CREDIT CARD FRAUD DETECTION SYSTEM

GROUP-19

GROUP MEMBERS:

Advaith Dev Krishnan -AM.EN.U4CSE20003

Divya Bisht -AM.EN.U4CSE20021

Hari Chandan -AM.EN.U4CSE20034

Likhith Reddy -AM.EN.U4CSE20039

Rithwik Sai -AM.EN.U4CSE20057

Importing the Dependencies

In [3]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

In [4]:

```
# loading the dataset to a pandas dataframe
credit_card_data = pd.read_csv("C:/Users/haric/(S-5)/-Courses/data science/DS project/creditcard.csv")
```

In [5]:

```
# first five rows of the dataset
credit_card_data.head()
```

Out[5]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.

5 rows × 31 columns

In [6]:

```
#last five rows of the dataset
credit_card_data.tail()
```

Out[6]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014480	-0.50
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.012463	-1.01
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.037501	0.64
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.163298	0.12
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.376777	0.00

5 rows × 31 columns

In [7]:

```
# some info about the data
credit_card_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
```

In [8]:

```
#checking the number of missing values in each column
credit_card_data.isnull().sum()
```

Out[8]:

```
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 [9]:

```
# Distribution of legit transaction and fraudulent transaction
credit_card_data['Class'].value_counts()
```

Out[9]:

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

Dataset is highly unbalanced

0--> Normal Transaction

1-->Fraudulent Transaction

In [10]:

```
#separating the data for analysis
legit = credit_card_data[credit_card_data.Class == 0]
fraud = credit_card_data[credit_card_data.Class == 1]
```

In [11]:

```
legit.shape
```

Out[11]:

```
(284315, 31)
```

In [12]:

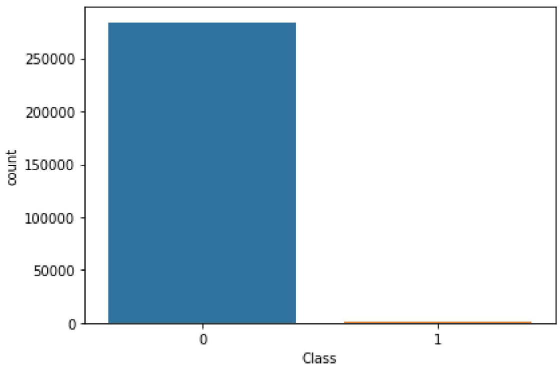
```
fraud.shape
```

Out[12]:

```
(492, 31)
```

In [13]:

```
import matplotlib.pyplot as plt
import seaborn as sns
fig, ax=plt.subplots(figsize=(6, 4))
ax=sns.countplot(x='Class', data=credit_card_data)
plt.tight_layout()
```



In [14]:

```
#statistical measures of the data for each type of class.
legit.Amount.describe()
```

Out[14]:

```
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
```

In [15]:

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

Out[15]:

```
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 [16]:

```
# compare the values for both transaction
credit_card_data.groupby('Class').mean()
```

Out[16]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	
Class															
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.000987	0.004467	...	-0.000644	-0.001235	-0.000024	
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	...	0.372319	0.713588	0.014049	-

2 rows × 30 columns

Under-Sampling to resolve the problem of unbalanced dataset

Build a sample dataset containg similar distribution of normal transaction and Fradulent Transaction

we would randomly take 492 data from legit dataset and we would take the fraud dataset completely and then use them to train the model

Number of Fraudulent Transaction = 492

In [17]:

```
legit_sample = legit.sample(n=492)
```

Concatenating two Dataframes using 'concat' function.

In [18]:

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

In [19]:

```
# first five rows of the dataset
new_dataset.head()
```

Out[19]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	
173202	121384.0	1.959801	0.016050	-1.390979	1.156715	0.372663	-0.522305	0.234468	-0.225080	0.272641	...	0.287826	0.845287	-0.016184	0.61
7619	10540.0	1.320260	0.479284	0.110144	0.697217	0.146416	-0.544057	0.005614	-0.257043	1.309131	...	-0.488651	-1.133276	0.032352	-0.5
279855	169135.0	-2.475069	1.708159	-0.362032	-1.384144	0.475650	0.517683	0.365589	-2.176044	-0.338360	...	2.137680	-1.177631	0.069729	0.17
271985	164855.0	2.013716	0.439072	-2.498592	1.265654	1.248825	-0.390429	0.541139	-0.141226	-0.057473	...	-0.018697	0.099682	-0.056729	0.0
72054	54548.0	-1.130235	-0.396668	1.326885	-1.521651	-0.239949	-1.149648	0.160416	0.166297	-1.204608	...	-0.034319	-0.122656	0.113664	0.3

5 rows × 31 columns

In [20]:

```python
#last five rows of  the dataset
new_dataset.tail()
```

Out[20]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.697211	-2.064945	...	0.778584	-0.319189	0.639419	-0.29
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.248525	-1.127396	...	0.370612	0.028234	-0.145640	-0.08
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.210158	-0.652250	...	0.751826	0.834108	0.190944	0.03
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.058733	-1.632333	...	0.583276	-0.269209	-0.456108	-0.18
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.068384	0.577829	...	-0.164350	-0.295135	-0.072173	-0.45

5 rows × 31 columns

Checking whether class types are equal or not.

In [21]:

```python
new_dataset['Class'].value_counts()
```

Out[21]:

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

In [22]:

```python
new_dataset.groupby('Class').mean()
```

Out[22]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	
Class															
0	92367.105691	0.076783	0.082529	0.149726	0.099438	-0.046944	-0.015630	0.078347	-0.014006	0.103854	...	-0.012634	-0.012197	-0.021440	0
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	...	0.372319	0.713588	0.014049	-0

2 rows × 30 columns

In [23]:

```
# Correlation
# Correlation is useful to find peers of input field so we are aware when building models, either to transform them (principal component)

# In this particular data, it is told that all the V-variables are already principal components. So correlation is not useful to inspect t

# But one thing we can do is to find some correlation between V-variables and Class. Recall now that coding Class into a integer data type

# code for correlation

new_dataset.corr()

# Splitting the data into features and target variable
X = new_dataset.drop(columns='Class',axis=1)
Y = new_dataset['Class']

print(X)
```

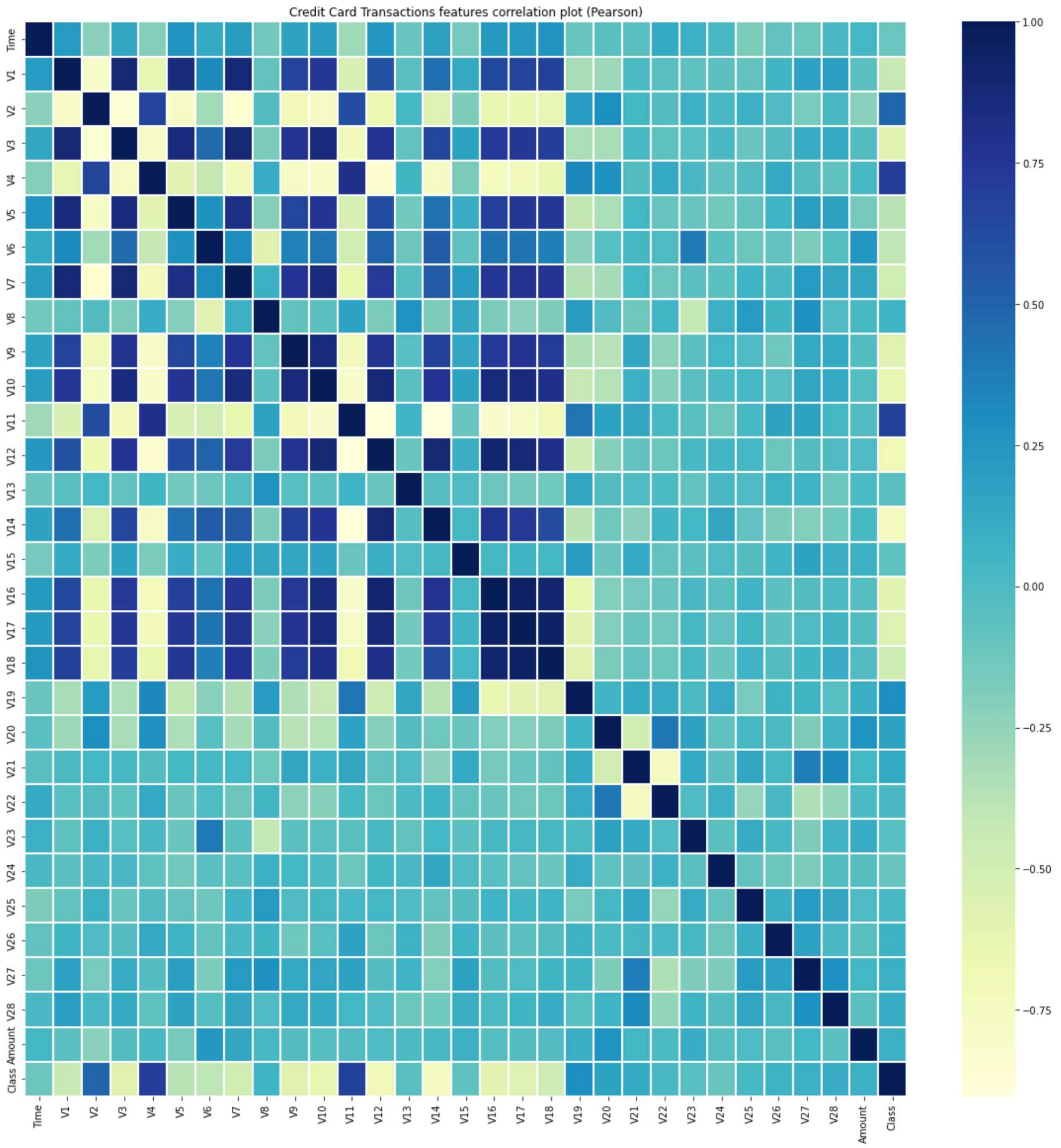
```
            Time        V1        V2        V3        V4        V5        V6  \
173202  121384.0  1.959801  0.016050 -1.390979  1.156715  0.372663 -0.522305
7619     10540.0  1.320260  0.479284  0.110144  0.697217  0.146416 -0.544057
279855  169135.0 -2.475069  1.708159 -0.362032 -1.384144  0.475650  0.517683
271985  164855.0  2.013716  0.439072 -2.498592  1.265654  1.248825 -0.390429
72054    54548.0 -1.130235 -0.396668  1.326885 -1.521651 -0.239949 -1.149648
...          ...       ...       ...       ...       ...       ...       ...
279863  169142.0 -1.927883  1.125653 -4.518331  1.749293 -1.566487 -2.010494
280143  169347.0  1.378559  1.289381 -5.004247  1.411850  0.442581 -1.326536
280149  169351.0 -0.676143  1.126366 -2.213700  0.468308 -1.120541 -0.003346
281144  169966.0 -3.113832  0.585864 -5.399730  1.817092 -0.840618 -2.943548
281674  170348.0  1.991976  0.158476 -2.583441  0.408670  1.151147 -0.096695

              V7        V8        V9  ...       V20       V21       V22  \
173202  0.234468 -0.225080  0.272641  ... -0.157721  0.287826  0.845287
7619    0.005614 -0.257043  1.309131  ... -0.071264 -0.488651 -1.133276
279855  0.365589 -2.176044 -0.338360  ... -1.057272  2.137680 -1.177631
271985  0.541139 -0.141226 -0.057473  ... -0.215777 -0.018697  0.099682
72054   0.160416  0.166297 -1.204608  ... -0.384260 -0.034319 -0.122656
...          ...       ...       ...  ...       ...       ...       ...
279863 -0.882850  0.697211 -2.064945  ...  1.252967  0.778584 -0.319189
280143 -1.413170  0.248525 -1.127396  ...  0.226138  0.370612  0.028234
280149 -2.234739  1.210158 -0.652250  ...  0.247968  0.751826  0.834108
281144 -2.208002  1.058733 -1.632333  ...  0.306271  0.583276 -0.269209
281674  0.223050 -0.068384  0.577829  ... -0.017652 -0.164350 -0.295135

             V23       V24       V25       V26       V27       V28  Amount
173202 -0.016184  0.612282  0.310599 -0.447180 -0.003685 -0.038164   44.80
7619    0.032352 -0.557525  0.319637  0.107080 -0.047788  0.016557    0.89
279855  0.069729  0.172081  0.979678  0.015022 -0.442540 -0.051430   79.00
271985 -0.056729  0.088720  0.498872 -0.511308 -0.008039 -0.037120   10.03
72054   0.113664  0.361958 -0.106352  0.296521  0.005758  0.147132  100.90
...          ...       ...       ...       ...       ...       ...     ...
279863  0.639419 -0.294885  0.537503  0.788395  0.292680  0.147968  390.00
280143 -0.145640 -0.081049  0.521875  0.739467  0.389152  0.186637    0.76
280149  0.190944  0.032070 -0.739695  0.471111  0.385107  0.194361   77.89
281144 -0.456108 -0.183659 -0.328168  0.606116  0.884876 -0.253700  245.00
281674 -0.072173 -0.450261  0.313267 -0.289617  0.002988 -0.015309   42.53

[984 rows x 30 columns]
```

Type *Markdown* and LaTeX: α^2

We used the matplotlib and seaborn libraries to create a plot visualizing the correlation between the features of a dataset.

In [24]:

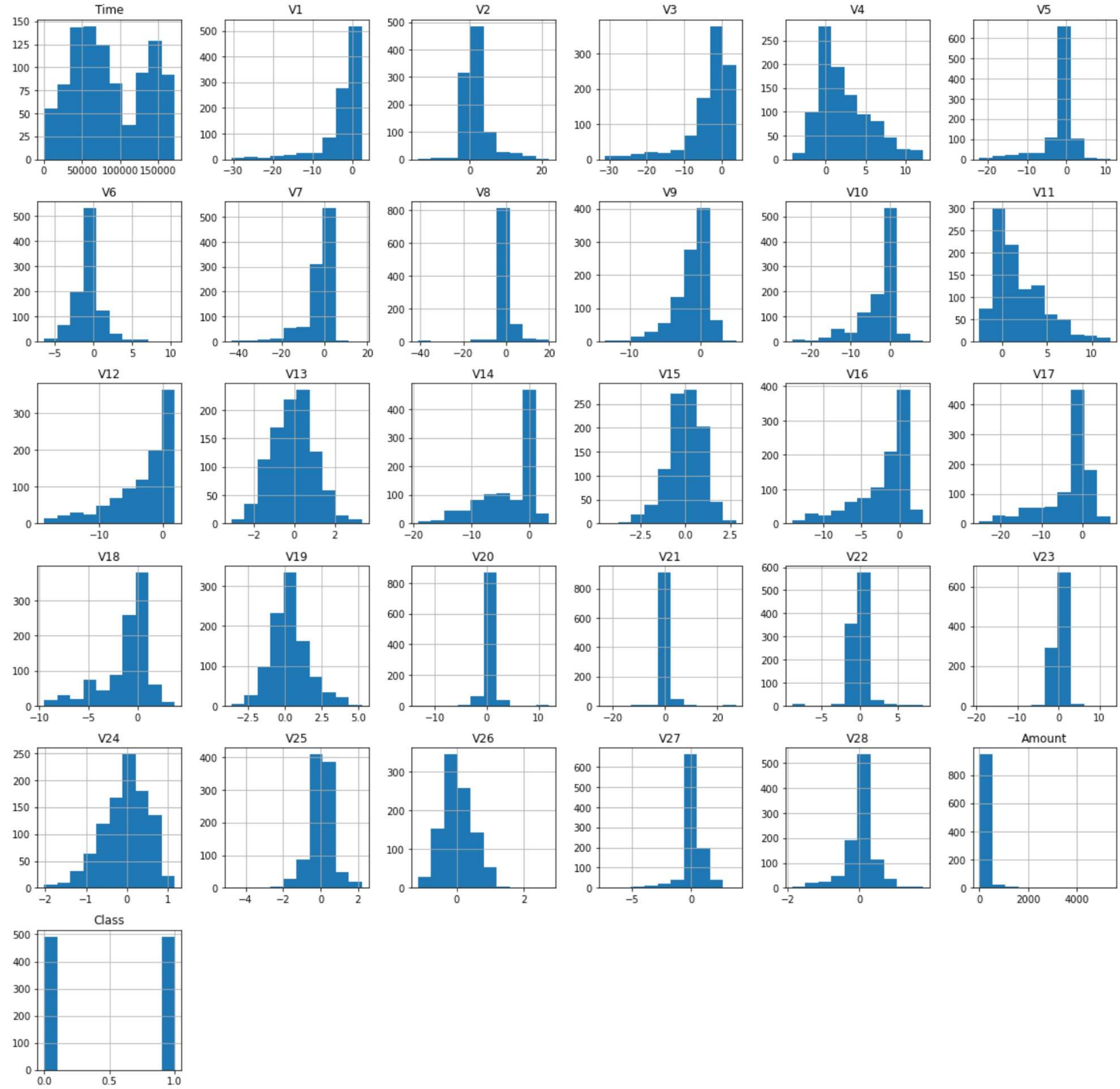
```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(20,20))
plt.title('Credit Card Transactions features correlation plot (Pearson)')
corr = new_dataset.corr()
sns.heatmap(corr,xticklabels=corr.columns,yticklabels=corr.columns,linewidths=.1,cmap="YlGnBu")
plt.show()
```



Splitting the data into features and targets

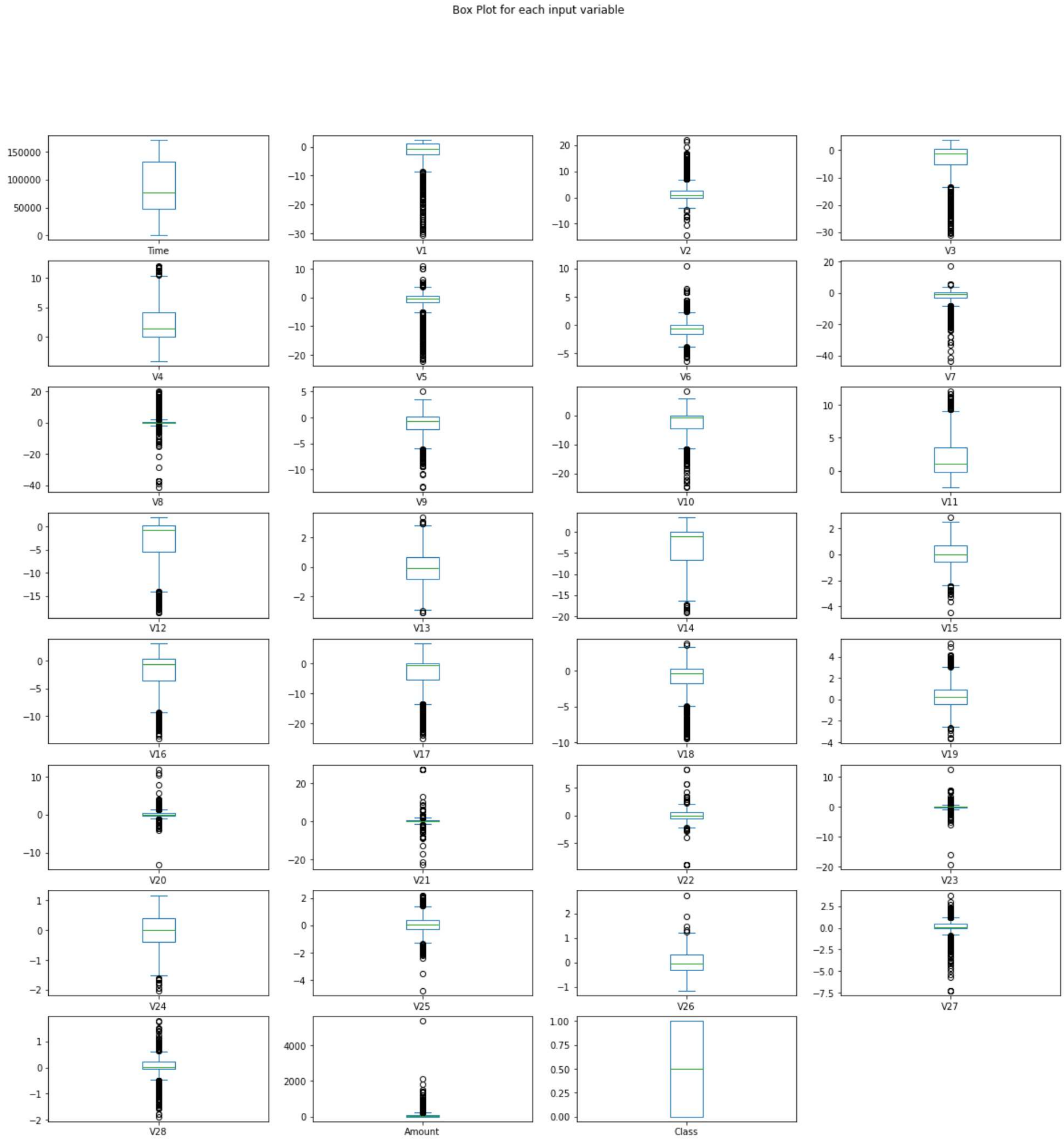
In [25]:

```
# Histogram for all the variables
new_dataset.hist(figsize = (20, 20))
plt.show()
```



In [26]:

```
# box plot for all the variables
new_dataset.plot(kind='box', subplots=True, layout=(8,4), sharex=False, sharey=False, figsize=(20,20), title='Box Plot for each input vari
plt.savefig('creditcard_box')
plt.show()
```



In [27]:

```
# Prepare train, validation, test dataset

# Splitting the data into training and testing data

X = new_dataset.drop(columns = 'Class',axis =1)
Y = new_dataset['Class']
```

In [28]:

```
## normalize numeric variables
## We have divided the dataset into 80% training data and 20% testing data.
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)

print('X_train shape: ', X_train.shape)

print('X_test shape: ', X_test.shape)

print('Y_train shape: ', Y_train.shape)

print('Y_test shape: ', Y_test.shape)
```

```
X_train shape:  (787, 30)
X_test shape:  (197, 30)
Y_train shape:  (787,)
Y_test shape:  (197,)
```

In [29]:

```
print(X)
```

```
[[ 0.73270511  0.77969632 -0.50369123 ... -0.09310331 -0.17261576
  -0.20814817]
 [-1.59927477  0.66393041 -0.37668314 ... -0.13661072 -0.04123192
  -0.37403617]
 [ 1.73730952 -0.02307683 -0.03975405 ... -0.52603272 -0.20446777
  -0.07894368]
 ...
 [ 1.74185381  0.30255384 -0.19926824 ...  0.29043926  0.38566857
  -0.08313716]
 [ 1.75479242 -0.13870174 -0.34746133 ...  0.78345913 -0.69011016
   0.54818926]
 [ 1.76282909  0.78552043 -0.46464129 ... -0.08652096 -0.11774154
  -0.21672403]]
```

In [30]:

```
print(Y)
```

```
173202    0
7619      0
279855    0
271985    0
72054     0
         ..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 984, dtype: int64
```

Splitting the data into training data and test data

In [31]:

```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.2,stratify = Y, random_state=3)
```

In [32]:

```
print(X.shape,X_train.shape,X_test.shape)
```

```
(984, 30) (787, 30) (197, 30)
```

```
## Model Training

1) LogisticRegression
```

In [33]:

```
model=LogisticRegression()
```

In [34]:

```
# training the Logistic Regression model with the training data
model.fit(X_train,Y_train)
```

Out[34]:

```
LogisticRegression()
```

In [35]:

```
#accuracy on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction,Y_train)
```

In [36]:

```
print('Accuracy on Training data : ',training_data_accuracy*100)
```

```
Accuracy on Training data :  95.1715374841169
```

In [37]:

```
#accuracy on test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction,Y_test)
```

In [38]:

```
print('Accuracy on Test data : ',test_data_accuracy*100)
```

```
Accuracy on Test data :  94.9238578680203
```

In [39]:

```
# KNN model

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train,Y_train)
knn.score(X_test,Y_test)

# accuracy on training data

X_train_prediction = knn.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction,Y_train)

print('Accuracy on Training data : ',training_data_accuracy*100)

#accuracy on test data

X_test_prediction = knn.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction,Y_test)

print('Accuracy on Test data : ',test_data_accuracy*100)
```

```
Accuracy on Training data :  93.01143583227446
Accuracy on Test data :  91.87817258883248
```

In [40]:

```
# SVM model

from sklearn import svm
svm_model = svm.SVC(kernel='linear')

# training the Support Vector Machine model with the training data
svm_model.fit(X_train,Y_train)

#accuracy on training data
X_train_prediction = svm_model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction,Y_train)

print('Accuracy on Training data : ',training_data_accuracy*100)

#accuracy on test data
X_test_prediction = svm_model.predict(X_test)

test_data_accuracy = accuracy_score(X_test_prediction,Y_test)

print('Accuracy on Test data : ',test_data_accuracy*100)
```

```
Accuracy on Training data :  95.04447268106735
Accuracy on Test data :  95.43147208121827
```