Exercise 7

7.1 Problem Statement:

Use Naïve Bayes classifier to solve the credit card fraud detection problem over a skewed dataset.

7.2 Description of Machine Learning Algorithm:

A Naive Bayes classifier is a probabilistic machine learning model that's used for classification task. It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

7.3 Description of Data Set:

Title of the data set: Credit card fraud detection

The data represents credit card transactions that occurred over two days in September 2013 by European cardholders.

Each record is classified as normal (class "0") or fraudulent (class "1") and the transactions are heavily skewed towards normal. Specifically, there are 492 fraudulent credit card transactions out of a total of 284,807 transactions, which is a total of about 0.172% of all transactions.

7.4 Data Preprocessing and Exploratory Data Analysis:

Data preprocessing is the process of transforming raw data into an understandable format. It is also an important step in data mining as we cannot work with raw data. The quality of the data should be checked before applying machine learning or data mining algorithms.

Major Tasks in Data Preprocessing:

1. Data cleaning
2. Data integration
3. Data reduction
4. Data transformation

Exploratory data analysis (EDA) is a technique that data professionals can use to understand a dataset before they start to model it. Some people refer to EDA as data exploration. The goal of conducting EDA is to determine the characteristics of the dataset. Conducting EDA can help data analysts make predictions and assumptions about data. Often, EDA involves data visualization, including creating graphs like histograms, scatter plots and box plots.

Major Tasks in EDA:

1. Observe your dataset
2. Find any missing values
3. Categorize your values
4. Find the shape of your dataset
5. Identify relationships in your dataset
6. Locate any outliers in your dataset

7.5 Machine Learning Package Used for Model building:

For classification of model we use scikit-learn

Scikit-learn is an open source Machine Learning Python package that offers functionality supporting supervised and unsupervised learning. Additionally, it provides tools for model development, selection and evaluation as well as many other utilities including data pre-processing functionality.

More specifically, scikit-learn's main functionality includes classification, regression, clustering, dimensionality reduction, model selection and pre-processing. sThe library is very simple to use and most importantly efficient as it is built on **NumPy**, **SciPy** and **matplotlib.**

Neural networks are a machine learning method inspired by how the human brain works. They are particularly good at doing pattern recognition and classification tasks, often using images as inputs. They're a well established machine learning technique that has been around since the 1950s but have gone through several iterations since that have overcome fundamental limitations of the previous one. The current state of the art neural networks are often referred to as deep learning.

7.6 Implementation:

```
from google.colab import drive
drive.mount('/content/drive')
Mounted at /content/drive
```

Data Analysis Importing the libraries

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import recall_score, precision_score, accuracy_score, f1_s
core
import scipy.stats as stats
%matplotlib inline
plt.style.use('ggplot')
```

Loading dataset

```
# Data Handling: Load CSV
df = pd.read_csv("/content/drive/MyDrive/ML Lab dataset/creditcard.csv")

# get to know list of features, data shape, stat. description.
print(df.shape)

print("First 5 lines:")
print(df.head(5))

print("describe: ")
print(df.describe())

print("info: ")
```

```
print(df.info())
(284807, 31)
First 5 lines:
   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
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9  ...       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
describe:
                Time            V1            V2            V3            V4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean    94813.859575  3.918649e-15  5.682686e-16 -8.761736e-15  2.811118e-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  -1.552103e-15  2.040130e-15 -1.698953e-15 -1.893285e-16 -3.147640e-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   ...  1.473120e-16  8.042109e-16  5.282512e-16  4.456271e-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   1.426896e-15  1.701640e-15 -3.662252e-16 -1.217809e-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]
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
None
```

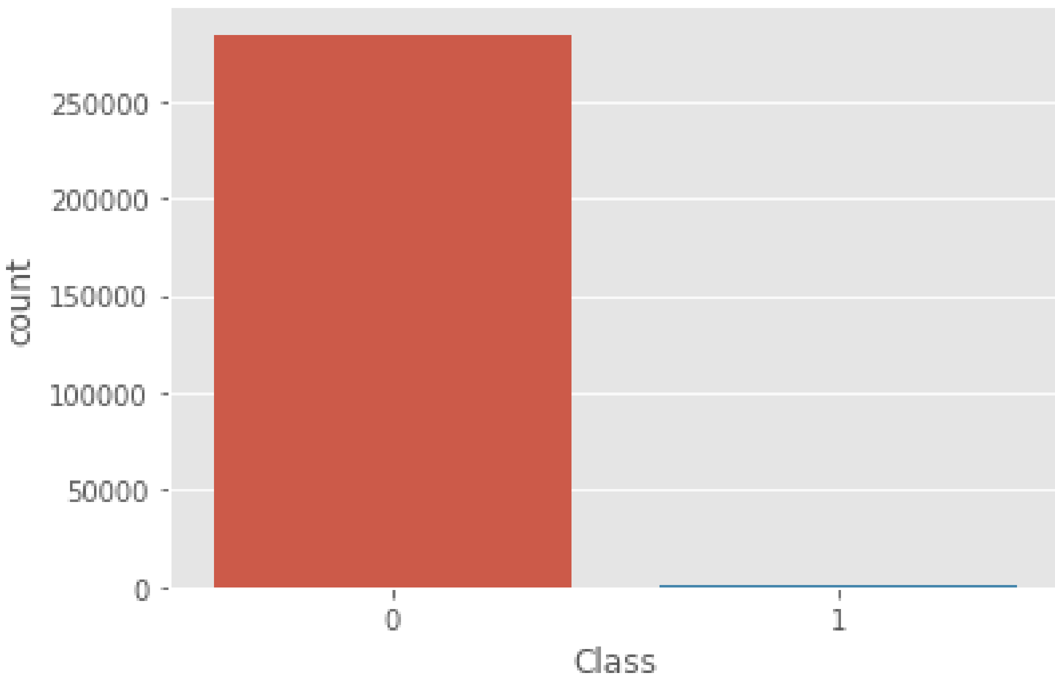
```
df.columns # Column names
```

```
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')
```

Exploratory Data Analysis

Analysis of Target Variable

```
sns.countplot(data=df, x='Class')
```



Number of Genuine and Fraud Transactions

```
fraud = df[df['Class']==1]
genuine = df[df['Class']==0]
perc_genuine = (len(genuine)/(len(genuine)+len(fraud)))*100
print('Number of Genuine Transactions = {} and the percentage of genuine transa
ctions = {:.3f} %'.format(len(genuine),perc_genuine))
```

```
Number of Genuine Transactions = 284315 and the percentage of genuine
transactions = 99.827 %
```

```
perc_fraud = (len(fraud)/(len(genuine)+len(fraud)))*100
print('Number of fraud Transactions = {} and the percentage of fraud transactio
ns = {:.3f} %'.format(len(fraud),perc_fraud))
```

```
Number of fraud Transactions = 492 and the percentage of fraud transactions =
0.173 %
```

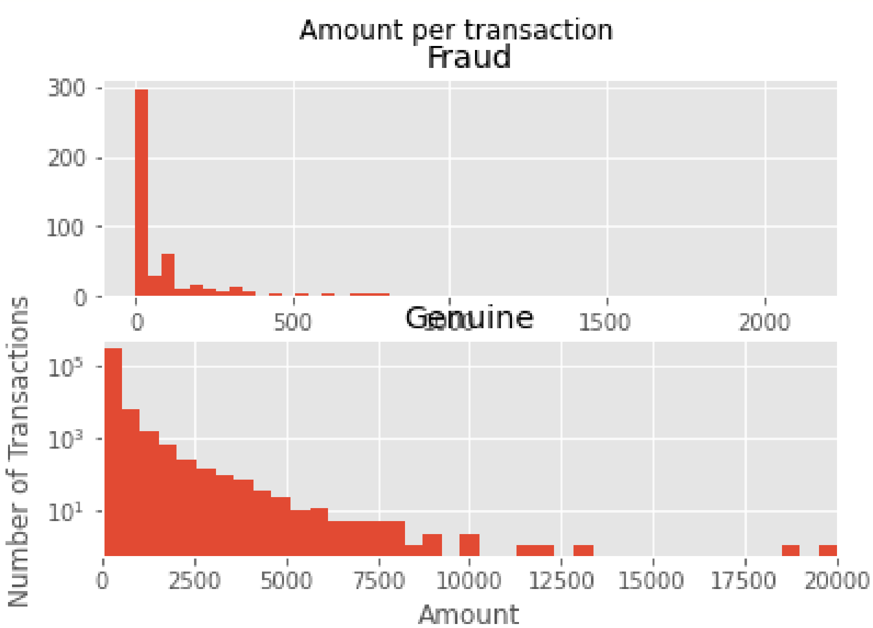
Data is highly imbalanced

```
fraud.Amount.describe()
```

```
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
```

```
genuine.Amount.describe()
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
```

```
plot,(axis1, axis2) = plt.subplots(2, 1)
axis1.hist(fraud.Amount, bins = 50)
axis1.set_title('Fraud')
axis2.hist(genuine.Amount, bins = 50)
axis2.set_title('Genuine')
plt.xlabel('Amount')
plt.ylabel('Number of Transactions')
plt.xlim((0, 20000))
plot.suptitle('Amount per transaction')
plt.yscale('log')
plt.show()
```

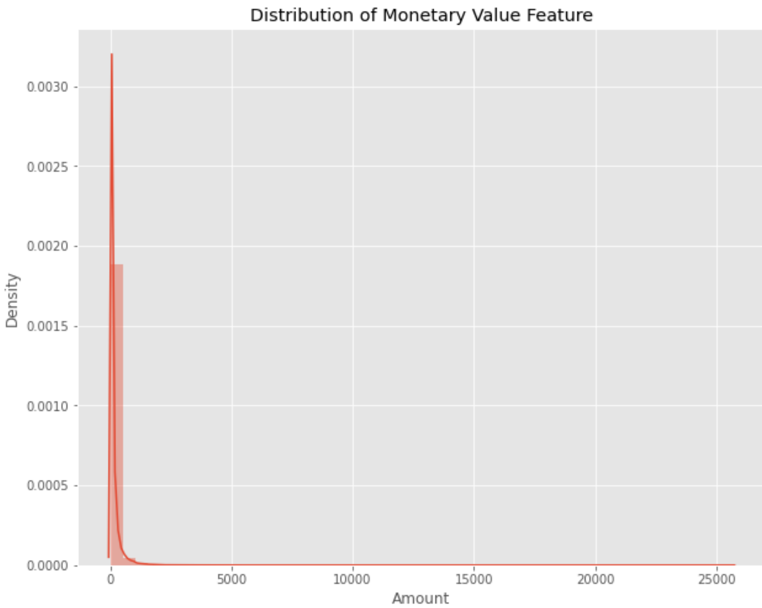


Fraud transactions are minimal

Analyzing Features

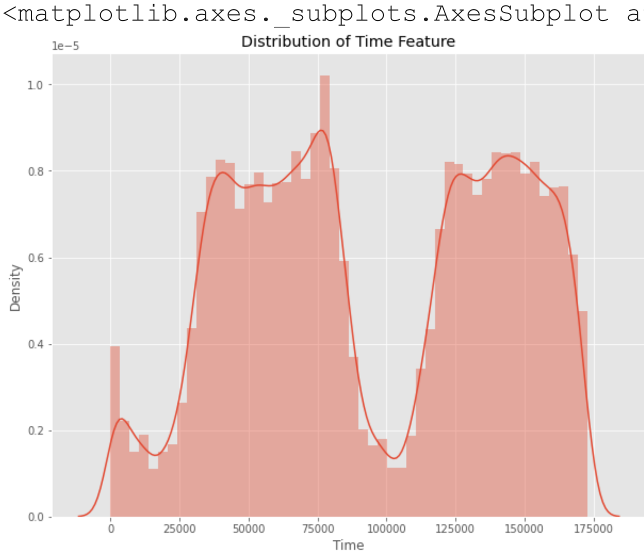
```
plt.figure(figsize=(10,8))
plt.title('Distribution of Monetary Value Feature')
sns.distplot(df.Amount)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level function
with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f715499c3d0>
```




```
plt.figure(figsize=(10,8))
plt.title('Distribution of Time Feature')
sns.distplot(df.Time)
```

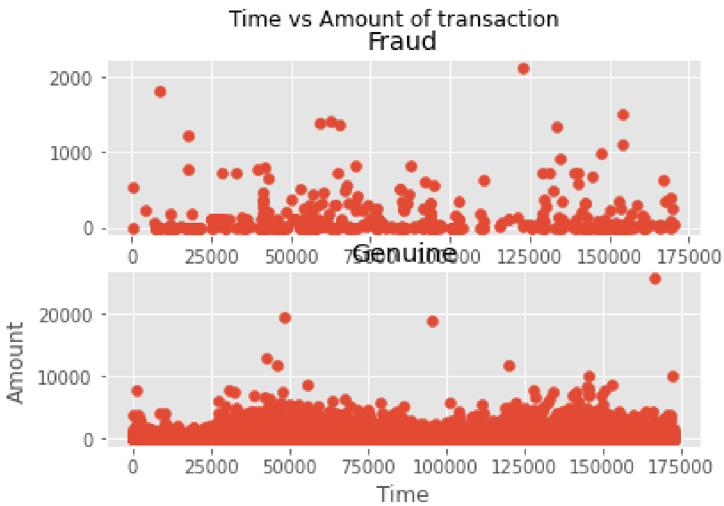
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level function
with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f7152062510>
```



Visualizing time with respect to class

```
plot, (axis1, axis2) = plt.subplots(2, 1)
axis1.scatter(fraud.Time, fraud.Amount)
axis1.set_title('Fraud')
axis2.scatter(genuine.Time, genuine.Amount)
axis2.set_title('Genuine')
plt.xlabel('Time')
plt.ylabel('Amount')
plot.suptitle('Time vs Amount of transaction')
#plt.show();
```

```
Text(0.5, 0.98, 'Time vs Amount of transaction')
```



Time of transaction has no effect

Correlation

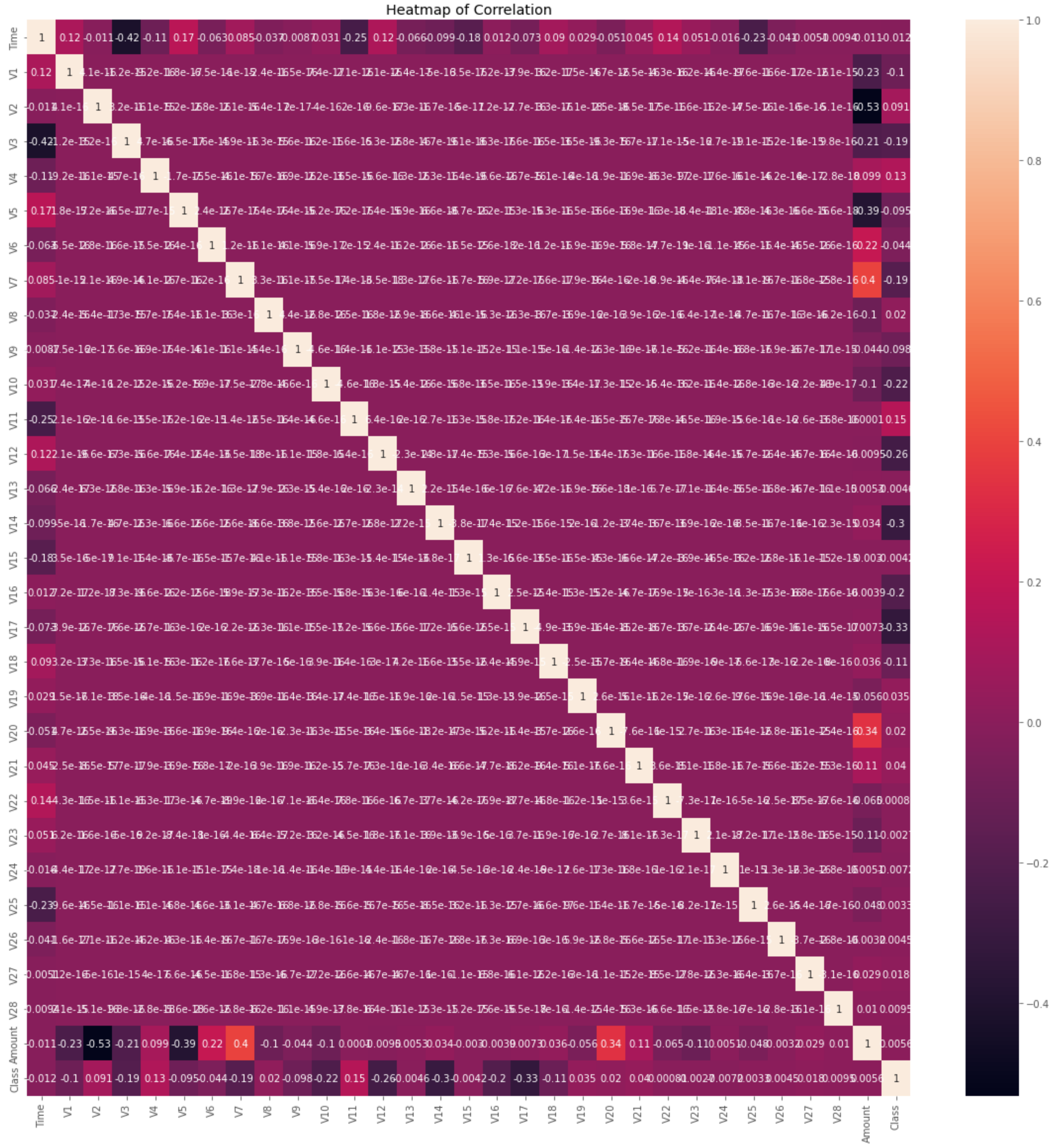
Visualizing correlation of all the features

```
corr = df.corr()
corr
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
Time	1.000000	1.173963e-01	-1.059333e-02	-4.196182e-01	-1.052602e-01	1.730721e-01	-6.301647e-02	8.471437e-02	-3.694943e-02	-8.660434e-03
V1	0.117396	1.000000e+00	4.135835e-16	-1.227819e-15	-9.215150e-16	1.812612e-17	-6.506567e-16	-1.005191e-15	-2.433822e-16	-1.513678e-16
V2	-0.010593	4.135835e-16	1.000000e+00	3.243764e-16	-1.121065e-15	5.157519e-16	2.787346e-16	2.055934e-16	-5.377041e-17	1.978488e-17
V3	-0.419618	-1.227819e-15	3.243764e-16	1.000000e+00	4.711293e-16	-6.539009e-17	1.627627e-15	4.895305e-16	-1.268779e-15	5.568367e-16
V4	-0.105260	-9.215150e-16	-1.121065e-15	4.711293e-16	1.000000e+00	-1.719944e-15	-7.491959e-16	-4.104503e-16	5.697192e-16	6.923247e-16
V5	0.173072	1.812612e-17	5.157519e-16	-6.539009e-17	-1.719944e-15	1.000000e+00	2.408382e-16	2.715541e-16	7.437229e-16	7.391702e-16
V6	-0.063016	-6.506567e-16	2.787346e-16	1.627627e-15	-7.491959e-16	2.408382e-16	1.000000e+00	1.191668e-16	-1.104219e-16	4.131207e-16
V7	0.084714	-1.005191e-15	2.055934e-16	4.895305e-16	-4.104503e-16	2.715541e-16	1.191668e-16	1.000000e+00	3.344412e-16	1.122501e-15
V8	-0.036949	-2.433822e-16	-5.377041e-17	-1.268779e-15	5.697192e-16	7.437229e-16	-1.104219e-16	3.344412e-16	1.000000e+00	4.356078e-16
V9	-0.008660	-1.513678e-16	1.978488e-17	5.568367e-16	6.923247e-16	7.391702e-16	4.131207e-16	1.122501e-15	4.356078e-16	1.000000e+00
V10	0.030617	7.388135e-17	-3.991394e-16	1.156587e-15	2.232685e-16	-5.202306e-16	5.932243e-17	-7.492834e-17	-2.801370e-16	-4.642274e-16
V11	-0.247689	2.125498e-16	1.975426e-16	1.576830e-15	3.459380e-16	7.203963e-16	1.980503e-15	1.425248e-16	2.487043e-16	1.354680e-16
V12	0.124348	2.053457e-16	-9.568710e-17	6.310231e-16	-5.625518e-16	7.412552e-16	2.375468e-16	-3.536655e-18	1.839891e-16	-1.079314e-15
V13	-0.065902	-2.425603e-17	6.295388e-16	2.807652e-16	1.303306e-16	5.886991e-16	-1.211182e-16	1.266462e-17	-2.921856e-16	2.251072e-15
V14	-0.098757	-5.020280e-16	-1.730566e-16	4.739859e-16	2.282280e-16	6.565143e-16	2.621312e-16	2.607772e-16	-8.599156e-16	3.784757e-15
V15	-0.183453	3.547782e-16	-4.995814e-17	9.068793e-16	1.377649e-16	-8.720275e-16	-1.531188e-15	-1.690540e-16	4.127777e-16	-1.051167e-15
V16	0.011903	7.212815e-17	1.177316e-17	8.299445e-16	-9.614528e-16	2.246261e-15	2.623672e-18	5.869302e-17	-5.254741e-16	-1.214086e-15
V17	-0.073297	-3.879840e-16	-2.685296e-16	7.614712e-16	-2.699612e-16	1.281914e-16	2.015618e-16	2.177192e-16	-2.269549e-16	1.113695e-15
V18	0.090438	3.230206e-17	3.284605e-16	1.509897e-16	-5.103644e-16	5.308590e-16	1.223814e-16	7.604126e-17	-3.667974e-16	4.993240e-16
V19	0.028975	1.502024e-16	-7.118719e-18	3.463522e-16	-3.980557e-16	-1.450421e-16	-1.865597e-16	-1.881008e-16	-3.875186e-16	-1.376135e-16
V20	-0.050866	4.654551e-16	2.506675e-16	-9.316409e-16	-1.857247e-16	-3.554057e-16	-1.858755e-16	9.379684e-16	2.033737e-16	-2.343720e-16
V21	0.044736	-2.457409e-16	-8.480447e-17	5.706192e-17	-1.949553e-16	-3.920976e-16	5.833316e-17	-2.027779e-16	3.892798e-16	1.936953e-16
V22	0.144059	-4.290944e-16	1.526333e-16	-1.133902e-15	-6.276051e-17	1.253751e-16	-4.705235e-19	-8.898922e-16	2.026927e-16	-7.071869e-16
V23	0.051142	6.168652e-16	1.634231e-16	-4.983035e-16	9.164206e-17	-8.428683e-18	1.046712e-16	-4.387401e-16	6.377260e-17	-5.214137e-16
V24	-0.016182	-4.425156e-17	1.247925e-17	2.686834e-19	1.584638e-16	-1.149255e-15	-1.071589e-15	7.434913e-18	-1.047097e-16	-1.430343e-16
V25	-0.233083	-9.605737e-16	-4.478846e-16	-1.104734e-15	6.070716e-16	4.808532e-16	4.562861e-16	-3.094082e-16	-4.653279e-16	6.757763e-16
V26	-0.041407	-1.581290e-17	2.057310e-16	-1.238062e-16	-4.247268e-16	4.319541e-16	-1.357067e-16	-9.657637e-16	-1.727276e-16	-7.888853e-16
V27	-0.005135	1.198124e-16	-4.966953e-16	1.045747e-15	3.977061e-17	6.590482e-16	-4.452461e-16	-1.782106e-15	1.299943e-16	-6.709655e-17
V28	-0.009413	2.083082e-15	-5.093836e-16	9.775546e-16	-2.761403e-18	-5.613951e-18	2.594754e-16	-2.776530e-16	-6.200930e-16	1.110541e-15

	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
Time	...	4.473573e-02	1.440591e-01	5.114236e-02	-1.618187e-02	-2.330828e-01	-4.140710e-02	-5.134591e-03	-9.412688e-03	-0.010596	-0.012323
V1	...	-2.457409e-16	-4.290944e-16	6.168652e-16	-4.425156e-17	-9.605737e-16	-1.581290e-17	1.198124e-16	2.083082e-15	-0.227709	-0.101347
V2	...	-8.480447e-17	1.526333e-16	1.634231e-16	1.247925e-17	-4.478846e-16	2.057310e-16	-4.966953e-16	-5.093836e-16	-0.531409	0.091289
V3	...	5.706192e-17	-1.133902e-15	-4.983035e-16	2.686834e-19	-1.104734e-15	-1.238062e-16	1.045747e-15	9.775546e-16	-0.210880	-0.192961
V4	...	-1.949553e-16	-6.276051e-17	9.164206e-17	1.584638e-16	6.070716e-16	-4.247268e-16	3.977061e-17	-2.761403e-18	0.098732	0.133447
V5	...	-3.920976e-16	1.253751e-16	-8.428683e-18	-1.149255e-15	4.808532e-16	4.319541e-16	6.590482e-16	-5.613951e-18	-0.386356	-0.094974
V6	...	5.833316e-17	-4.705235e-19	1.046712e-16	-1.071589e-15	4.562861e-16	-1.357067e-16	-4.452461e-16	2.594754e-16	0.215981	-0.043643
V7	...	-2.027779e-16	-8.898922e-16	-4.387401e-16	7.434913e-18	-3.094082e-16	-9.657637e-16	-1.782106e-15	-2.776530e-16	0.397311	-0.187257
V8	...	3.892798e-16	2.026927e-16	6.377260e-17	-1.047097e-16	-4.653279e-16	-1.727276e-16	1.299943e-16	-6.200930e-16	-0.103079	0.019875
V9	...	1.936953e-16	-7.071869e-16	-5.214137e-16	-1.430343e-16	6.757763e-16	-7.888853e-16	-6.709655e-17	1.110541e-15	-0.044246	-0.097733
V10	...	1.177547e-15	-6.418202e-16	3.214491e-16	-1.355885e-16	-2.846052e-16	-3.028119e-16	-2.197977e-16	4.864782e-17	-0.101502	-0.216883
V11	...	-5.658364e-16	7.772895e-16	-4.505332e-16	1.933267e-15	-5.600475e-16	-1.003221e-16	-2.640281e-16	-3.792314e-16	0.000104	0.154876
V12	...	7.300527e-16	1.644699e-16	1.800885e-16	4.436512e-16	-5.712973e-16	-2.359969e-16	-4.672391e-16	6.415167e-16	-0.009542	-0.260593
V13	...	1.008461e-16	6.747721e-17	-7.132064e-16	-1.397470e-16	-5.497612e-16	-1.769255e-16	-4.720898e-16	1.144372e-15	0.005293	-0.004570
V14	...	-3.356561e-16	3.740383e-16	3.883204e-16	2.003482e-16	-8.547932e-16	-1.660327e-16	1.044274e-16	2.289427e-15	0.033751	-0.302544
V15	...	6.605263e-17	-4.208921e-16	-3.912243e-16	-4.478263e-16	3.206423e-16	2.817791e-16	-1.143519e-15	-1.194130e-15	-0.002986	-0.004223
V16	...	-4.715090e-16	-7.923387e-17	5.020770e-16	-3.005985e-16	-1.345418e-15	-7.290010e-16	6.789513e-16	7.588849e-16	-0.003910	-0.196539
V17	...	-8.230527e-16	-8.743398e-16	3.706214e-16	-2.403828e-16	2.666806e-16	6.932833e-16	6.148525e-16	-5.534540e-17	0.007309	-0.326481
V18	...	-9.408680e-16	-4.819365e-16	-1.912006e-16	-8.986916e-17	-6.629212e-17	2.990167e-16	2.242791e-16	7.976796e-16	0.035650	-0.111485
V19	...	5.115885e-16	-1.163768e-15	7.032035e-16	2.587708e-17	9.577163e-16	5.898033e-16	-2.959370e-16	-1.405379e-15	-0.056151	0.034783
V20	...	-7.614597e-16	1.009285e-15	2.712885e-16	1.277215e-16	1.410054e-16	-2.803504e-16	-1.138829e-15	-2.436795e-16	0.339403	0.020090
V21	...	1.000000e+00	3.649908e-15	8.119580e-16	1.761054e-16	-1.686082e-16	-5.557329e-16	-1.211281e-15	5.278775e-16	0.105999	0.040413
V22	...	3.649908e-15	1.000000e+00	-7.303916e-17	9.970809e-17	-5.018575e-16	-2.503187e-17	8.461337e-17	-6.627203e-16	-0.064801	0.000805
V23	...	8.119580e-16	-7.303916e-17	1.000000e+00	2.130519e-17	-8.232727e-17	1.114524e-15	2.839721e-16	1.481903e-15	-0.112633	-0.002685
V24	...	1.761054e-16	9.970809e-17	2.130519e-17	1.000000e+00	1.015391e-15	1.343722e-16	-2.274142e-16	-2.819805e-16	0.005146	-0.007221
V25	...	-1.686082e-16	-5.018575e-16	-8.232727e-17	1.015391e-15	1.000000e+00	2.646517e-15	-6.406679e-16	-7.008939e-16	-0.047837	0.003308
V26	...	-5.557329e-16	-2.503187e-17	1.114524e-15	1.343722e-16	2.646517e-15	1.000000e+00	-3.667715e-16	-2.782204e-16	-0.003208	0.004455
V27	...	-1.211281e-15	8.461337e-17	2.839721e-16	-2.274142e-16	-6.406679e-16	-3.667715e-16	1.000000e+00	-3.061287e-16	0.028825	0.017580
V28	...	5.278775e-16	-6.627203e-16	1.481903e-15	-2.819805e-16	-7.008939e-16	-2.782204e-16	-3.061287e-16	1.000000e+00	0.010258	0.009536

```
#heatmap
corr = df.corr()
plt.figure(figsize=(20,20))
heat = sns.heatmap(data=corr,annot=True)
plt.title('Heatmap of Correlation')
```



Data Preprocessing

Missing data

```
df.isnull().values.any()
```

```
False
```

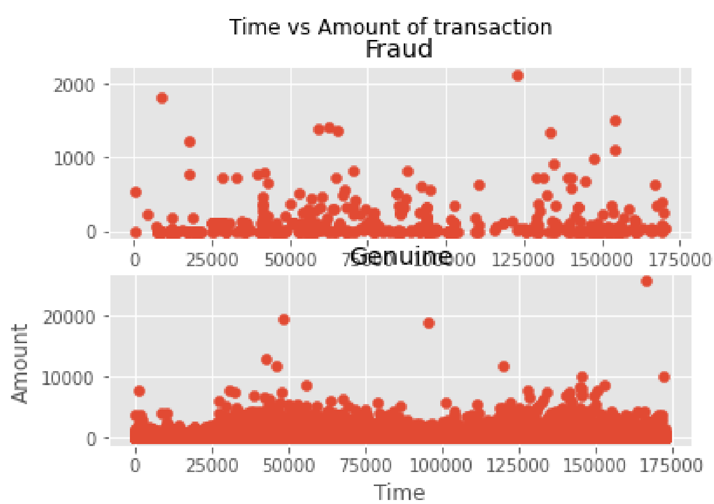
Feature Scaling

```
scaler = StandardScaler()
scaler2 = StandardScaler()
#scaling time
scaled_time = scaler.fit_transform(df[['Time']])
flat_list1 = [item for sublist in scaled_time.tolist() for item in sublist]
scaled_time = pd.Series(flat_list1)
#scaling the amount column
scaled_amount = scaler2.fit_transform(df[['Amount']])
flat_list2 = [item for sublist in scaled_amount.tolist() for item in sublist]
scaled_amount = pd.Series(flat_list2)
#concatenating newly created columns w original df
df = pd.concat([df, scaled_amount.rename('scaled_amount'), scaled_time.rename('
scaled_time')], axis=1)
df.sample(5)
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V23	V24	V25	V26	V27	V28	Amount	Class	scaled_amount	scaled_time
98975	66903.0	1.289482	-1.582951	0.348539	-1.520838	-1.581663	0.005741	-1.281381	0.079642	-1.673657	...	-0.206424	-0.524138	0.347005	-0.078579	0.019208	0.026865	128.00	0	0.158526	-0.587745
153407	98938.0	-0.632977	-0.486623	1.385542	-0.970855	-0.044483	0.426488	0.734698	-0.521861	0.497694	...	-0.686892	-0.419792	1.055355	0.507998	-0.419325	-0.406969	139.00	0	0.202505	0.086846
89025	62390.0	0.784985	-0.690212	0.894514	1.299230	-0.811195	0.746248	-0.606867	0.396667	0.633693	...	-0.242930	-0.315704	0.346549	-0.223209	0.041405	0.045083	170.00	0	0.326445	-0.682779
135501	81282.0	1.192938	0.119649	0.035486	1.212958	0.221914	0.292477	0.057719	0.044891	0.379221	...	-0.177587	-0.763538	0.761675	-0.261159	0.034483	0.009699	24.19	0	-0.256516	-0.284953
168808	119393.0	2.090275	-0.455652	-1.876569	-1.454087	-0.174865	-1.854779	0.400016	-0.505552	1.495054	...	0.002438	0.033655	0.278185	-0.108404	-0.041460	-0.066570	30.00	0	-0.233287	0.517586

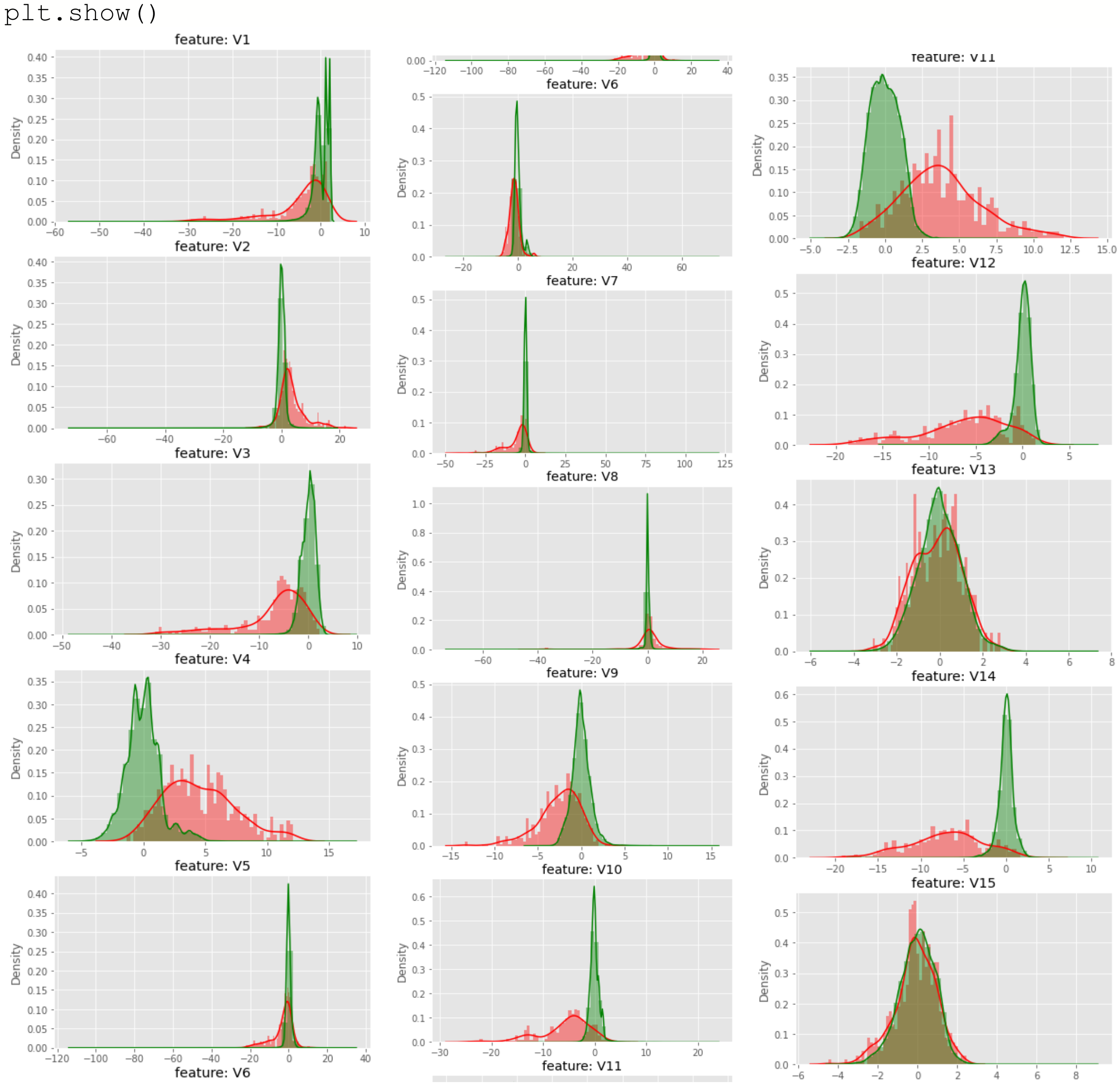
5 rows × 33 columns

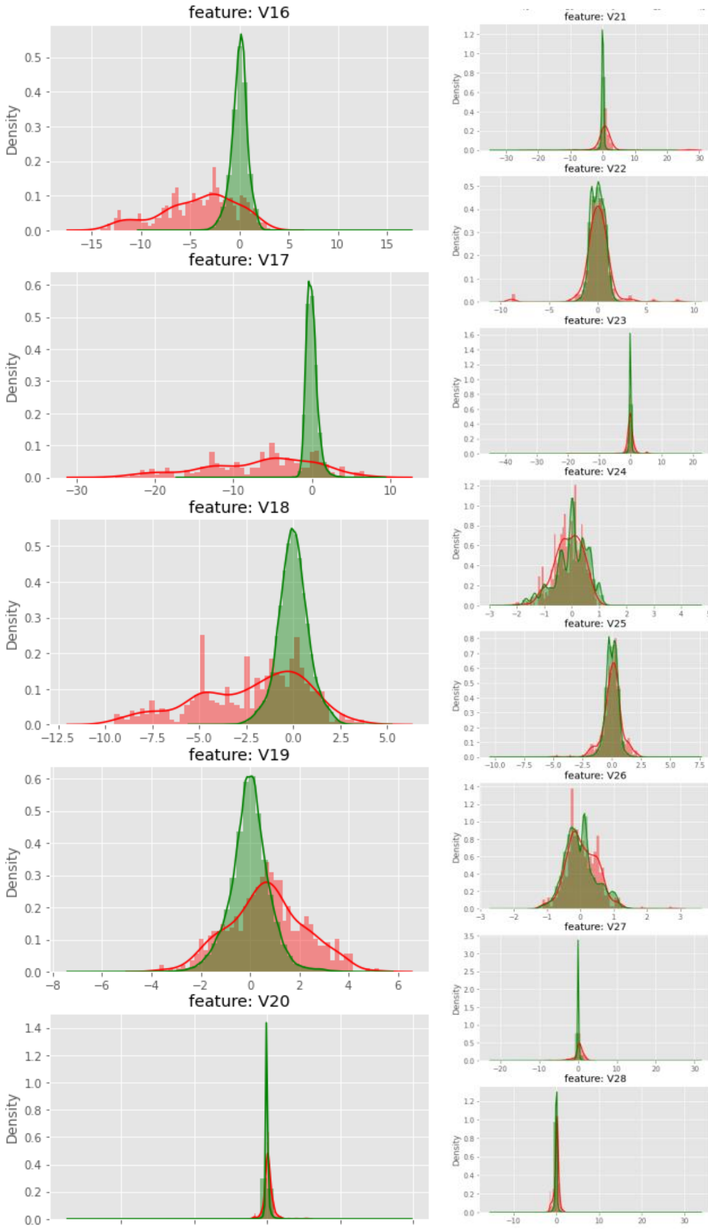
```
plot, (axis1, axis2) = plt.subplots(2, 1)
axis1.scatter(fraud.Time, fraud.Amount)
axis1.set_title('Fraud')
axis2.scatter(genuine.Time, genuine.Amount)
axis2.set_title('Genuine')
plt.xlabel('Time')
plt.ylabel('Amount')
plot.suptitle('Time vs Amount of transaction')
#plt.show();
Text(0.5, 0.98, 'Time vs Amount of transaction')
```



By seeing the graph we can say that they have nothing much information to prove that transaction is fraud. So it is not having any prediction power coz of it we can simply drop the time variable from dataset.

```
#dropping old amount and time columns
df.drop(['Amount', 'Time'], axis=1, inplace=True)
#let us check correlations and shapes of those 25 principal components.
# Features V1, V2, ... V28 are the principal components obtained with PCA.
import seaborn as sns
import matplotlib.gridspec as gridspec
gs = gridspec.GridSpec(28, 1)
plt.figure(figsize=(6,28*4))
for i, col in enumerate(df[df.iloc[:,0:28].columns]):
    ax5 = plt.subplot(gs[i])
    sns.distplot(df[col][df.Class == 1], bins=50, color='r')
    sns.distplot(df[col][df.Class == 0], bins=50, color='g')
    ax5.set_xlabel('')
    ax5.set_title('feature: ' + str(col))
plt.show()
```





Split data into train and test set

```
from sklearn.model_selection import train_test_split
y = df['Class'].values #target
X = df.drop(['Class'],axis=1).values #features
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=42, stratify=y
)
print("train-set size: ", len(y_train),
      "\ntest-set size: ", len(y_test))
print("fraud cases in test-set: ", sum(y_test))
train-set size:  227845
test-set size:  56962
```

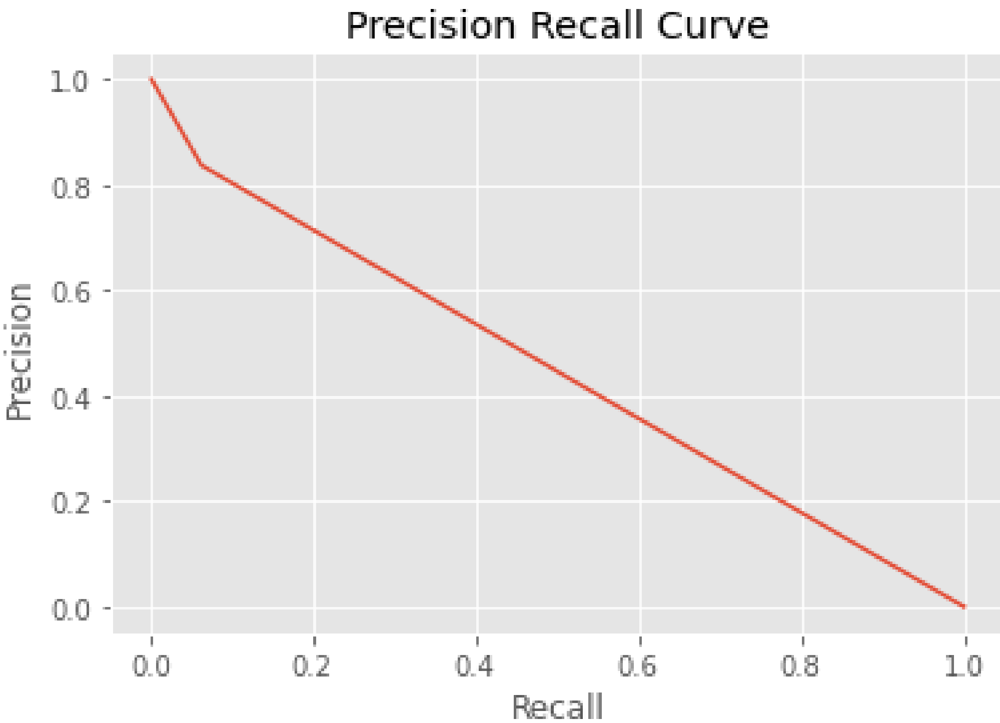
```
fraud cases in test-set:  98
```

```
# splitting dataset to train & test dataset
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=0)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
(199364, 30)
(85443, 30)
(199364,)
(85443,)
```

Model Building using Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve,auc,precision_recall_curve,roc_curve
NBclf=GaussianNB()
NBclf.fit(X_train,y_train)
NB_pred,NB_pred_prob=NBclf.predict(X_test),NBclf.predict_proba(X_test)
acc_score = NBclf.score(X_test, y_test)
print(f'Accuracy of model on test dataset :- {acc_score}')
# predict result using test dataset
y_pred = NBclf.predict(X_test)
# confusion matrix
print(f"Confusion Matrix :- \n {confusion_matrix(y_test, y_pred)}")
# classification report for f1-score
print(f"Classification Report :- \n {classification_report(y_test, y_pred)}")

precision, recall, thresholds = precision_recall_curve(y_test, y_pred)

# Plot ROC curve
plt.plot(precision, recall)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision Recall Curve')
plt.show()
Accuracy of model on test dataset :- 0.9784651756141521
Confusion Matrix :-
 [[83480  1816]
 [   24   123]]
Classification Report :-
               precision    recall  f1-score   support

           0       1.00      0.98      0.99     85296
           1       0.06      0.84      0.12       147

    accuracy                           0.98     85443
```

```
   macro avg       0.53      0.91      0.55     85443
weighted avg       1.00      0.98      0.99     85443
```



7.7 Results and Discussion:

Naïve Bayes classifier is used to solve the credit card fraud detection problem over a skewed dataset. The model is 98% accurate.