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.

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import warnings
warnings.filterwarnings("ignore")
```

In [164…

```
df=pd.read_csv("creditcard.csv")
df.head()
```

Out[164]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0

5 rows × 31 columns

In [165…

```
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 [166…

```
df.describe()
```

Out[166]:

	Time	V1	V2	V3	V4	V5	V(
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+0
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-1
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+0
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+0
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-0
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-0
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-0
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+0

8 rows × 31 columns

In [167…

```
df.isnull().sum()
```

Out[167]:
```
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 [168…
```
df["Class"].value_counts()
```

Out[168]:
```
0    284315
1       492
Name: Class, dtype: int64
```

above we see class columns having not balanced data distribution This Dataset is highly unblanced

0 --> Normal Transaction

1 --> fraudulent transaction

In [169…
```
legit=df[df["Class"]==0]
fraud=df[df["Class"]==1]
```

In [170…
```
print(legit.shape)
print(fraud.shape)
```

```
(284315, 31)
(492, 31)
```

In [171…
```
legit.value_counts().sum()
```

Out[171]:
```
284315
```

In [172…
```
fraud.value_counts().sum()
```

Out[172]:
```
492
```

In [173…

```
#statistical infomation about class with respect to amount
legit.Amount.describe()
```

Out[173]:

```
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 [174…

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

Out[174]:

```
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 [175…

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

In [176…

```
legit_sample.Amount.describe()
```

Out[176]:

```
count     492.000000
mean       81.803171
std       281.646188
min         0.000000
25%         4.957500
50%        19.995000
75%        64.162500
max      4959.850000
Name: Amount, dtype: float64
```

In above output we see there is not much more difference in output of legit_sample and legit

In [177…

```
#now concating the fraud and legit_sample together rowise
```

In [178…

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

In [179…

```
df_new.head()
```

Out[179]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
51948	45204.0	-0.742751	0.475642	1.266253	0.514362	-0.310715	1.112718	1.079678	0.026520	0.3491
72090	54560.0	1.086081	-1.076948	0.735946	-0.785195	-1.146546	0.315644	-1.000523	0.164404	-0.7800
187130	127399.0	1.910441	-0.751755	-0.554896	0.148276	-0.705147	-0.123455	-0.740698	0.123589	1.7253
35402	38105.0	-1.433971	1.372233	0.588119	-0.603508	1.311183	0.268878	1.288159	-0.558558	0.9129
134114	80652.0	1.154212	0.221124	0.541786	1.316365	-0.220439	-0.207687	0.006521	0.002216	0.2139

5 rows × 31 columns

In [180…

```
df_new.tail()
```

Out[180]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.697211	-2.06494
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.248525	-1.12739
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.210158	-0.65225
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.058733	-1.63233
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.068384	0.57782

5 rows × 31 columns

In [181…

```
df_new["Class"].value_counts()
```

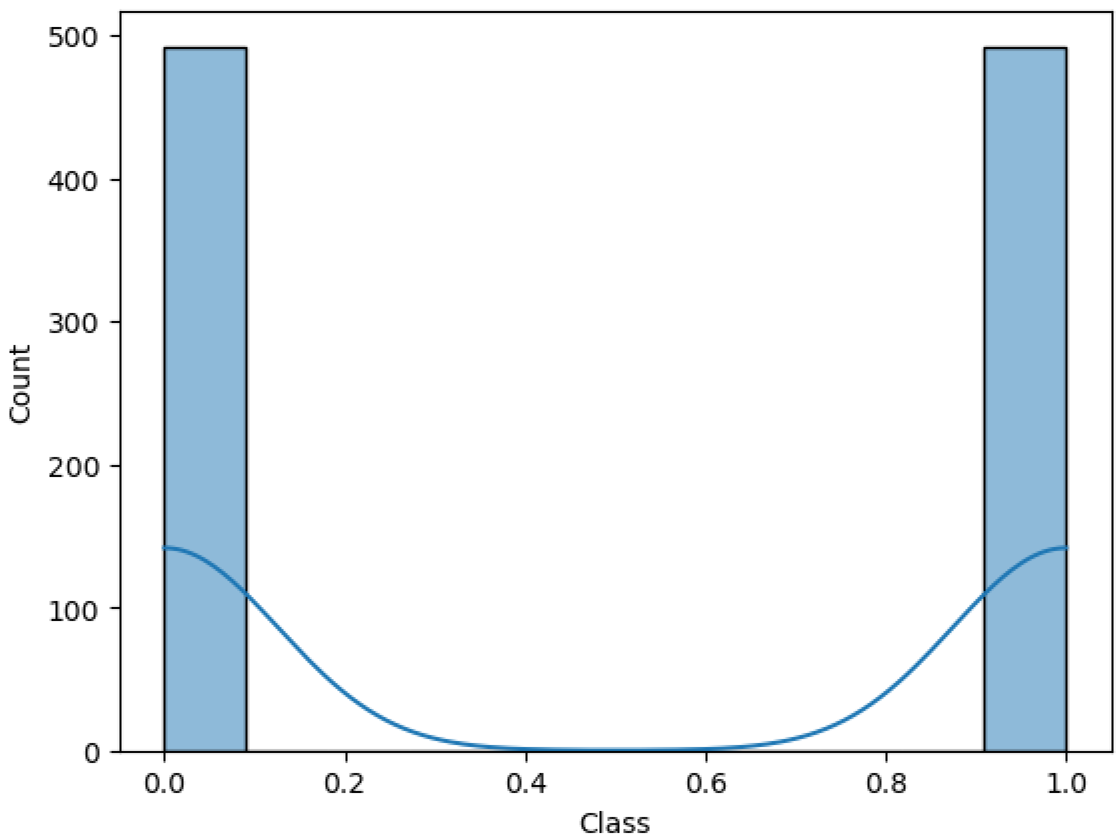
Out[181]:

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

In [182…

```
#visualization of class distribution of df_new
sns.histplot(df_new["Class"],kde=True)
```

Out[182]: <Axes: xlabel='Class', ylabel='Count'>



In [183…

```
df_new.head(2)
```

Out[183]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
51948	45204.0	-0.742751	0.475642	1.266253	0.514362	-0.310715	1.112718	1.079678	0.026520	0.349170
72090	54560.0	1.086081	-1.076948	0.735946	-0.785195	-1.146546	0.315644	-1.000523	0.164404	-0.780097

2 rows × 31 columns

In [184…

```
x=df_new.drop(["Time","Class"],axis=1)
y=df_new["Class"]
```

In [185…
```
y.value_counts()
```
Out[185]:
```
0    492
1    492
Name: Class, dtype: int64
```

In [186…
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=42)
```

In [187…
```
#logistic regression modeling

lr=LogisticRegression()
lr.fit(x_train,y_train)
```
Out[187]:
```
▼ LogisticRegression
LogisticRegression()
```

In [188…
```
y_train_pred=lr.predict(x_train)
```

Evalution of model

In [189…
```
from sklearn.metrics import accuracy_score
```

In [190…
```
accuracy_score_train= accuracy_score(y_train,y_train_pred)
accuracy_score_train
```
Out[190]:
```
0.9552845528455285
```

In [191…
```
y_test_pred=lr.predict(x_test)
```

In [193…
```
accuracy_score_test=accuracy_score(y_test,y_test_pred)
accuracy_score_test
```
Out[193]:
```
0.9105691056910569
```

Cross val checking

In [196…
```
from sklearn.model_selection import cross_val_score
cvs=cross_val_score(lr,x,y,cv=200)
```

In [276…
```
cv_1=np.mean(cvs)
cv_1
```
Out[276]:
```
0.93125
```

In [262…
```
```
Out[262]:
```
0.93125
```

In [202…
```
from sklearn.naive_bayes import GaussianNB
gn=GaussianNB()
gn.fit(x_train,y_train)
```
Out[202]:
```
▼ GaussianNB
GaussianNB()
```

Evalution of model

In [211…
```
y_train_pred=gn.predict(x_train)
```

In [212…
```
train_accuracy_score=accuracy_score(y_train,y_train_pred)
train_accuracy_score
```

Out[212]:
```
0.9254742547425474
```

In [213…
```
y_test_pred=gn.predict(x_test)
```

In [214…
```
test_accuracy_score=accuracy_score(y_test,y_test_pred)
test_accuracy_score
```

Out[214]:
```
0.8861788617886179
```

In [215…
```
#checking cross value score
cv1=cross_val_score(gn,x,y,cv=100)
```

In [275…
```
cv_2=np.mean(cv1)
cv_2
```

Out[275]:
```
0.9007777777777777
```

In [ ]:
```

```

In [219…
```
#checking accuracy by confusion matrix

from sklearn.metrics import confusion_matrix
cnn=confusion_matrix(y_test,y_test_pred)
cnn
```

Out[219]:
```
array([[119,   7],
       [ 21,  99]], dtype=int64)
```

In [220…
```
199+99
```

Out[220]:
```
298
```

In [221…
```
298+28
```

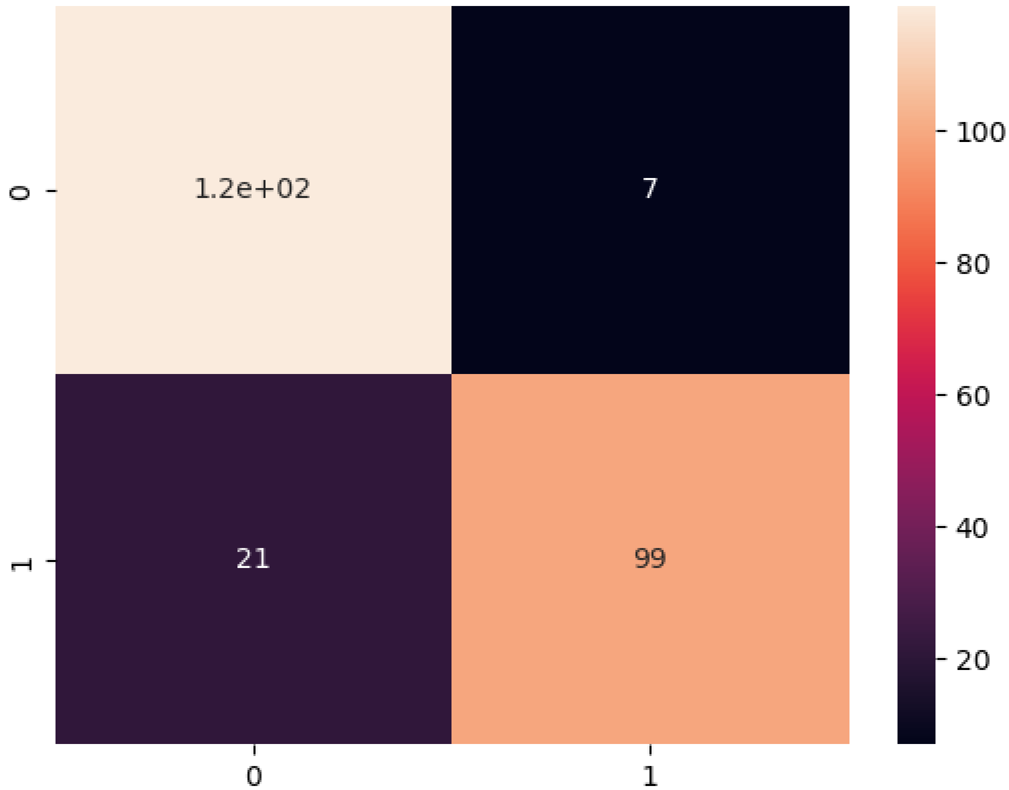
Out[221]:
```
326
```

In [222…
```
298/326
```

Out[222]:
```
0.9141104294478528
```

In [224…
```
sns.heatmap(cnn,annot=True)
```

Out[224]:
```
<Axes: >
```



In [225…
```
from sklearn import svm
```

In [227…
```
clf=svm.SVC()
clf.fit(x_train,y_train)
```

Out[227]:
```
▼ SVC
SVC()
```

In [228…
```
y_train_pred=clf.predict(x_train)
```

In [229…
```
accuracy_score_train=accuracy_score(y_train,y_train_pred)
accuracy_score_train
```

Out[229]:
```
0.7913279132791328
```

In [230…
```
y_test_pred=clf.predict(x_test)
```

In [232…
```
accuracy_score_test=accuracy_score(y_test,y_test_pred)
accuracy_score_test
```

Out[232]:
```
0.8089430894308943
```

In [233…
```
cnn=confusion_matrix(y_test,y_test_pred)
cnn
```

Out[233]:
```
array([[119,   7],
       [ 40,  80]], dtype=int64)
```

In [235…
```
print(119+80)
print(40+7)
print(199+47)
```

```
199
47
246
```

In [236…

```
199/246
```

Out[236]:

```
0.8089430894308943
```

checking cross val score

In [237…

```
cv2=cross_val_score(clf,x,y,cv=100)
```

In [265…

```
cv_3=np.mean(cv2)
cv_3
```

Out[265]:

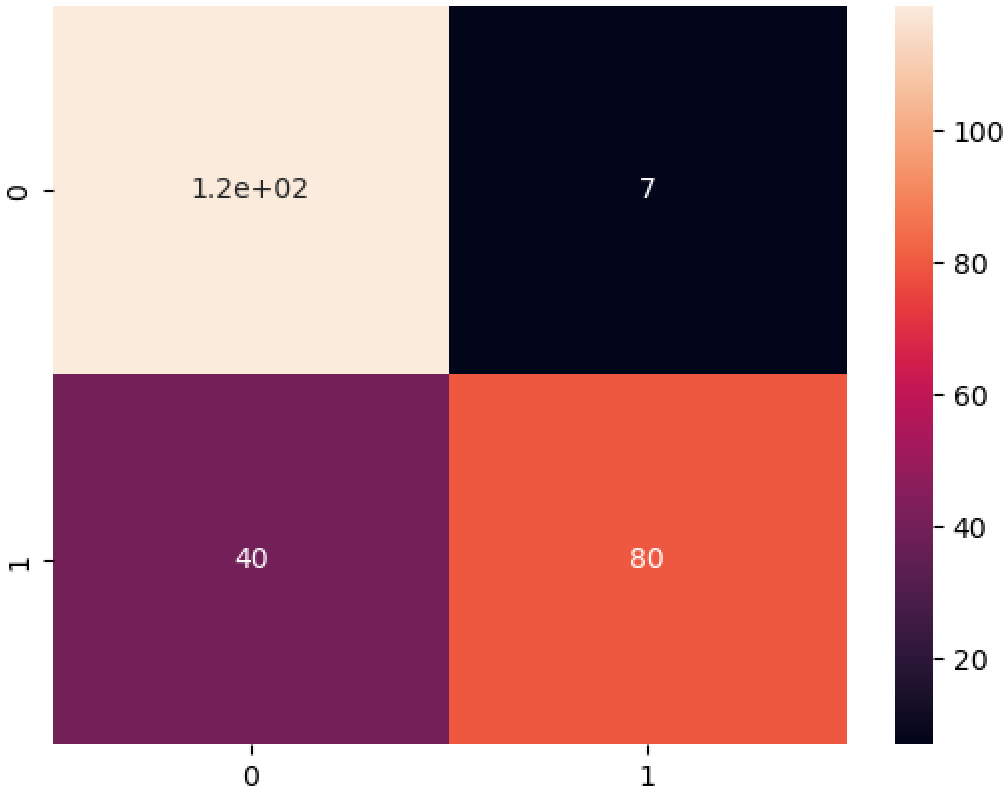
```
0.8141111111111111
```

In [239…

```
sns.heatmap(cnn,annot=True)
```

Out[239]:

```
<Axes: >
```



In [281…

```
train_score={"logistic_regression":accuracy_score_train ,"Naive Bayes":train_accuracy_sc
test_score={"logistic_regression":accuracy_score_test ,"Naive Bayes":test_accuracy_score
```

In [285…

```
print("Train score:",train_score)
print("Test score: ", test_score)
```

```
Train score: {'logistic_regression': 0.7913279132791328, 'Naive Bayes': 0.92547425474254
74, 'SVM': 0.93125}
Test score:  {'logistic_regression': 0.8089430894308943, 'Naive Bayes': 0.88617886178861
79, 'SVM': 0.9007777777777777}
```

from above we conclude that SVM(Support vetor machine) give more accuracy on test data so we can say that we use SVM for credit card fraud Detection

In [ ]: