Project : Credit card Fraud Detection

We are going to build project on The Deep Learning model to detect the Credit Card fraud done by European cardholders in september 2013. We will learn Deep Learning concepts such as Artification Neural Networks,Layers,Activation Functions,Neurons. and top of that we will build Deep Learning Algorithm which mimics humain br ain We analyse data to get meaning information from them and Visualize them to recognise different patterns among them and showing meaning full information It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

Importing neccessary Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

#Over Sampling and under sampling libraries
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler

from keras.models import Sequential
from keras.layers import Dense, Dropout
import warnings
warnings.filterwarnings("ignore")
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/externals/six.py:31: FutureWarning: The module is deprecated in version 0.21 and will be removed in version 0.23 since we've dropped support for Python 2.7. Please rely on the official version of six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:144: FutureWarning: The sklearn.neighbors.base module is  deprecated in version 0.22 and will be removed in version 0.24. The corresponding classes / functions should instead be imported from sklearn.neighbors. Anything that cannot be imported from sklearn.neighbors is now part of the private API.
  warnings.warn(message, FutureWarning)
```

Importing Data set

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions. It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. 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,

In [2]:
```
path="/content/drive/MyDrive/Ml\AI/Deep_Learning_Itvedant/Project DL/creditcard.csv"
df=pd.read_csv(path)
df.head()
```

Out[2]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.991
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.489
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.717
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.507
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.345

In [3]:
```
df.shape
```

Out[3]:
```
(284807, 31)
```

In [4]:
```
df.size
```

Out[4]:
```
8829017
```

In [5]:
```
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 [6]:
```
df.describe()
```

Out[6]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	3.919560e-15	5.688174e-16	-8.769071e-15	2.782312e-15	-1.552563e-15	2.010663e-15	-1.694249e-15	-1.927028e-16
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01

In [7]:
```
# compare the values for both transactions
df.groupby('Class').mean()
```

Out[7]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	
Class													
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.000987	0.004467	0.009824	-0.006576	0.01
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	-5.676883	3.800173	-6.25

Checking missing values

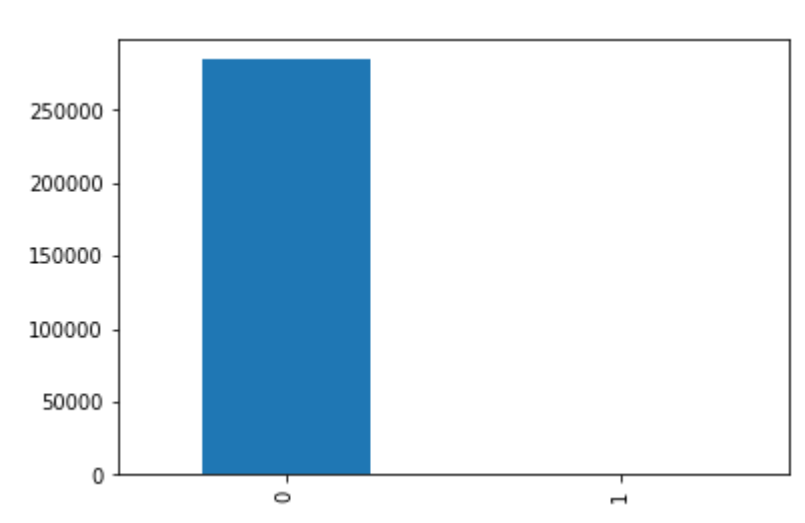
In [8]:
```
df.isna().sum()
```

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
```

Data Visualization

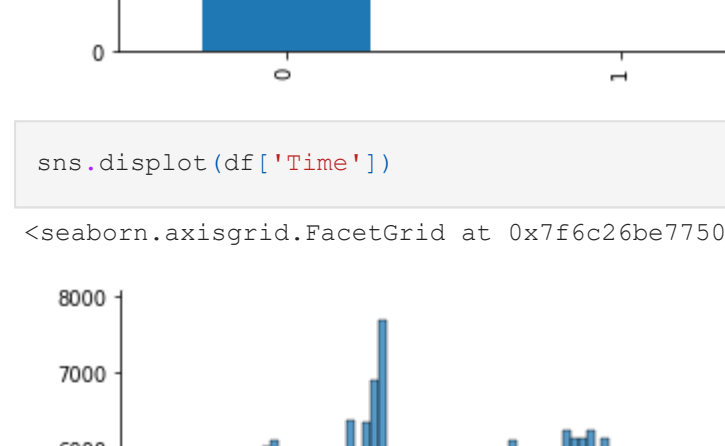
In [33]:
```
df['Class'].value_counts().plot(kind='bar')
```

Out[33]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f128ddebad0>
```



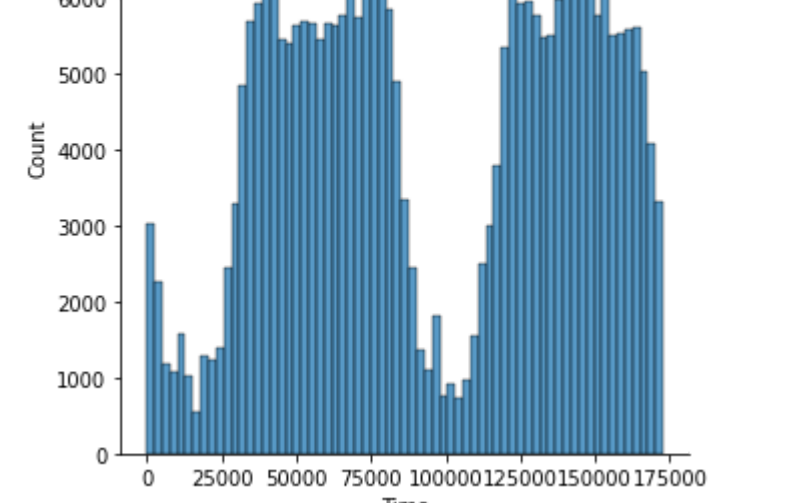
In [ ]:
```
sns.displot(df['Time'])
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x7f6c26be7750>
```



In [ ]:
```
sns.scatterplot(x='Time',y='Amount',hue='Class',data=df)
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6c263ea450>
```



In [ ]:
```
plt.figure(figsize=(28,18))
sns.heatmap(df.corr(),annot=True)
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6c26349e90>
```

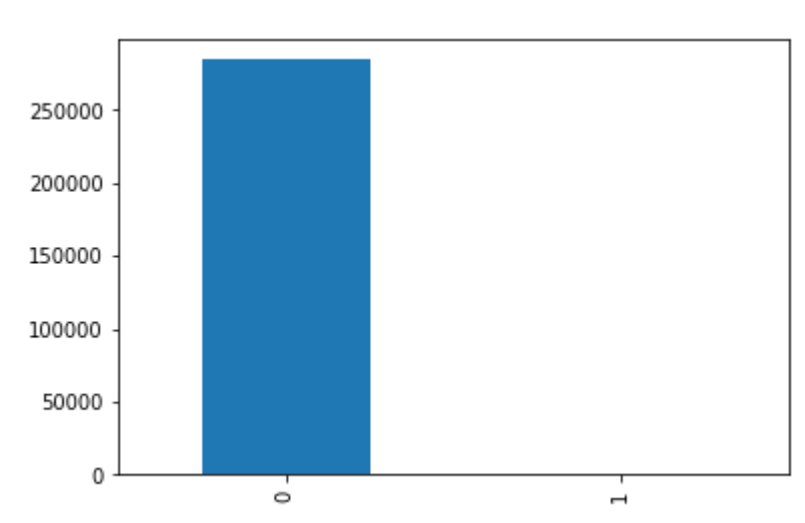
Checking class Distribution

In [9]:
```
df["Class"].value_counts()
```

Out[9]:
```
0    284315
1       492
Name: Class, dtype: int64
```

In [10]:
```
df['Class'].value_counts().plot(kind='bar')
```

Out[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1291e2c850>
```



Dividing independant Variable and Target Variable

In [11]:
```
x = df.drop(["Class","Time"],axis=1)
y = df["Class"]
x
```

Out[11]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.99
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.48
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.71
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.50
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.34
...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	4.356170	-1.593105	2.711941	-0.68
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	-0.975926	-0.150189	0.915802	1.21
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	-0.484782	0.411614	0.063119	-0.18
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	-0.399126	-1.933849	-0.962886	-1.04
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	-0.915427	-1.040458	-0.031513	-0.18

284807 rows × 29 columns

In [12]:
```
y
```

Out[12]:
```
0         0
1         0
2         0
3         0
4         0
         ..
284802    0
284803    0
284804    0
284805    0
284806    0
Name: Class, Length: 284807, dtype: int64
```

In [13]:
```
# sc=StandardScaler()
# scaled_Amount=pd.DataFrame(df["Amount"])

# scaled_Amount=pd.DataFrame(sc.fit_transform(scaled_Amount))
# scaled_Amount
```

In [14]:
```
# x=pd.concat([x,scaled_Amount],axis=1)
# x
```

Splitting Data Into Train and Test Data

In [15]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=1)
```

In [16]:
```
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(random_state=1)
x_train_bal, y_train_bal = rus.fit_resample(x_train, y_train)
pd.Series(y_train_bal).value_counts()
```

Out[16]:
```
1    381
0    381
dtype: int64
```

Scaling

In [17]:
```
sc=StandardScaler()
x_train_ss=sc.fit_transform(x_train_bal)
x_test_ss=sc.transform(x_test)
x_train_ss
```

Out[17]:
```
array([[ 1.38543605e-01, -1.20612787e-01,  5.43003906e-01, ...,
        -3.48398357e-02, -1.05010852e-03, -4.41048280e-01],
       [ 1.58481250e-01, -1.53955277e-01,  6.37894043e-01, ...,
         2.69919264e-01, -5.66765190e-01, -4.42669777e-01],
       [ 2.60320118e-01, -4.89798488e-01,  6.95967075e-01, ...,
        -1.26601442e-01, -4.54569552e-03, -3.17131748e-01],
       ...,
       [ 6.48799392e-01,  2.63492459e-01, -7.02782645e-02, ...,
        -4.82815815e-02,  2.94618432e-01, -4.55471072e-01],
       [-3.79288956e-01,  4.78857964e-01, -2.65053220e-01, ...,
        -2.24940145e-01, -2.59406057e-01, -4.55471072e-01],
       [-1.21833684e-01, -4.36079305e-01,  3.13967005e-02, ...,
         7.27333757e-01,  4.48796292e-01,  1.35975251e+00]])
```

Neural Network

In [18]:
```
model=Sequential()
model.add(Dense(32,activation='relu',input_dim=29))
model.add(Dense(64,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dropout(0.2))



model.add(Dense(1,activation='sigmoid'))
```

In [19]:
```
model.compile(optimizer="adam",loss="binary_crossentropy")
```

In [20]:
```
trained_model = model.fit(x_train_ss,y_train_bal,epochs=25,batch_size=50)
```

```
Epoch 1/25
16/16 [==============================] - 1s 2ms/step - loss: 0.6150
Epoch 2/25
16/16 [==============================] - 0s 2ms/step - loss: 0.4241
Epoch 3/25
16/16 [==============================] - 0s 2ms/step - loss: 0.3149
Epoch 4/25
16/16 [==============================] - 0s 2ms/step - loss: 0.2442
Epoch 5/25
16/16 [==============================] - 0s 2ms/step - loss: 0.2073
Epoch 6/25
16/16 [==============================] - 0s 2ms/step - loss: 0.1852
Epoch 7/25
16/16 [==============================] - 0s 2ms/step - loss: 0.1732
Epoch 8/25
16/16 [==============================] - 0s 2ms/step - loss: 0.1583
Epoch 9/25
16/16 [==============================] - 0s 2ms/step - loss: 0.1535
Epoch 10/25
16/16 [==============================] - 0s 2ms/step - loss: 0.1429
Epoch 11/25
16/16 [==============================] - 0s 2ms/step - loss: 0.1348
Epoch 12/25
16/16 [==============================] - 0s 2ms/step - loss: 0.1268
Epoch 13/25
16/16 [==============================] - 0s 2ms/step - loss: 0.1291
Epoch 14/25
16/16 [==============================] - 0s 3ms/step - loss: 0.1169
Epoch 15/25
16/16 [==============================] - 0s 2ms/step - loss: 0.1197
Epoch 16/25
16/16 [==============================] - 0s 2ms/step - loss: 0.1180
Epoch 17/25
16/16 [==============================] - 0s 2ms/step - loss: 0.1129
Epoch 18/25
16/16 [==============================] - 0s 2ms/step - loss: 0.1130
Epoch 19/25
16/16 [==============================] - 0s 2ms/step - loss: 0.1090
Epoch 20/25
16/16 [==============================] - 0s 3ms/step - loss: 0.0993
Epoch 21/25
16/16 [==============================] - 0s 2ms/step - loss: 0.0972
Epoch 22/25
16/16 [==============================] - 0s 2ms/step - loss: 0.0894
Epoch 23/25
16/16 [==============================] - 0s 2ms/step - loss: 0.0931
Epoch 24/25
16/16 [==============================] - 0s 2ms/step - loss: 0.0862
Epoch 25/25
16/16 [==============================] - 0s 2ms/step - loss: 0.0896
```

In [21]:
```
plt.plot(trained_model.history['loss'])
```

Out[21]:
```
[<matplotlib.lines.Line2D at 0x7f128c580b50>]
```

Model Prediction

In [22]:
```
y_pred = model.predict(x_test_ss)
```

In [23]:
```
y_pred = np.where(y_pred >= 0.5,1,0)
y_pred
```

Out[23]:
```
array([[0],
       [0],
       [0],
       ...,
       [0],
       [0],
       [0]])
```

Evaluation Of Model

In [24]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.97      0.98     71091
           1       0.04      0.86      0.07       111

    accuracy                           0.96     71202
   macro avg       0.52      0.92      0.53     71202
weighted avg       1.00      0.96      0.98     71202
```

In [25]:
```
# from imblearn.over_sampling import RandomOverSampler
# ros = RandomOverSampler(random_state=1)
# xtrain_bal1, ytrain_bal1 = ros.fit_resample(x_train, y_train)
```

In [26]:
```
# pd.Series(ytrain_bal1).value_counts()
```

In [27]:
```
# sc=StandardScaler()
# x_train_ss1=sc.fit_transform(xtrain_bal1)
# x_test_ss1=sc.transform(x_test)
# x_train_ss1
```

In [28]:
```
#  model1=Sequential()
# model1.add(Dense(32,activation='relu',input_dim=30))
# model1.add(Dense(64,activation='relu'))
# model1.add(Dense(64,activation='relu'))


# model1.add(Dense(1,activation='sigmoid'))
```

In [29]:
```
# model1.compile(optimizer="adam",loss="binary_crossentropy")
```

In [30]:
```
# trained_model1 = model1.fit(x_train_ss1,ytrain_bal1,epochs=25,batch_size=50)
```

In [31]:
```
# y_pred1 = model.predict(x_test_ss1)
# y_pred1 = np.where(y_pred >= 0.5,1,0)
# print(classification_report(y_test,y_pred1))
```