credit_data.shape[0]} credit card transactions data in our dataset.')
```

```
 We have 284807 credit card transactions data in our dataset.
```

[8]:
```
credit_data.columns
```

[8]:
```
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')
```

V1 to V28 columns has been hidden on purpose due to various privacy reasons and also it is mentioned that V1 to V28 are the principle components obtained after PCA reduction

[9]:
```
credit_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
```

Features V1, V2, … V28 are the principal components obtained with PCA. The only features which have not been transformed with PCA are 'Time' and 'Amount'.

- **Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset.**
- **The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning.**
- **Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.**

[10]:
```
credit_data.Class.value_counts()
```

[10]:
```
Class
0    284315
1       492
Name: count, dtype: int64
```

Observation: - class is categorical data with 2 categories: 0 and 1. - highly imbalanced dataset with 492 anomaly transactions.

[11]:
```
print(f'There are only {credit_data[credit_data.Class == 1].shape[0]} number of␣
 ↪fraudulent transactions out of total {credit_data.shape[0]} total␣
 ↪transactions.')
```

```
There are only 492 number of fraudulent transactions out of total 284807 total
transactions.
```

[12]:
```
print(f'Fraud transaction happened only {round(credit_data[credit_data.Class ==␣
 ↪1].shape[0]/credit_data.shape[0] * 100 , 3)} % of time.')
```

```
Fraud transaction happened only 0.173 % of time.
```

0.0.2 thus our data is highly imbalanced and if we try training any supervised algorithm with this data, then it will most likely mark fraud transaction as geniuene.

What is Imbalanced Data?

Imbalanced data refers to a situation in machine learning where one category or class of data (called the minority class) is much less common than another category (the majority class). This imbalance can be a problem because machine learning models may not work well when they are heavily biased towards the majority class.

Techniques to Handle Imbalanced Data

Resampling Techniques: These methods involve adjusting the number of examples in each class. Oversampling creates more copies of the minority class to balance the dataset, while undersampling reduces the number of majority class examples to balance the dataset.

Cost-Sensitive Learning: This approach involves modifying the loss function of the model to give more weight to the minority class. It assigns a higher "cost" to mistakes in the minority class, making the model more sensitive to misclassifying examples from the minority class.

Ensemble Methods: These methods are like teamwork for models. Instead of relying on one model, you use multiple models and combine their predictions. This can help effectively handle imbalanced data.

Thresholding: You can change the decision threshold of your model. By making the threshold higher, you make your model more cautious, making it more likely to classify examples as the minority class. Lowering the threshold makes it more likely to classify examples as the majority class.

Generative Adversarial Networks (GANs): GANs are used to create artificial new examples of the minority class. These artificial synthetic examples are generated by the GAN to balance out the dataset, making it more even.

[13]:
```
credit_data.describe()
```

[13]:
```
                Time            V1            V2            V3            V4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05   \
mean    94813.859575  1.168375e-15  3.416908e-16 -1.379537e-15  2.074095e-15
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01

                 V5            V6            V7            V8            V9
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  \
mean   9.604066e-16  1.487313e-15 -5.556467e-16  1.213481e-16 -2.406331e-15
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00
```

```
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01

       …           V21           V22           V23           V24
count  …  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  \
mean   …  1.654067e-16 -3.568593e-16  2.578648e-16  4.473266e-15
std    …  7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min    … -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%    … -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50%    … -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%    …  1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max    …  2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00

                V25           V26           V27           V28         Amount
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  284807.000000  \
mean   5.340915e-16  1.683437e-15 -3.660091e-16 -1.227390e-16      88.349619
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01     250.120109
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01       0.000000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02       5.600000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02      22.000000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02      77.165000
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01   25691.160000

              Class
count  284807.000000
mean        0.001727
std         0.041527
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

[8 rows x 31 columns]
```

[14]:
```
credit_data.isna().sum()
```

[14]:
```
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
```

[15]:
```
credit_data.duplicated().sum()
```

[15]:
```
1081
```

Observation: - no missing values - 1081 transactions have been duplicated(maybe same amount transaction done twice)

[16]:
```
credit_data.drop_duplicates(inplace = True)
```

Pandas Profiler - Auto EDA

[17]:
```
# report = ProfileReport(credit_data, title="Profiling Report")
# report.to_file('ProfileReport.html')
```

[18]:
```
# lets see the transactions count !

print(f' Minimum transaction amount: {credit_data.Amount.min()}')
print(f' Maximum transaction amount: {credit_data.Amount.max()}')
```

```
Minimum transaction amount: 0.0
Maximum transaction amount: 25691.16
```

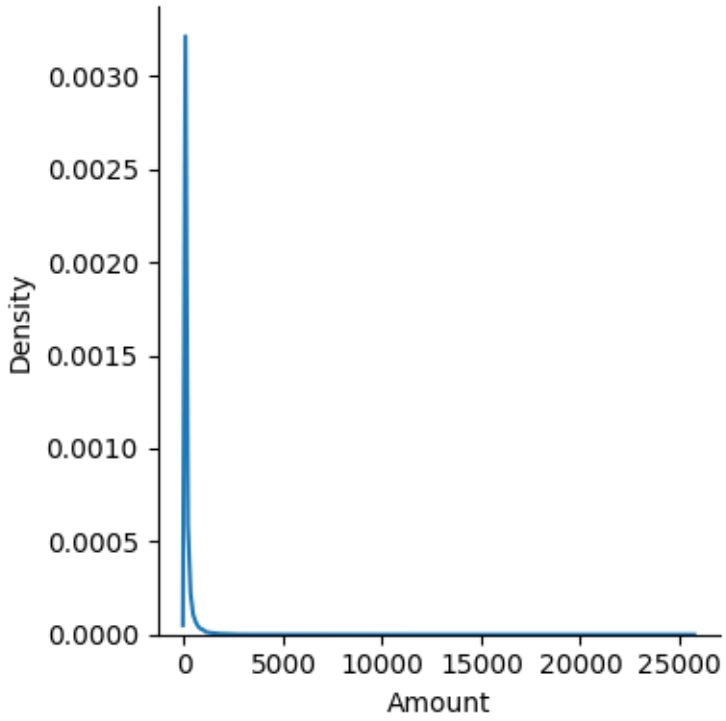
[19]:
```
credit_data.Amount.describe()
```

[19]:
```
count    283726.000000
mean         88.472687
std         250.399437
min           0.000000
25%           5.600000
50%          22.000000
75%          77.510000
max       25691.160000
Name: Amount, dtype: float64
```

Observation: - 0 rupees transaction of credit card means?

[20]:
```
sns.displot(data = credit_data, x = "Amount", kind = 'kde', height=4)
```

[20]:
```
<seaborn.axisgrid.FacetGrid at 0x1f34fd4ecd0>
```

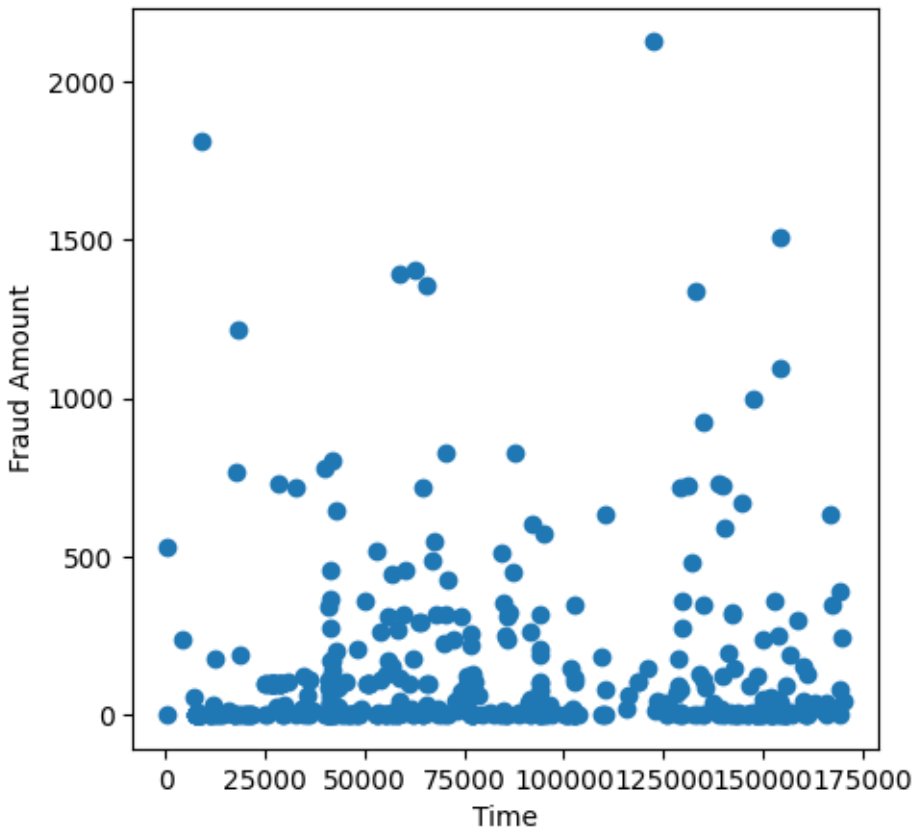


[21]:
```
fraud_df = credit_data[credit_data.Class == 1]

fraud_df.head()
```

[21]:
```
         Time        V1        V2        V3        V4        V5        V6  \
541     406.0 -2.312227  1.951992 -1.609851  3.997906 -0.522188 -1.426545
623     472.0 -3.043541 -3.157307  1.088463  2.288644  1.359805 -1.064823
4920   4462.0 -2.303350  1.759247 -0.359745  2.330243 -0.821628 -0.075788
6108   6986.0 -4.397974  1.358367 -2.592844  2.679787 -1.128131 -1.706536
6329   7519.0  1.234235  3.019740 -4.304597  4.732795  3.624201 -1.357746

            V7        V8        V9  …       V21       V22       V23  \
541  -2.537387  1.391657 -2.770089  …  0.517232 -0.035049 -0.465211
623   0.325574 -0.067794 -0.270953  …  0.661696  0.435477  1.375966
4920  0.562320 -0.399147 -0.238253  … -0.294166 -0.932391  0.172726
6108 -3.496197 -0.248778 -0.247768  …  0.573574  0.176968 -0.436207
6329  1.713445 -0.496358 -1.282858  … -0.379068 -0.704181 -0.656805

           V24       V25       V26       V27       V28  Amount  Class
541   0.320198  0.044519  0.177840  0.261145 -0.143276    0.00      1
623  -0.293803  0.279798 -0.145362 -0.252773  0.035764  529.00      1
4920 -0.087330 -0.156114 -0.542628  0.039566 -0.153029  239.93      1
6108 -0.053502  0.252405 -0.657488 -0.827136  0.849573   59.00      1
6329 -1.632653  1.488901  0.566797 -0.010016  0.146793    1.00      1

[5 rows x 31 columns]
```

[22]:
```
plt.figure(figsize = (5,5))

plt.subplot(1,1,1)
plt.scatter(fraud_df.Time, fraud_df.Amount)
plt.ylabel('Fraud Amount')
plt.xlabel('Time')
```

[22]:
```
Text(0.5, 0, 'Time')
```



most of the fraud amount is near 0 and less than 200 and only few of them are above 1500$.

[23]:
```
print(f' There are {fraud_df[fraud_df.Amount == 0].shape[0]}  number of frauds␣
 ↪with amount 0.00 $ out of {fraud_df.shape[0]} frauds.')
```

```
 There are 25  number of frauds with amount 0.00 $ out of 473 frauds.
```

[24]:
```
print(f' There are only {fraud_df[fraud_df.Amount >= 1000 ].shape[0]}  number␣
 ↪of frauds with amount greater than equal to 1000$, out of {fraud_df.
 ↪shape[0]} frauds.')
```

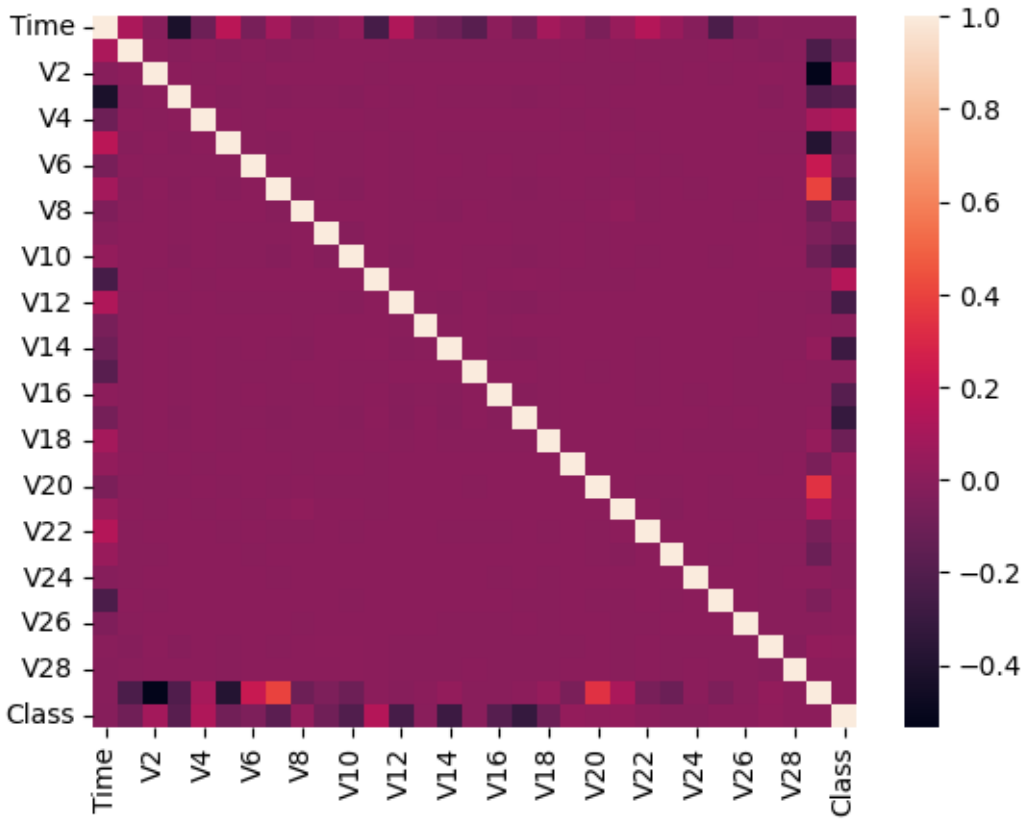
```
 There are only 9  number of frauds with amount greater than equal to 1000$, out
of 473 frauds.
```

[25]:
```
print(f' Maximum fraud amount: {fraud_df.Amount.max()}')
```

```
 Maximum fraud amount: 2125.87
```

[26]:
```
sns.heatmap(credit_data.corr())
```

[26]:
```
<Axes: >
```



no features are much corelated with each other. maybe they are but we cant see them cause of imbalanced dataset.

1 scaling and splitting

scaling the remaining columns time and amount using robust scaler cause robust scalar handles outliers pretty well.

[27]:
```
from sklearn.preprocessing import RobustScaler

robust_scaler = RobustScaler()

credit_data['scaled_amount'] = robust_scaler.fit_transform(credit_data.Amount.
 ↪values.reshape(-1,1))
credit_data['scaled_time'] = robust_scaler.fit_transform(credit_data.Time.
 ↪values.reshape(-1,1))
credit_data.drop(['Time','Amount'], axis = 1 , inplace = True)
```

[28]:
```
print('Final Scaled data looks like:')
credit_data.head()
```

Final Scaled data looks like:

[28]:

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

	V8	V9	V10	…	V22	V23	V24	V25	\
0	0.098698	0.363787	0.090794	…	0.277838	-0.110474	0.066928	0.128539	\
1	0.085102	-0.255425	-0.166974	…	-0.638672	0.101288	-0.339846	0.167170	
2	0.247676	-1.514654	0.207643	…	0.771679	0.909412	-0.689281	-0.327642	
3	0.377436	-1.387024	-0.054952	…	0.005274	-0.190321	-1.175575	0.647376	
4	-0.270533	0.817739	0.753074	…	0.798278	-0.137458	0.141267	-0.206010	

	V26	V27	V28	Class	scaled_amount	scaled_time
0	-0.189115	0.133558	-0.021053	0	1.774718	-0.995290
1	0.125895	-0.008983	0.014724	0	-0.268530	-0.995290
2	-0.139097	-0.055353	-0.059752	0	4.959811	-0.995279
3	-0.221929	0.062723	0.061458	0	1.411487	-0.995279
4	0.502292	0.219422	0.215153	0	0.667362	-0.995267

[5 rows x 31 columns]

We need to create a sub-sample of our actual dataframe i.e undersampling

1) It would reduce the overfitting error where most of the data are being predicted as geniuene due to imbalanced dataset.

2) It would allow us to see the actual correlation between various features , which we are not able to see right now due to the imbalance in dataset.

[29]:
```
from sklearn.utils import shuffle
credit_data = shuffle(credit_data)
```

[30]:
```
fraud_df = credit_data[credit_data.Class == 1]
geniuene_df = credit_data[credit_data.Class == 0]
```

[31]:
```
fraud_df.shape
```

[31]: (473, 31)

[32]:
```
geniuene_df = geniuene_df[0:fraud_df.shape[0]]
```

[33]:
```
geniuene_df.shape
```

[33]: (473, 31)

[34]:
```
new_df = pd.concat([fraud_df, geniuene_df])
new_df = shuffle(new_df)
```

[35]:
```
new_df.isna().sum().max()
```

[35]:
```
0
```

[36]:
```
new_df.shape
```

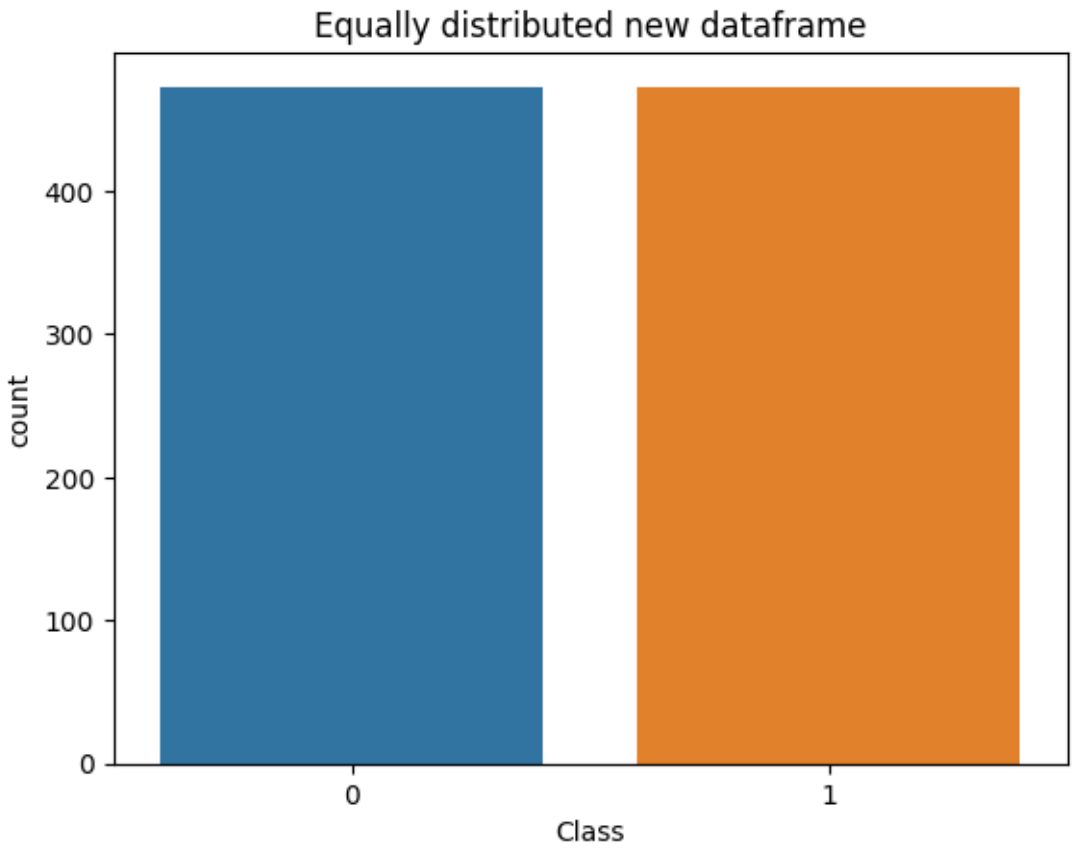
[36]:
```
(946, 31)
```

[37]:
```
new_df.Class.value_counts()
# equally distributed
```

[37]:
```
Class
1    473
0    473
Name: count, dtype: int64
```

[38]:
```
sns.countplot(data = new_df, x= 'Class')
plt.title('Equally distributed new dataframe')
```

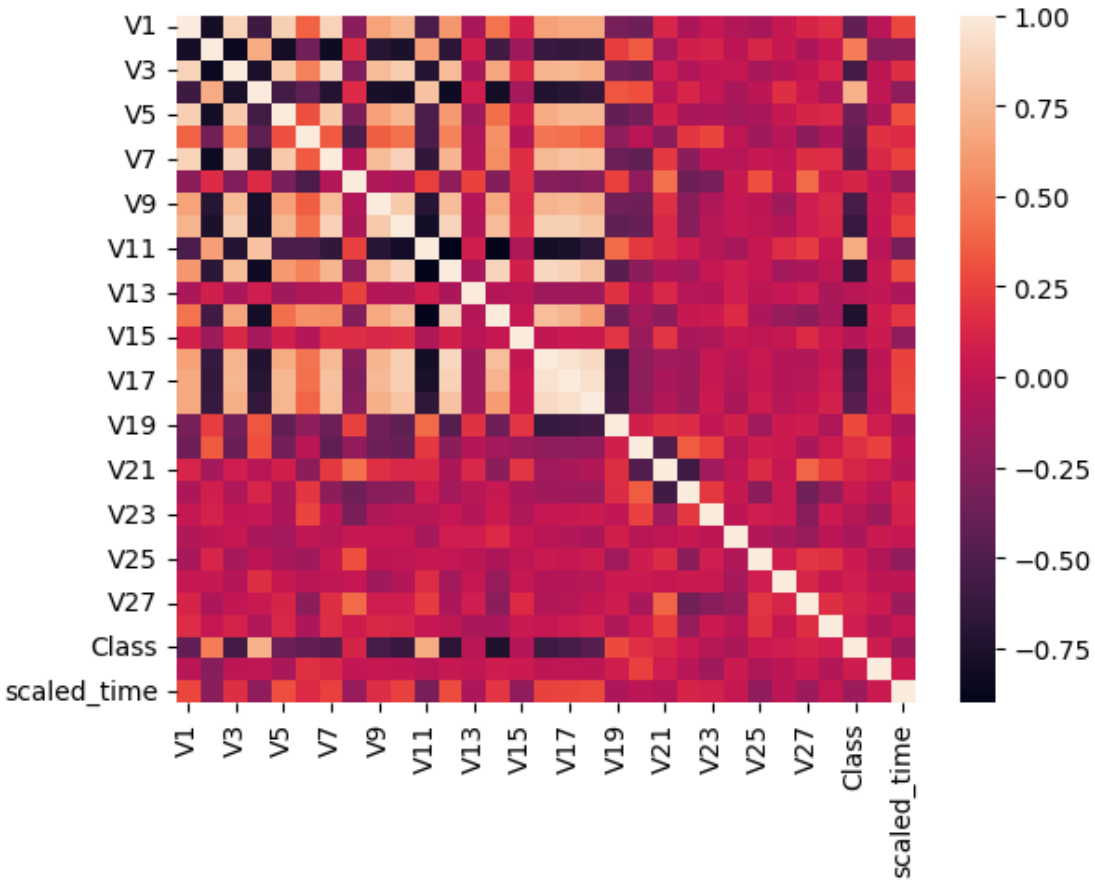
[38]:
```
Text(0.5, 1.0, 'Equally distributed new dataframe')
```



lets work with this new df

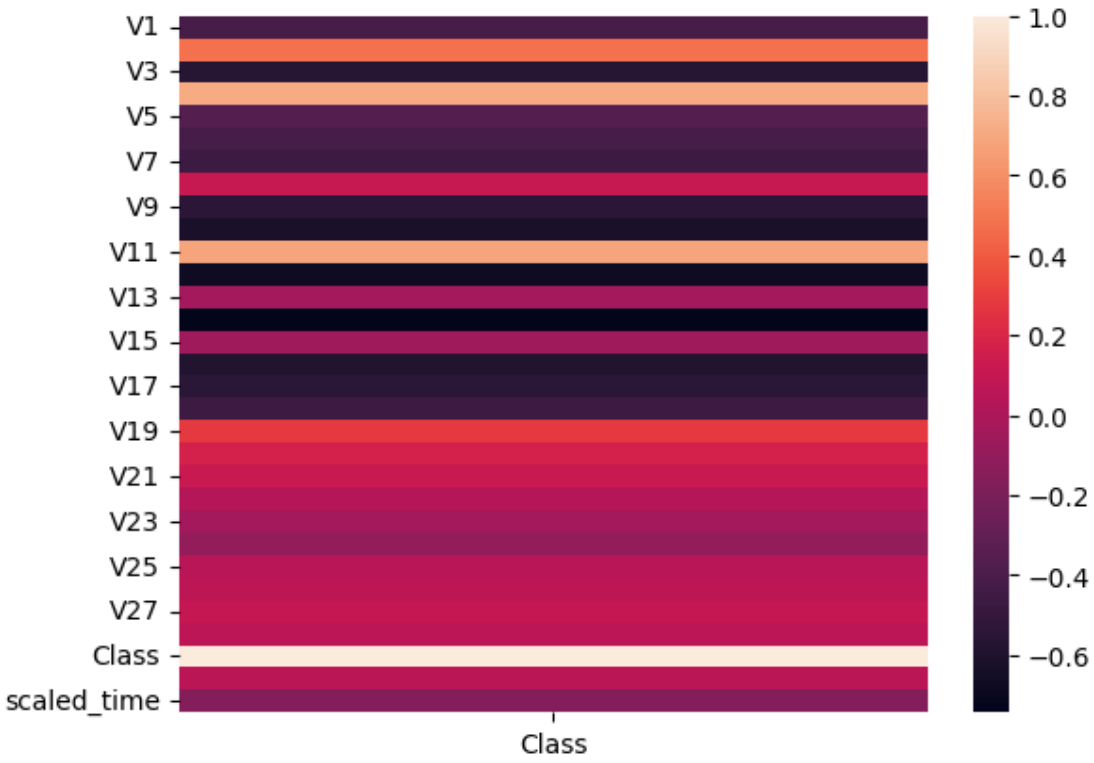
[39]:
```
sns.heatmap(new_df.corr())
```

[39]: <Axes: >



[40]:
```
sns.heatmap(new_df.corr()[['Class']])
```

[40]: <Axes: >



[41]:
```
correlation_with_class = new_df.corr()['Class']
sorted_correlation = correlation_with_class.sort_values(ascending = False)

sorted_correlation
```

[41]:
```
Class            1.000000
V4               0.712006
V11              0.684557
V2               0.481537
V19              0.283489
V20              0.170251
V21              0.120214
V8               0.111469
V27              0.102290
V26              0.067344
V28              0.065633
scaled_amount    0.057612
V25              0.049034
V22              0.039081
V13             -0.032680
V23             -0.041518
V15             -0.049737
```

```
V24             -0.099929
scaled_time     -0.161796
V5              -0.360913
V1              -0.420694
V6              -0.423729
V18             -0.460328
V7              -0.469674
V9              -0.540416
V17             -0.547273
V3              -0.561393
V16             -0.587112
V10             -0.623313
V12             -0.680856
V14             -0.744570
Name: Class, dtype: float64
```

V4, V11, V2, V19 having positive correlation with class. V16, V10, V12, V14 having negative correlation with class.

[42]:
```
# train test split
X = new_df.drop('Class', axis = 1)
y = new_df['Class']
```

[43]:
```
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,␣
 ↪random_state=42)
```

[44]:
```
X_train.head()
```

[44]:

	V1	V2	V3	V4	V5	V6	V7	\
199896	-1.212682	-2.484824	-6.397186	3.670562	-0.863375	-1.855855	1.017732	\
30314	-2.044489	3.368306	-3.937111	5.623120	-3.079232	-1.253474	-5.778880	
116789	1.091495	0.103068	0.509610	1.023919	-0.347728	-0.238461	-0.108192	
234632	1.261324	2.726800	-5.435019	5.342759	1.447043	-1.442584	-0.898702	
208246	0.154291	2.077213	-2.251206	3.898550	3.380157	4.730897	-1.073033	

	V8	V9	V10	…	V21	V22	V23	
199896	-0.544704	-1.703378	-3.739659	…	1.396872	0.092073	-1.492882	\
30314	1.707428	-4.467103	-6.067798	…	1.483594	0.834311	-0.148486	
116789	0.136412	-0.070387	0.141683	…	0.039717	0.042894	0.016290	
234632	0.123062	-2.748496	-3.202436	…	0.209086	-0.425938	-0.154440	
208246	-1.555674	-2.361496	0.051794	…	-1.620212	-0.655294	0.430456	

	V24	V25	V26	V27	V28	scaled_amount	
199896	-0.204227	0.532511	-0.293871	0.212663	0.431095	18.258935	\
30314	0.001669	-0.038996	0.389526	1.300236	0.549940	-0.200111	

```
116789  0.178556  0.369578 -0.437591  0.024846  0.015469       0.041580
234632 -0.018820  0.632234  0.192922  0.468181  0.280486      -0.283827
208246  0.384708 -0.816623  0.035954  0.055785  0.048362      -0.223891

        scaled_time
199896     0.569863
30314     -0.573800
116789    -0.120615
234632     0.744601
208246     0.615131

[5 rows x 30 columns]
```

[45]:
```
X_train.shape
```

[45]:
```
(756, 30)
```

[46]:
```
X_test.shape
```

[46]:
```
(190, 30)
```

1.0.1 Supervised - Classifiers

[47]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import cross_val_score
```

[48]:
```
from sklearn.metrics import␣
 ↪accuracy_score,classification_report,confusion_matrix
from sklearn.model_selection import train_test_split,cross_val_predict
```

Random Forest

[49]:
```
random_forest = RandomForestClassifier(n_estimators=100, max_depth = 1)
random_forest.fit(X_train, y_train)

y_pred_rf = random_forest.predict(X_test)
random_forest.score(X_train, y_train)

random_forest_train = round(random_forest.score(X_train, y_train) * 100, 2)
random_forest_accuracy = round(accuracy_score(y_pred_rf, y_test) * 100, 2)

print("Training Accuracy    :",random_forest_train ,"%")
print("Model Accuracy Score :",random_forest_accuracy ,"%")
print("\033[1m------------------------------------------------------------\033[0m")
```

```
print("Classification_Report: \n",classification_report(y_test,y_pred_rf))
print("\033[1m-------------------------------------------------------\033[0m")
```

```
Training Accuracy    : 90.61 %
Model Accuracy Score : 95.79 %
-------------------------------------------------------
Classification_Report:
               precision    recall  f1-score   support

           0       0.92      1.00      0.96        95
           1       1.00      0.92      0.96        95

    accuracy                           0.96       190
   macro avg       0.96      0.96      0.96       190
weighted avg       0.96      0.96      0.96       190

-------------------------------------------------------
```

[50]:
```
from sklearn.model_selection import cross_val_score
cv_scores = cross_val_score(random_forest, X_train, y_train, cv=5,␣
 ↪scoring='accuracy')
print("Cross-Validation Scores:", cv_scores)
print("Average Accuracy:", cv_scores.mean())
```

```
Cross-Validation Scores: [0.91447368 0.93377483 0.88741722 0.88741722 0.9205298
]
Average Accuracy: 0.9087225514116417
```

SVM

[51]:
```
svm = SVC()
model3=svm.fit(X_train,y_train)
y_pred3=model3.predict(X_test)


print(f'classification report :{classification_report(y_test,y_pred3)}')
print('************************************')
print(f'confusion matrix :{confusion_matrix(y_test,y_pred3)}')
print('************************************')
print(f'accuracy score :{accuracy_score(y_test,y_pred3)}')
```

```
classification report :              precision    recall  f1-score   support

           0       0.94      1.00      0.97        95
           1       1.00      0.94      0.97        95

    accuracy                           0.97       190
   macro avg       0.97      0.97      0.97       190
```

```
weighted avg       0.97      0.97      0.97       190

************************************
confusion matrix :[[95  0]
 [ 6 89]]
************************************
accuracy score :0.968421052631579
```

[52]:
```
from sklearn.model_selection import cross_val_score
cv_scores = cross_val_score(svm, X_train, y_train, cv=5, scoring='accuracy')
print("Cross-Validation Scores:", cv_scores)
print("Average Accuracy:", round(cv_scores.mean(),2))
```

```
Cross-Validation Scores: [0.92105263 0.94039735 0.90066225 0.90066225
0.93377483]
Average Accuracy: 0.92
```

Logistic Regression

[53]:
```
model = LogisticRegression(penalty='l1', solver='liblinear')

# Fit the model on the training data
model.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9631578947368421
```

[54]:
```
from sklearn.model_selection import cross_val_score
cv_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
print("Cross-Validation Scores:", cv_scores)
print("Average Accuracy:", cv_scores.mean())
```

```
Cross-Validation Scores: [0.93421053 0.94701987 0.9205298  0.90728477
0.93377483]
Average Accuracy: 0.9285639595677937
```