Credit Card Frauds Detection Using machine Learning🎯 🔎

Dataset Information

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, 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.

Given the class imbalance ratio, we recommend measuring the accuracy using the Area Under the Precision-Recall Curve (AUPRC). Confusion matrix accuracy is not meaningful for unbalanced classification.

Import modules

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

1. Data Collection🎯 📚

In [2]:
```
df = pd.read_csv("creditcard.csv")
```

Ask some basic question about your data

a. how big the data?

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

Out[3]: (284807, 31)

b. how the data look like?

In [4]:
```
df.head(5)
```

Out[4]:

	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	0.6
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 [5]:
```
df.sample(5)
```

Out[5]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	
113509	73094.0	-2.097985	-1.164713	1.871078	-1.357878	-1.035937	0.901266	-0.497693	1.021664	-0.902012	...	0.346244	0.452661	0.311866	-0.29
49250	43974.0	1.346923	-1.023541	0.036388	-1.266140	-1.224028	-1.004749	-0.453017	-0.282107	-2.220760	...	-0.415028	-0.979059	0.162147	0.35
239437	150108.0	1.893253	0.426856	-0.157178	3.532845	0.521660	1.049874	-0.319696	0.142693	-0.686691	...	-0.216662	-0.590834	0.360117	-0.01
122012	76402.0	1.179768	0.265222	0.722161	0.689779	-0.492357	-0.811274	0.042268	-0.140118	-0.098995	...	-0.186806	-0.475905	0.223435	0.61
79318	57967.0	-3.176619	2.383538	-2.549888	-0.954973	0.773420	3.638557	-2.233492	-0.066616	-0.900788	...	-1.500868	-0.685052	0.502935	0.91

5 rows × 31 columns

2. Data Preprocessing 🔎 ✨

c. Understand data type of any columns

In [6]:
```
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
```

d. is there any null values present or not?

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
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
```

e. highly mathematicla information about your data

In [8]:
```
df.describe()
```

Out[8]:

	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	2.848070e+
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1.213481e-16	-2.406331e-
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	1.098632e+
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	-1.343407e+
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	-6.430976e-
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	-5.142873e-
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	5.971390e-
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	1.559499e+

8 rows × 31 columns

In [9]:
```
# show all columns
df.columns
```

Out[9]:
```
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')
```

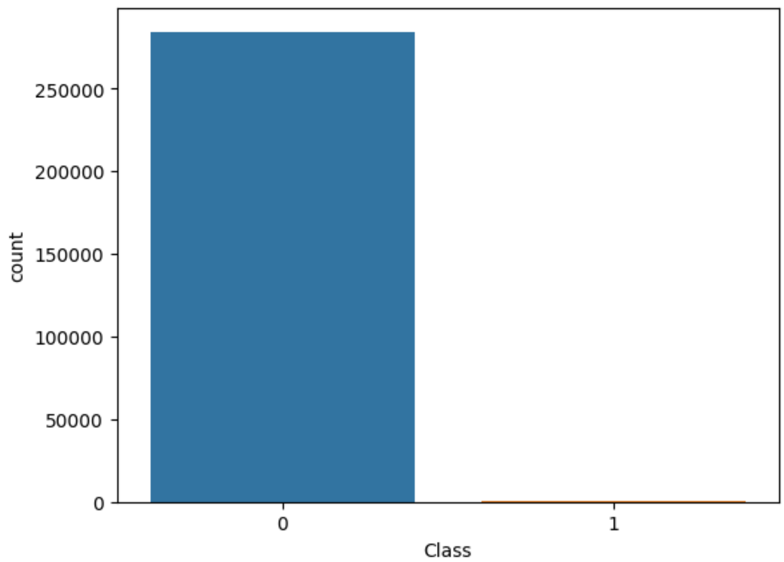
EDA(Exploratoty Data Analyis) perform in the dataset

In [10]:
```
df["Class"].value_counts()
```

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

In [11]:
```
sns.countplot(data = df,x = df["Class"])
```

Out[11]:
```
&lt;AxesSubplot: xlabel='Class', ylabel='count'&gt;
```



In [12]:
```
legit = df[df.Class ==0]
fraud = df[df.Class==1]
```

In [13]:
```
legit
```

Out[13]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	
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	0.0
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	-0.3
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	-0.6
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	-1.1
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	0.1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
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	-0.5
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	-1.0
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	0.6
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	0.1
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	0.0

284315 rows × 31 columns

In [14]:
```
fraud
```

Out[14]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	
541	406.0	-2.312227	1.951992	-1.609851	3.997906	-0.522188	-1.426545	-2.537387	1.391657	-2.770089	...	0.517232	-0.035049	-0.465211	0.32
623	472.0	-3.043541	-3.157307	1.088463	2.288644	1.359805	-1.064823	0.325574	-0.067794	-0.270953	...	0.661696	0.435477	1.375966	-0.29
4920	4462.0	-2.303350	1.759247	-0.359745	2.330243	-0.821628	-0.075788	0.562320	-0.399147	-0.238253	...	-0.294166	-0.932391	0.172726	-0.08
6108	6986.0	-4.397974	1.358367	-2.592844	2.679787	-1.128131	-1.706536	-3.496197	-0.248778	-0.247768	...	0.573574	0.176968	-0.436207	-0.05
6329	7519.0	1.234235	3.019740	-4.304597	4.732795	3.624201	-1.357746	1.713445	-0.496358	-1.282858	...	-0.379068	-0.704181	-0.656805	-1.63
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
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

492 rows × 31 columns

In [15]:
```
legit.shape
```

Out[15]: (284315, 31)

In [16]:
```
fraud.shape
```

Out[16]: (492, 31)

In [17]:
```
legit = legit.sample(492)
```

In [18]:
```
legit.shape
```

Out[18]: (492, 31)

In [19]:
```
df = pd.concat([legit,fraud],axis = 0)
```

In [20]:
```
df.shape
```

Out[20]: (984, 31)

In [21]:
```
df["Class"].value_counts()
```

Out[21]:
```
0    492
1    492
Name: Class, dtype: int64
```

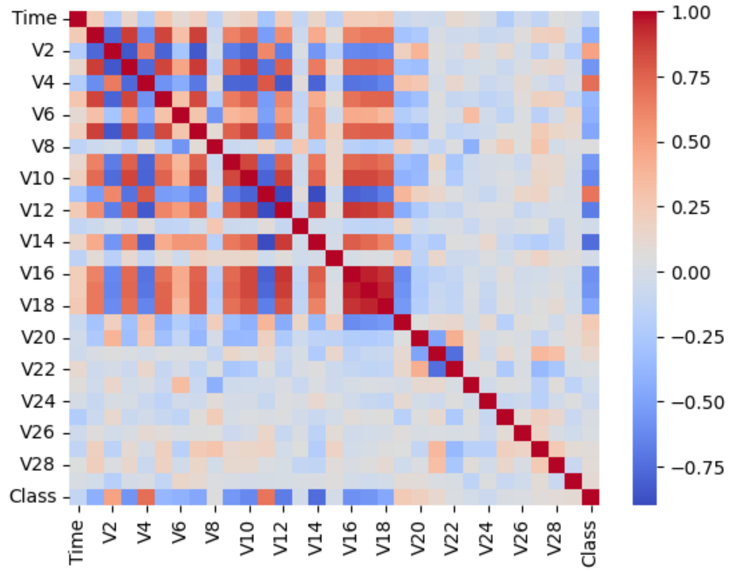
In [22]:
```
df.groupby("Class").mean()
```

Out[22]:

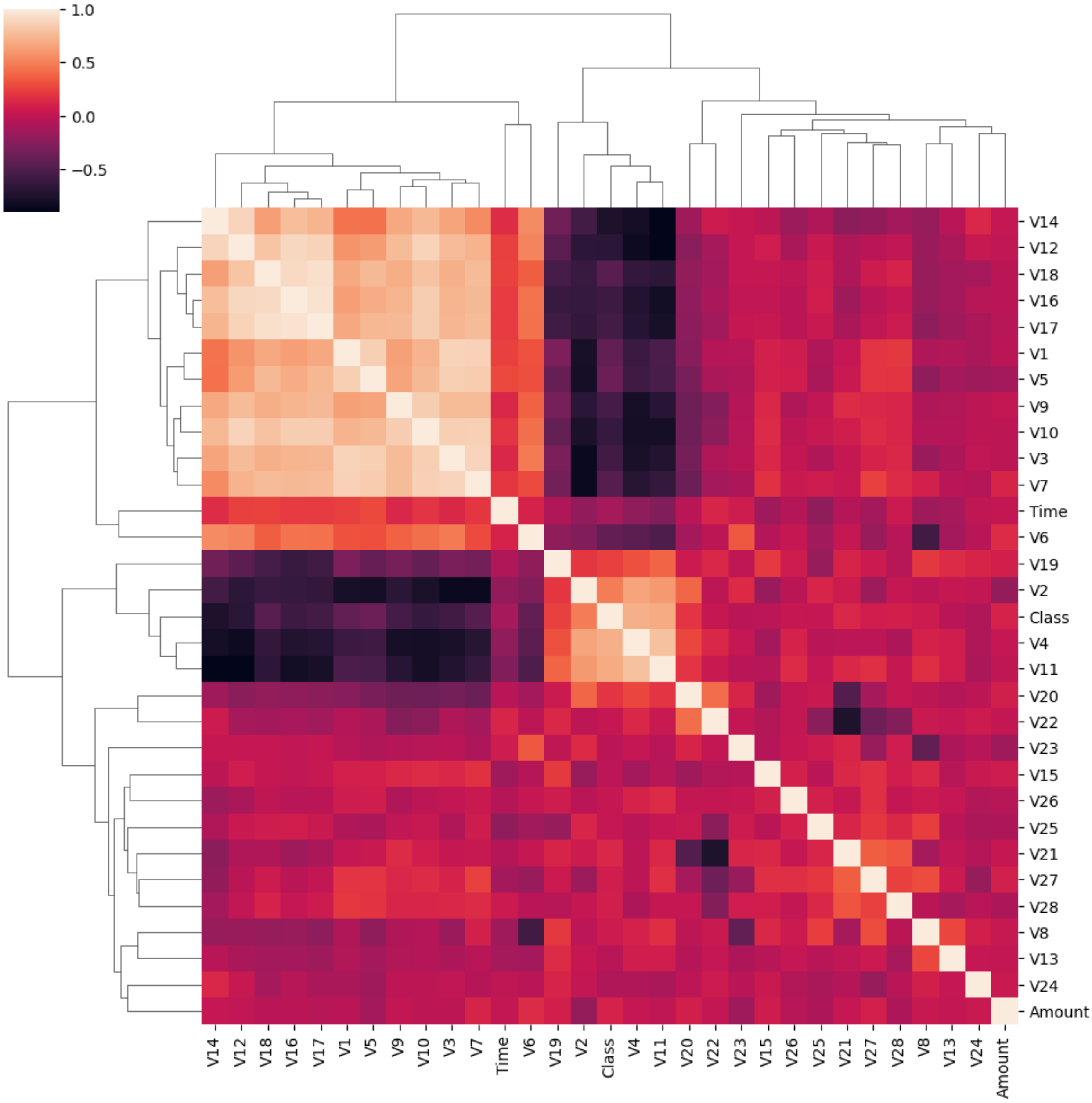
V5	V6	V7	V8	V9	...	V20	V21	V22	V23	V24	V25	V26	V27	V28	Am
-0.008025	0.024056	-0.033637	0.032243	0.025884	...	-0.076983	-0.010141	-0.049904	0.026096	-0.02090	0.016342	0.029892	0.010050	0.002546	75.13
-3.151225	-1.397737	-5.568731	0.570636	-2.581123	...	0.372319	0.713588	0.014049	-0.040308	-0.10513	0.041449	0.051648	0.170575	0.075667	122.21

In [23]:
```
sns.heatmap(df.corr(),cmap="coolwarm")
```

Out[23]: <AxesSubplot: >



In [24]:
```
sns.clustermap(df.corr())
```
Out[24]: <seaborn.matrix.ClusterGrid at 0x14562397af0>



In [25]:
```
x = df.drop("Class",axis = "columns")
```
In [26]:
```
y = df["Class"]
```
In [27]:
```
x.shape
```
Out[27]: (984, 30)

In [28]:
```
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=2)
```
In [29]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(x_train,y_train)
```
Out[29]:
```
▾ LogisticRegression
LogisticRegression()
```
In [30]:
```
y_pred = model.predict(x_test)
```
In [31]:
```
from sklearn.metrics import accuracy_score
```

In [32]:
```python
print(f"accuracy score of this model is {accuracy_score(y_test,y_pred)}")
```

```
accuracy score of this model is 0.934010152284264
```

In [ ]:

In [37]:
```python
input_data = (91595.506098,-0.084798,0.016967,0.072183,-0.007679,-0.008025,0.024056,-0.033637,0.032243,0.025884,-0.076983,-0
input_data_as_numpy_array = np.asarray(input_data)
input_data_reshape = input_data_as_numpy_array.reshape(1,-1)
prediction = model.predict(input_data_reshape)
print(prediction)
if(prediction[0]==1):
    print("Fraud Transection")
else:
    print("NOt Fraud")
```

```
[0]
NOt Fraud
```

In [ ]:

In [ ]:

In [ ]: